The Exploration of an Integrated Representation for the Conceptual Phase of Structural Design for Tall Buildings Through Distributed Multi-Reasoning Algorithms

by

Lucio Soibelman

Engenheiro Civil, 1985
Universidade Federal do Rio Grande do Sul

Master of Science in Civil Engineering, 1993
Universidade Federal do Rio Grande do Sul

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Civil Engineering Systems

at the

Massachusetts Institute of Technology

April 1998

Author ______________________________

Lucio Soibelman
April 29, 1998

Certified by ______________________________

Feniosky Peña-Mora
Assistant Professor of Civil and Environmental Engineering, Thesis Supervisor

Accepted by ______________________________

Joseph M.Sussman
Chairman, Departmental Committee on Graduate Students

The Exploration of an Integrated Representation for the Conceptual Phase of Structural Design for Tall Buildings Through Distributed Multi-Reasoning Algorithms

By
Lucio Soibelman

Submitted to the Department of Civil and Environmental Engineering
on April 29, 1998, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Civil Engineering Systems

Abstract

The design process is an information processing activity. More detailed information about the task itself, about the constraints, about possible solutions principles, and about known solutions for similar problems is extremely useful in the process of defining the problem and finding a solution to the design problem. This research presents the M-RAM (Multi-Reasoning Artificial Mind) model that aim to assist engineers in the conceptual phase of the structural design of tall buildings by providing him/her with organized and reliable information. This preliminary conceptual design involves selecting preliminary materials, selecting the overall structural form of the building, producing a rough dimensional layout, and considering technological possibilities. Decisions are made on the basis of such information as height of the building, building use, typical live load, wind velocity, earthquake loading, design fundamental period, design acceleration, maximum lateral deflection, spans, story height, and other client requirements. The M-RAM objective is to provide designers with adapted past design solutions with the help of a distributed multi-reasoning mechanism creating a support system to enhance creativity, engineering knowledge and experience of designers. To test the feasibility of the proposed model a prototype of a distributed artificial intelligence system was developed where the Internet was used as a communication backbone among the different systems that implemented the reasoning mechanisms employed. Each different reasoning mechanism was considered as an autonomous module acting as an intelligent agent supporting the design process.

Thesis Supervisor: Feniosky Peña-Mora
Title: Assistant Professor of Civil and Environmental Engineering

Acknowledgments

I wish to thank:

- Professor Feniosky Peña-Mora for his friendship, support and guidance during these five years at MIT. His support is present in every part of this thesis.

- Professor Robert D. Logcher for his useful comments and insights both in the classroom and in the IESL group, and more importantly, for allowing me to be his apprentice.

- All the members of the doctoral committee for their valuable support and feedback during this process: Prof. Connor, Prof. Williams, Prof. Logcher, and Prof. Peña-Mora.

- Professor Rafael L.Bras for his friendship, support and guidance.

- Karim Hussein, for his friendship and the several discussions we have made that has brought new interesting ideas.

- Professor Jayachandran Parnnasivam for his invaluable explanations, advice and time.

- Pedro Ledesma for his help in creating the database home pages.

- To my mother, Cecilia Soibelman, for having taught me the real value of education.
- To my father, Jayme Soibelman, and my uncle, Isaac Soibelman, for teaching me everything that I know about engineering practice.
- CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) for the financial support.
- Joan McCusker, Patricia Dixon, Elaine Healy, Cynthia Stewart, Jessie Williamson, Stephanie George, Danielle Severino, Callum Valleli, Karen Horton, and Patricia Vargas for their help in all the administrative matters.
- All Da Vinci Initiative and IESL members for their support inside and outside of academics specially Sanjeev, Kennedy, Karim, and Siva.
- Prof. Jarbas Milititsky, Prof. Luiz F. Heineck, and Prof. Carlos Torres Formoso for introducing me the research environment.
- MIT's Civil and Environmental Engineering Department for the additional financial support.
- Most importantly, my wife, my daughter, and my son for bearing with me all my sorrows and joys.

Dedication

To:

- **My parents** who by example taught me the power of principle.
- **Tânia** for loving me so much and providing me support in everything I do.
- **Livia** for being my friend, my daughter, my pride and my best work.
- **Eric** for the unbelievable effort that he made to be among us in this important phase of my professional life

Contents

List of Figures

Chapter 1

1 Introduction

1.1 Overview

The main task of designers is to apply their scientific and engineering knowledge to the solution of technical problems, and then to optimize those solutions within the requirements and constraints set by material, technological, economic, legal, environmental and human-related considerations [Pahl and Beitz, 1996].

Design can be described as an information processing activity. After each information output, it might become necessary to improve or increase the value of the result of the previous step repeating each step until the necessary improvements have been achieved. Every design involves first of all a confrontation of the problem with what is known or not known. The intensity of this confrontation depends on the knowledge, ability and experience of the designers on the particular field in which they are engaged. In all cases, however, more detailed information about the task itself, about the constraints, about possible solution principles, and about known solutions for similar problems is extremely useful as it clarifies the precise nature of the requirements. This information can also reduce confrontation and increase confidence for finding solutions.

This research presents the M-RAM (Multi-Reasoning Artificial Mind) model that aim to assist engineers in the initial stages of the structural design of tall buildings. This preliminary conceptual design involves selecting preliminary materials, selecting the overall structural form of the building, producing a rough dimensional layout, and considering technological possibilities. Decisions are made on the basis of such information as height of the building, building use, typical live load, wind velocity, earthquake loading, design fundamental period, design acceleration, maximum lateral deflection, spans, story height, and other client requirements. Based on functional, architectural, and aesthetic requirements, the conceptual design is accomplished, where the designer arrive at a few possible configurations for the structural systems, mainly based on geometry and load distribution for gravity and lateral loads, such as dead, life, wind, and earthquake loading.

The conceptual design phase is the first step in the design process. After knowing the problem definition, a designer makes an overall guess about the feasible solution, consistent with designer's experience and knowledge, the constraints, and the requirements. The efficiency and success of the design process depends heavily on the initial guess. Because this design phase requires a lot of knowledge, use of past experience, rules of thumb, intuition, and so forth it is very difficult to use any procedural programming language for its computerization [Maher, 1987]. The human intelligence plays a very important role being hard to computerize because it needs human intuition.

In the conceptual design phase past experience plays a vital role, where the process is often assisted by identifying, retrieving and then modifying appropriate past design cases, and generalizing the cases, with the implied benefit of being able to utilize relevant past experience [Manfaat et al. 1997].

During this research a model and a system prototype were developed with the objective of providing designers with adapted past design solutions. The application of a new Artificial Intelligence approach allowed to overcome the difficulties presented by traditional procedural programming languages to computerize or to support the conceptual phase of structural design for tall buildings.

1.2 Research Problem

The design problem is a complex problem that cannot be solved by only one reasoning mechanism. To solve this kind of problem human designers use a combination of heuristic cases, first principles, and past experience to reduce complexity of the problem and to come up with possible solutions.

The most powerful way known for discovering how to solve a complex problem is to find a method that splits it into several simpler ones, each of which can be solved separately. The division of the problem into its smallest unit parts has the objective of imitating the way the human brain deals with complex problems. According to Minsky [1988], each mind is made of many smaller processes. These, he calls agents. Each mental agent by itself can only do some simple thing that needs no mind or thought at all. Yet when those agents are joined in societies – in certain very special ways – this lead to true intelligence [Minsky, 1988]. The human brain, when dealing with a complex problem uses different reasoning mechanisms to solve each part of the divided problem. No reasoning mechanism by itself is powerful enough to solve a complex problem like the conceptual design. The perception of the complexity of the problem being studied led to the formulation of the following questions:

1. Is it possible to decompose the design problem to allow complexity reduction?
2. Is it feasible to develop a model that mimics the designer reasoning?
3. Can we map different Artificial Intelligence tools to each part of the decomposed problem?
4. Can a Distributed Artificial Intelligence system support this design process?

1.3 Research Objectives

The aim of this research is to develop a methodology to support designers to solve the initial structural design problem that can be classified as a complex problem. The final objective is to propose an architecture, based on collaborative solutions of a problem by several reasoning mechanisms, allowing complexity reduction or, in other words, reduction of the search space by decomposing a problem into interacting modules, yielding smaller problems that are much easier to tackle.

The M-RAM (Multi-Reasoning Artificial Mind) model goal is to provide designers with adapted past design solutions with the help of a distributed multi-reasoning mechanism and to support designers to reason in a better way employing past experience and avoiding past mistakes. The final objective is to relieve structural designers from having to "reinvent the wheel" with the need to develop feasible structural configuration for each new design situation from scratch and to allow designers to avoid limiting

The scope of testing of both the model and the prototype is limited to small-scale problems dealing with the conceptual phase of structural design for high rise buildings. The approach is potentially extensible to apply in the design of large-scale design problems.

1.4 Thesis Organization

- **Chapter 2** presents the background information for the research that supported the research described in the reminder of the thesis. Conceptual structural design theory and principles, traditional artificial intelligence, machine learning algorithms, reasoning mechanisms, distributed artificial intelligence, knowledge discovery in databases, and data mining research are introduced.

- **Chapter 3** describes the M-RAM model, which incorporates the different elements of conceptual design. This chapter presents the M-RAM primitive classes, its relationships, and the process of design with M-RAM objects.

- **Chapter 4** presents the functionality of the developed prototype that supports structural designers in the conceptual phase of the structural design of tall buildings. The different sections of this chapter cover the system architecture and implementation.

- **Chapter 5** illustrates the system at work through an example.

- **Chapter 6** summarizes the main issues discussed in the thesis. It also discusses future research issues.

1.5 Conclusions

Finding a solution for the conceptual phase of the structural design requires detailed information about the task itself, about project constraints, about possible solutions principles, and about known solutions for similar problems. It requires a lot of designer intelligence being very hard to computerize because it needs human intuition. However, this stage of the design is of fundamental importance for the quality of the design as a whole and consequently for the quality of the final product, the building being designed. Therefore the need for models to better understand this design process and for computer systems to support designers during this design phase must be addressed.

Chapter 2

2 Supporting Research

2.1 Introduction

This chapter summarizes the knowledge about artificial intelligence, machine learning algorithms, reasoning mechanisms, support algorithms, distributed artificial intelligence and knowledge discovery in databases with the objective of understanding how the research available in these areas could support the development of a model and a system prototype to assist designers during the conceptual phase of structural design that is also summarized in the beginning of the chapter.

This research finds a connection among the different studies presented in this chapter. It focuses on distributedness and parallel implementation as factors crucial to a fast and robust system. Instead of building general functional models, an approach is proposed by developing competence models that provide expertise for particular and small task-oriented competence. In this approach each module is responsible for doing all the representation, computation, reasoning, and execution, related to its particular competence.

The emphasis in the proposed architecture is on a more direct coupling of perception to action, distributedness and decentralization, dynamic interaction with the environment and intrinsic mechanisms to cope with resource limitations and incomplete knowledge. Machine-learning algorithms and reasoning mechanisms were obtained from the literature and were adapted but not developed during this research. The focus of this research was to develop a methodology for the interconnection, collaboration and distribution of existing algorithms, mechanism, and models at the technical level, together with the application of this different expertise to support designers during the conceptual phase of structural design. To do so this research joins the efforts of the Knowledge Discovery In Databases (KDD) community in solving the coordination problems between different reasoning mechanisms and the efforts of the Distributed Artificial Intelligence community in defining communication protocols between cooperative agents.

Section 2.2 offers a summary of design theory denoting where the M-RAM research fits in this large research area. Section 2.3 presents an introduction to traditional artificial intelligence. Section 2.4, 2.5, and 2.6 provides an overview of available machine learning algorithms, reasoning mechanisms, and support algorithms. Section 2.7 provides an introduction to the distributed artificial intelligence research and Section 2.8 presents an overview of the research in knowledge discovery in databases area. Finally, Section 2.9 provides a summary of this chapter.

2.2 Design Process

Engineering design is an activity in which people have been engaged for centuries. It is only in the past few decades that this activity has been perceived as a systematic process capable of comprehensive analysis and improvement. The penetration of knowledge-based tools into the design process is widely perceived as leading to fundamental changes not only in the accuracy and speed in which a design is achieved, but also in the design process itself.

The overall design process can be decomposed into various stages [Woodson 1966, Dixon 1966, Hubka 1982, French 1985, Gardam 1994, and Pahl and Beitz, 1996]. The decomposition of the design process may vary from one theory to another, but in general it contains:

1. **Needs analysis**: Needs analysis is the stage in which a need is identified and justified.

2. **Problem statement**: Problem statement is the phase in which the problem specifications are defined.

3. **Conceptual design**: Conceptual design is the phase in which various designs alternatives are generated and evaluated.

4. **Embodiment stage**: Embodiment stage refines the design alternatives an performs more sophisticated analysis and evaluation.

5. **Detailed design**: Detailed design is the phase that lays out deign specifications for production.

6. **Production**: Production is the phase where the designed product is built. This stage may include part of the manufacturing efforts when design for manufacture is considered.

7. **Consumption**: Consumption is the stage in which the marketing plan is developed.

According to Serrano [1987] design is primarily a constraint satisfaction process. The major cause for revisiting previous stages is either constraint violations or the detection of missing information. The result of the design process is the description, of the artifact.

The engineering design process is primarily a synthesis analysis repetition. The engineer synthesizes (formulates) the problem and then analyzes it. If certain constraints are not met then he/she goes back to the synthesis part, changes the formulation and then analyzes again and so on until the specific constraints are met [Pouangare, 1986]. This synthesis analysis loop is also denoted as the design-analysis cycle.

As far as the structural engineer is concerned the design process consists only of two stages. The synthesis and the analysis evaluation stages. The synthesis stage can be sub-divided into two stages: selection and elaboration. Design is then a selection-elaboration-analyses-evaluation loop (Figure 1)

Depending in the way that selection is performed the design can be classified in three different ways.

1. **Routine Design:** A known set of solutions to the problem exists. The subparts and alternatives are known in advance. The engineer has to find for each subpart the appropriate alternative that satisfies the given constraints. This is the most common form of design.

2. **Innovative Design:** The decomposition of the problem is known, but the alternatives for each of its subparts do not exist and must by synthesized. The designer uses some fundamental principles of the domain to develop alternatives.

3. **Creative Design:** A known plan for the solution of the problem does not exist. The designer uses a divergent thought process rather than a convergent line of reasoning to come to the solution. This type of design is rarely performed.

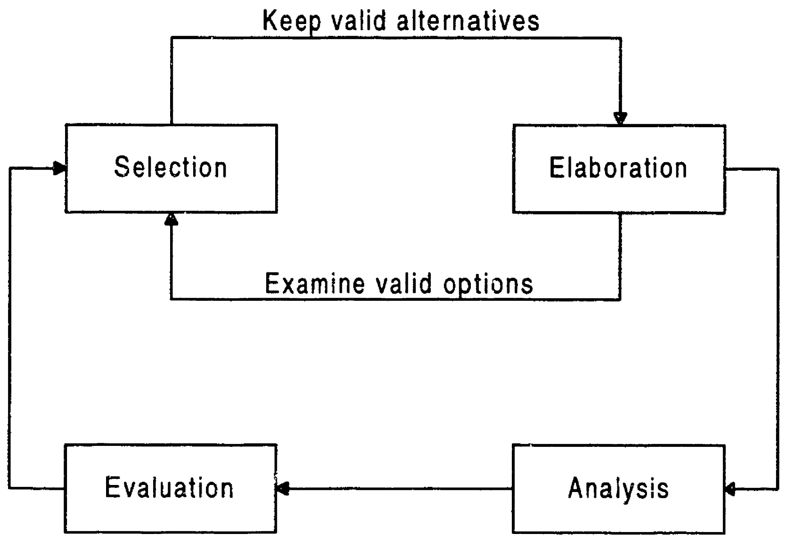


Figure 1 – Design Loop [adapted from Poungatre, 1986]

2.2.1 Structural Conceptual Design

Conceptual design can be described as the part of the design process that starts with the problem statement in the form of a set of specification. Its outcome is a set of broad (preliminary) solutions in the form of one or more concepts. A concept is a description of an artifact subject to constraints arising from several sources. The problem of conceptual design addresses both synthesis and analysis.

The designer generates possible design alternatives based on the specifications and the available knowledge. Such synthesis is followed by analysis. The analysis determines whether the proposed concept complies with the specifications. If the design fails the synthesis procedure must begin again.

Of all design phases the conceptualization phase is the most difficult one to computerize. In this phase, knowing just the basic needs, configuration of the structure and geometry of the site the structural designer has to define the structural system and the key parameters that are needed to proceed to the next design step.

The most important output of the conceptual phase of a structural design is the definition of the structural system. The structural system can be defined as an arrangement of structural components and subsystems to properly transmit the forces from the superstructure to the substructure subject to constraints on geometry, flexibility, ductility, durability and cost. The structural system must have the ability to transmit the forces both globally and locally. In the conceptualization phase the structural designer is only concerned with the global transmission of forces.

The conceptual phase of the structural design of the building involves the selection of a feasible structural system satisfying a few constraints. The key terms can be defined as selection and constraints.

- **Selection:** The selection of a structural system implies that there is a set of potential configurations from which to choose. The set of feasible configurations for a particular building must be defined with that building in mind. Classes of generic structural subsystems may be used as a basis for the generation of the set of feasible system. Some examples of structural subsystems are moment resisting frames, braced frames, shear walls, core and outriggers, tubular, and hybrid (see Section 4.4.2). These subsystem are not complete structural systems, because they do not specify the building to the extent needed for evaluation of alternatives or input to the next stage of design namely analysis. The generic structural subsystems are used as a starting point for the specification of the feasible systems and are expanded and combined to fit the needs of a particular building. The subsystem named above may be used as descriptors of the building's structural system. Typically, a high rise building is described in terms of its lateral load resisting system since this system is most critical to the function of the building. The other major functional system is the gravity load resisting system.

- **Constraints:** The constraints applicable to the conceptual phase of structural design may be grouped in several categories. The following list elaborated by Maher and Fenves [1985] defines the nature of the constraints in each group.

    1. **Spatial:** The spatial constraints defines areas set aside for circulation and mechanical equipment, and open areas needed for the functional use of the building.

2. **Administrative:** The administrative constraints include the zoning laws and heights restrictions on the building.

3. **Initial Economic:** The initial economic constraints include cost and construction time

4. **Long-term Economic:** The long term economic constraints are concerned with the operation and maintenance involved in the long-term use of the building.

5. **Horizontal Compatibility:** The horizontal compatibility constraints include considerations of compatibility between the structural system and the building components, namely, the foundations, mechanical systems, and electrical system.

6. **Vertical Compatibility:** The vertical compatibility constraints include considerations of ease of construction, contractor experience, site condition, delivery, erection method, etc.

7. **Functional:** The functional constraints are the major purely structural constraints concerned with the provision of a load path. There are three possible levels of specifying these constraints:

   - Provide a load path;
   - Provide the most direct load path;
   - Provide alternate load paths (redundancy).

8. **Stable Equilibrium:** These constraints ensure that the building or any of its components remains in stable equilibrium in its intended environment.

9. **Strength and Serviceability:** These constraints include component and system load capacity requirements as well as serviceability requirements such as stiffness. The formal representation of the constraint is contained in the applicable building and material codes, standards, and design specifications while the final design must obviously satisfy all applicable constraints, at the conceptual phase the designer must make a deliberate selection of a subset of controlling constraints to be used.

2.2.2 Related Research in the Conceptual Phase of Structural Design

In the early 1970s computers began to be applied as a helpful tool for structural design but until the early 1980s there were no tools available for the synthesis part of the design problem. This led to the situation where the engineer did the synthesis part and the computer did the analysis part of the design.

In mid 1980s with the growth of artificial intelligence science different researchers initiated distinct attempts to computerize the synthesis part of the design problem. Different expert systems were developed. One of the most innovative one called HI-RISE was developed by Maher and Fenves from Carnegie-Mellon University [Maher and Fenves, 1985]. Like the majority of the expert systems developed at that time HI-RISE fell short in the learning process (see Section 2.5.3). Knowledge acquisition and lack of flexibility were the main problems faced by expert systems like HI-RISE and those systems never evolved from the initial prototypes developed in the 1980s.

Just in mid 1990s with the development of new artificial intelligence tools that researchers reinitiated efforts to automate the synthesis part of the conceptual phase of structural design. Work done by Fenves et al. [1995] applied case based reasoning to allow designers to save solutions as a by-product of the design process. Such cases could then be retrieved in later "similar" design situations and adapted to the new situation. Maher and Garza [1996] went one step further applying genetic algorithms to adapt cases retrieved by the case based reasoning tool.

M-RAM research builds upon the work done by those researchers. The main difference resides in the approach that was applied. First a model was developed to allow a better understanding of the problem requirements. This led to a distributed agent architecture implementation providing a "plug and play" approach with easy system maintenance and evolution as will be demonstrated in the next two chapters.

2.3 Traditional Artificial Intelligence

Barr and Feigenbaum [1981] define artificial intelligence as:

> *"Artificial Intelligence is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior – understanding language, learning, reasoning, solving problems, and so on."*

The goals of creating artificial intelligence and artificial life can be traced back to the very beginnings of the computer age. The earliest computer scientists – Alan Turing, John von Neumann, Norbert Wiener, and others – were motivated in large part by visions of imbuing computer programs with intelligence, with the life-like ability to self-replicate, and with the adaptive capability to learn and control their environments. These early pioneers of computer science were as much interested in biology and psychology as in electronics, and they looked to natural systems as guiding metaphors for how to achieve their visions [Mitchell, 1997].

In 1956 the term Artificial Intelligence was coined when John McCarthy organized a two-month ten-man study of artificial intelligence at Dartmouth College, New Hampshire. In the 1960s general problem solving methods, supplemented by domain specific heuristics, were applied to a wide range of problems, and AI gradually separated out into the application areas of language understanding and generation, game playing, theorem proving, vision, and robotics. At this time there was little attempt to construct programs that accurately modeled the human mind; the emphasis was on performance – computer systems that acted in intelligent ways by, for instance, playing chess or solving mathematical problems.

Artificial Intelligence (AI) is a very general investigation of the nature of intelligence and the principles and mechanism required for understanding and replicating it. According to Sharples et al. [1989] AI is a cloth woven from three academic disciplines – psychology (cognitive modeling), philosophy (philosophy of mind), and computer science – with further strands from linguistics, mathematics, and logic. The aim of AI is broad: to get below the surface of human behavior, to discover the processes, systems, and principles that make intelligent behavior possible. Computers are needed as tools for modeling these mental states and processes.

A basic notion about computer-based problem solving can be traced back to early attempts to program computers to perform tasks such as game-playing and puzzle-solving programs.

The fundamental idea that came out of early research is called space search. Problems were formulated in terms of three ingredients:

1. **Starting state** such as the initial state of a chessboard.

2. **Termination test** for detecting final states or solutions to the problem, such as a simple rule for detecting checkmate in chess.

3. **Set of operations** that can be applied to change the current state of the problem, such as a legal move of chess.

The simplest form of state space search is called generate-and-test, and its basic algorithm is:

1. Generate a possible solution in form of a state in the search space, a new position as the result of a move.

2. Test to see if the state is actually a solution by seeing if it satisfies the conditions for success, such as a checkmate.

3. If the current state is a solution, then quit, else go back to step 1.

There are two main variants of the basic generate-and-test: depth-first search and breadth-first search. The difference between them lies in the order in which possible solutions are generated in step 1.

At any given state N, depth-first search consider the "successors" of N, those states which result from applying operations to N, before considering "siblings" of N (states which were generated along with N, when operations where applied to N's "ancestors"). In breadth-first search, it is the other way around; N's "siblings" are checked out before expanding N, that is, before going on to N's "successors". Thus, in breadth-first search, one searches layer by layer through successive levels of the search space, whereas in depth-first search one pursues a single path at a time, returning to N to pick another path only if the current path fails.

Depth-first search reach the solution faster as long as it is guided in some "intelligent" way, that is, if it makes good decisions when choosing which path to pursue next. On the other hand, breadth-first search may never terminate if the search space is infinite, even if a solution exists along some as yet unexplored path, the number of feasible sates may grow exponentially, phenomenon usually referred as combinatorial explosion.

In addition to game playing, another principal concern of early artificial intelligence researchers was what is called theorem proving. Theorem proving involves showing that some statement in which we are interested follows logically from a set of special statements, the axioms (which are known or assumed to be true), and is therefore a theorem. Thus, knowledge relevant to the solution of some problem can be represented as a set of axioms. Problem solving can be viewed as the process of showing that the desired solution is a theorem. Unfortunately, the process of generating all the theorems that follow from some set of axioms is also combinatorially explosive, since one can add any theorems derived to the axioms and use the new set of statements to derive still more theorems.

Given that exhaustive search is not feasible for anything other than small search spaces, some means of guiding the search is required. A search that uses one or more items of domain-specific knowledge to traverse a state space is called heuristic search. A heuristic is best thought of as a rule of thumb; it is not a guarantee to succeed, in the way that an algorithm or decision procedure is, but it was considered to be useful in the majority of cases.

A simple form of heuristic search is hill climbing. This involves giving the program an evaluation function which it can apply to the current state of the problem to obtain a rough estimate of how well things are going. There are well-known problems with this approach. The evaluation function may not be a faithful estimate of the "goodness" of the current state of the problem and even if the evaluation function gives a good estimate, there are various other problems like that of local maxima, which occurs when the evaluation function leads to a peak position, from which the only way is down, while the solution is on other, higher peak.

Early artificial intelligence attempted to tackle the difficulties of search by explicitly representing in detail both the knowledge that experts possess about some domain and the strategies that they use to reason about what they know.

The fundamental assumption in early Artificial Intelligence research was that experts possess the necessary knowledge regarding the problem domain, and that this expert knowledge can be explicitly written using formal representations. Most research has been carried out in such a way that researchers developed a highly intelligent system in a very restricted domain. Scholars believed that these systems could be increased in scope with larger funding and increased effort. However, experiences in expert systems, machine translation systems, and other knowledge-based systems indicate that scaling up is extremely difficult. As a consequence, poor results have been obtained. The few systems built showed deficiencies such as brittleness, inflexibility, and no real time operationalism [Maes, 1991]. Thus, three factors prevented the application of the traditional Artificial Intelligence approach in the real world: incompleteness - because it is almost impossible to obtain a detailed set of knowledge for a given problem domain; incorrectness - because there is no guarantee that expert knowledge is always correct, and inconsistency - because the set of knowledge may be inconsistent.

Subsequent research in Artificial Intelligence developed systems that assumed that data resources are inconsistent, incomplete, and inaccurate [Drumheller 1986, Robertson 1987, Dixon and de Kleer 1988, Kolodner 1988, Stanfill et al. 1989, Cook 1986, Geller 1991, Huhns and Bridgeland 1991, Gero and Qian 1992, Maher and Zhao 1992, and Ketler et al. 1993]. These were however, developed for very restricted domains. Furthermore, the studies lacked feasibility analyses of scaling up the system.

2.4 Machine Learning Algorithms

The M-RAM model as will be demonstrated in Chapter 3, divided the conceptual phase of the structural design problem in its small parts and each small part of the defined problem was mapped to the most suitable technical solution. To do so eight machine learning algorithms, reasoning mechanisms, and support algorithms were evaluated and, as will be demonstrated in chapters 3 and 4, three mechanisms were chosen as the most suitable ones.

Machine learning tools allow to process raw data, to search for patterns, to describe past trends, to predict future trends, to process information in order to extract meaning, and to generalize knowledge. The algorithms investigated were neural networks, C4.5 and genetic algorithms.

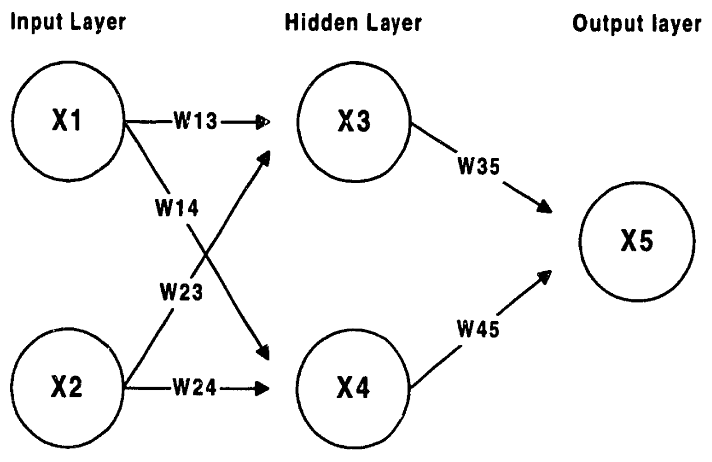
2.4.1 Neural Networks

A neural network is a computational structure inspired by the study of biological neural processing. This study has been motivated by the recognition that the brain computes in an entirely different way from the conventional digital computer [Haykin, 1994]. A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. Information processing takes place through the interaction of a large number of neurons, each of which sends excitatory and inhibitory signals to other neurons in the network. The network through a learning process acquires knowledge and interneuron connection strengths known as synaptic weights are used to store the knowledge.

A popular paradigm of learning called supervised learning involves the modification of the synaptic weights of a neural network by applying a set of labeled training samples or tasks examples. The training of the network is repeated for many examples in the set until the network reaches a steady state. The network learns from examples by constructing an input-output mapping for the problem in hand.

There are many different types of neural networks, from relatively simple to very complex, just as there are many theories on how biological neural process works. The most useful neural network model is called layered feed-forward neural network. It has layers, or subgroups of processing elements. A layer of processing elements makes independent computation on data that it receives and passes the results to another layer.

The next layer may in turn make independent computations and pass on the results to yet another layer. Finally, a subgroup of one or more processing elements makes its computation based upon a weighted sum of its inputs. The first layer is the input layer and the last the output layer. The layers that are placed between that first and last layers are the hidden layers (Figure 2).



For Example, for X3 and X5:

X3 = W23X2 + W13X1

X5 = W35X3 + W45X4

Figure 2 – A feed-forward neural network with topology 2-2-1

The processing elements are seen as units that are similar to the neurons in a human brain, and hence, they are referred as cells or artificial neurons [Rao and Rao, 1995]. In figure 2 the neurons are represented by the circular nodes. A threshold function is sometimes used to qualify the output of a neuron in the output layer. Finally, synapses between neurons are referred to as connections.

Basically, the internal activation or raw output of a neuron in a neural network is a weighted sum of its inputs, but a threshold function is also used to determine the final value, or the output. When the output is 1, the neuron is said to fire, and when it is 0, the neuron is considered not to have fired. When a threshold function is used, different results of activation, all in the same interval of values, can cause the same final output value.

Weights that are used on the connections between different layers have much significance in the working of the neural network and the characterization of a network. The following actions are possible in a neural network:

1. Start with one set of weights and run the network (no training).

2. Start with one set of weights, run the network, and modify some or all the weights, and run the network again with the new set of weights. Repeat the process until some predetermined goal is met (with training).

To be able to train the network it is important to have information fed back from the output neurons to neurons in some layer before that, to enable further processing and adjustment of weights on the connections. What is fed back is usually the error in the output, modified appropriately according to some useful paradigm. The process of feedback continues through the subsequent cycles of operation of the neural networks and ceases when training is completed.

The feed-forward network configuration with backpropagation training is the most common configuration in use due to its ease of training. It is estimated that over 80% of all neural network projects in development use backpropagation [Rao and Rao, 1995]. In backpropagation there are two phases in its learning cycle, one to propagate the input pattern through the network and the other to adapt the output, by changing the weights in the network. It is the error signals that are backpropagated in tl : network operation to the hidden layer(s). The portion of the error signal that a hidden-layer neuron receives in this process is an estimate of the contribution of a particular neuron to the output error. Adjusting on this basis the weights of the connections, the squared error, or some other metric, is reduced in each cycle and finally minimized, if possible.

Neural networks have the capability to adapt to changes in the surrounding environment, can provide information not only about which pattern to select, but also about the confidence in the decision made, exhibits a graceful degradation in performance rather than catastrophic failure, and, as a consequence of its massively parallel nature, it is potentially fast for the computation of certain tasks.

In the initial M-RAM implementation a five layers backpropagation feed-forward neural network agent was developed to work as a classification engine but because of the small size of the available data set, it had an unacceptable behavior and was not used in the final M-RAM implementation.

2.4.2 C4.5

In C4.5 numerous recorded classifications are examined and a model is constructed inductively by generalization from specific examples. This machine-learning algorithm can be defined as a set of computer programs that construct classification models by discovering and analyzing patterns found in given records. The algorithm that has fundamental importance in C4.5 is the one that generates the initial decision tree from a set of training cases. Since the cases do not all belong to the same class, a divide-and-conquer algorithm attempts to split them into subsets using statistical evaluations tests, and an information gain ratio criterion to find the best possible tree structure.

C4.5 was very useful in the current problem domain but first a definition of the most significant attributes of the design process had to be found. The biggest advantage of C4.5 is its simplicity of use and its final product: the classification tree. This tree can be described as a model of the problem being solved. The disadvantages are that, unlike neural network that can be used to predict real numerical values, C4.5 just allows us to obtain categorical results, and that it lacks incremental induction. Each time a new case is added to the collection of data a new classifier from all the accumulated data needs to be constructed.

C4.5 was one of the tools applied in the development of the M-RAM prototype. Section 4.4 presents a detailed description of its implementation and most important algorithms.

2.4.3 Genetic Algorithms

Genetic algorithms (GA) provide a robust yet efficient search methodology, explicitly modeled upon the biological "survival of the fittest" reproductive model. In the broadest sense, a GA creates a set of solutions that reproduce based on their fitness in a given environment. The process follows the following pattern: (1) An initial population of random solutions is created; (2) Each member of the population is assigned a fitness value based on its evaluation against the current problem; (3) Solutions with a higher fitness value are most likely to parent new solutions during reproduction; and (4) The new solution set replaces the old, a generation is complete, and the process continues from step 2. That sequence implements, in a most simplistic way, the concept of the survival of the fittest. The outcome of a genetic algorithm is based on probabilities just as biological success is grounded in chance. The standard model for a GA solution is a bit string called chromosome after its biological counterpart. During reproduction, the chromosomes of parent solutions combine and undergo mutation in creating the next generation. It should be noted that searching for an optimum value in a domain space can be called "learning" in the sense that one searches for this value because it is unknown, and after the search, the information represented by the value, is known. The method is not an adaptive learning mechanism like Neural Network but given a criterion function it does provide the means to learn the value of a target. Thus the method has found a place among machine learning applications such as a classifier system.

A genetic algorithm was another of the tools applied in the development of the M-RAM prototype. Section 4.6 presents a detailed description of its implementation.

2.5 Reasoning Mechanisms

These mechanisms are applied as tools for the analysis of previous similar situations and to model expert's knowledge. The mechanisms investigated are case-based reasoning, fuzzy systems, and heuristic reasoning.

2.5.1 Case-Based Reasoning

Case-based reasoning is the process of "remember and adapt" or "remember and compare [Kolodner, 1993]. A reasoner remembers previous situations similar to the current one and uses them to help solve the new problem. It is a model of reasoning that incorporates problem solving, understanding and learning and integrates all with the memory processes. Learning occurs as a natural consequence of reasoning where procedures applied to new problems are indexed in the memory. Feedback and analysis of feedback through follow-up procedures and explanatory reasoning are necessary parts of the complete reasoning-learning circle. The knowledge of a case-based reasoner is constantly changing as new experiences give rise to new cases that are stored for future use. A case-based reasoner learns from experience to exploit prior successes and avoid repeating prior failures [Leake, 1996].

Case-based reasoning provides a wide range of advantages. It allows the reasoner to propose solutions to problems quickly, avoiding the time necessary to derive the answers from scratch, allows a reasoner to propose solutions in domains that aren't completely understood, gives a reasoner a means of evaluating solutions when no algorithmic method is available for evaluation, makes it possible to interpret open-ended and ill-defined concepts, allows the reasoner to focus on important parts of a problem by pointing out what features of a problem are the crucial ones, and can warn of the potential for problems avoiding the repetition of past mistakes.

There is much evidence that people do, in fact, use case-based reasoning in their daily reasoning. People learning a new skill often refer to previous problems. Past experience can often provide guidelines on how to deal with current problems. The computer can be used as a retrieval tool to augment people's memories. References to old cases are advantageous in dealing with situations that recur. However, in applying Case-Based Reasoning to the current problem domain of conceptual structural design of tall buildings it is important to keep in mind, that since the description of problems are often incomplete, the further step of understanding or interpreting the problem is a necessary prerequisite for reasoning. Furthermore, since no previous case is ever exactly the same as a new one, it is usually necessary to adapt an old solution. Practical retrieval technologies are available, but the general adaptation problem remains extremely difficult for CBR systems.

The best use of CBR for today's applied systems is as advisory systems that rely on the user to perform evaluation and adaptation. Central question for adaptation are which aspects of a situation to adapt, which changes are reasonable for adapting them, and how to control the adaptation process.

The questions that then rise are that is it possible to interpret new data and adapt old solutions effectively in the conceptual structural design problem with the available information? How frequently do situations recur in the problem domain? This research evaluated the analysis of similar situations. Considering that few things facilitate a decision as much as a precedent, Case-Based Reasoning provided objective standards in the M-RAM model. It was another tool applied in the development of the M-RAM prototype. Section 4.5 presents a detailed description of its implementation.

2.5.2 Fuzzy Systems and Fuzzy Logic

Logic deals with true and false. A proposition can be true on one occasion and false on another. When, for example, a meteorologist says that is going to rain today he/she is making a statement with certainty. His/her statements in this case can be true or false. The truth-values of his/her statements can be only 1 or 0. This statement then is said to be crisp.

On the other hand, there are statements that cannot be made with certainty. The meteorologist may be able to say with a degree of certainty (e.g. 0.8 rather than 1) in his/her statement that it will rain today. This type of situation is what fuzzy logic was developed to model. A fuzzy system deals with propositions that can be true to a certain degree somewhere from 0 to 1. The degree of certainty sounds like a probability, but its not quite the same. Probabilities for mutually exclusive events cannot add up to more than 1, but their fuzzy values may.

According to Cox [1994] humans reason not in terms of discrete symbols and numbers but in terms of fuzzy sets. Fuzzy sets are actually functions that map a value that might be a member of the set to a number between zero (value is not in the set) and one (value completely representative of the set) indicating its actual degree of membership. Fuzzy logic is a calculus of compatibility. Unlike probability, which is based on frequency distributions in a random population, fuzzy logic deals with describing the characteristic of properties. Fuzziness is a measure of how well an instance (value) conforms to a semantic ideal or concept. It describes the degree of membership in a fuzzy set.

Fuzzy sets are actually functions that map a value that might be a member of the set to a number between 0 and 1 indicating its actual degree of membership. This produces a curve across the members of the set. Figure 3 illustrates how this concept might be represented.

The fuzzy set in figure 3 indicates to what degree a project of a specific number of stories is a member of the set shear wall system structure (see Section 4.4.2.2). As the number of stories approximates to the range of 25 to 30 floors increases our belief that shear wall system is a feasible solution as structural system to resist lateral loads. In the other hand, as the number of stories approximates the extreme values of 15 and 40 floors our belief that shear wall systems is a feasible solution to resist lateral loads decreases.

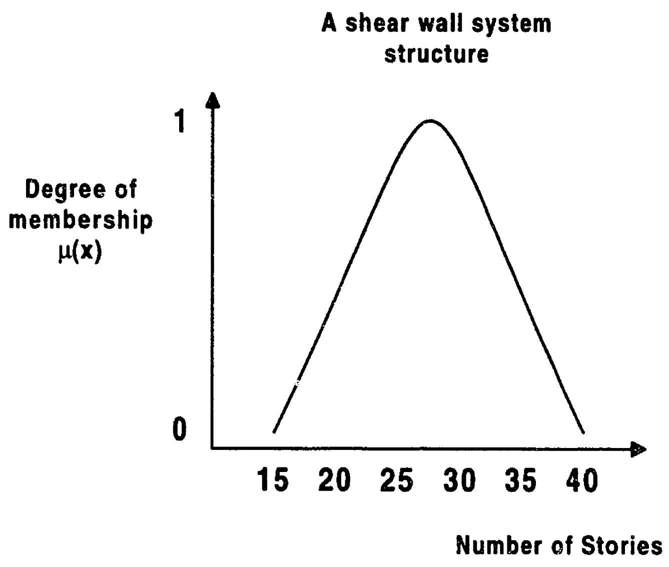


Figure 3 – Fuzzy set – Number of stories x degree of membership

Fuzzy set theory and fuzzy logic does not replace existing methods for dealing with real world problems arising in the analysis and design of decision, control and knowledge systems. Rather, they provide additional tools which enlarge the domain of problems which can be solved.

In the M-RAM implementation a fuzzyfier was applied to preprocess and to postprocess data for the neural network when the neural network agent wasn't producing the expected results as a classification engine. Because of the fact that the C4.5 agent outperformed the neural network implementation even after the application of the fuzzyfier function the fuzzy-neural network implementation was abandoned. The major difficulty faced during this implementation was knowledge acquisition for the definition of meaningful fuzzy sets. Other possible use of the fuzzy logic in the M-RAM model is to apply fuzzyfiers to improve the objective function of the genetic algorithm agent.

This was never tried but is a possible interesting improvement to the proposed model and prototype.

2.5.3 Heuristic Systems

Heuristic reasoning systems also known as the expert systems are computer programs that represent and reason with the knowledge of some specialist. Such systems may completely fulfill a function that normally requires human expertise, or may play the role of an assistant to a human decision-maker. An expert system simulates human reasoning about a problem domain rather than simulating the domain itself. It performs reasoning over representations of human knowledge. In addition to doing numerical calculations or data retrieval, it solves problems by heuristic (rule of thumb which encodes a piece of knowledge) or approximate methods [Jackson, 1990]. Expert systems do not require perfect data because solutions may be proposed with varying degrees of certainty. The expert must be able to perform the task, know how to perform the task, be able to explain how to perform the task, have the time to explain how to perform the task, and be motivated to cooperate in the enterprise. These systems fall short when inputs are not exactly as requested by the stored knowledge. Any deviation from the patterns they expect tends to result in a breakdown or impracticable behavior. They fall short too in the learning process. It is also difficult to integrate information with the already existing information. Knowledge acquisition is "the bottleneck problem" of expert systems applications.

How then can we elicit knowledge from a human expert and codify it? It must be taken into consideration that specialists have their own jargon, that facts and principles in many domains cannot be characterized precisely in terms of a mathematical theory or a deterministic world, that experts need to know more than the mere facts or principles of a domain in order to solve problems, and that the human expertise is enhanced by a lot of common sense (knowledge about the everyday world).

Expert systems were not applied in the M-RAM implementation. Difficulties to acquire and to codify knowledge from designers experts, difficulties to evolve the model integrating new with old information, lack of flexibility, and lessons learned with past expert systems implementation failures discouraged the use of those systems during the development of this research.

2.5.4 Qualitative Reasoning

Qualitative reasoning automates the process of determining all of the possible outcomes of a deterministic system where some factors necessary to analyze the system are unknown. It is thus a simulation technique, as well as an inference mechanism. An example of a deterministic system is a structural system of specific beams, columns, connections, and applied forces. In such a system, the fundamental physics necessary to analyze possible outcomes, and the differential equations representing the physics, are well known to engineers. With quantitative knowledge of the physical parameters – lengths, cross-sections, materials, forces - the system may be analyzed for highly accurate predictions of specific behavior outcomes of actual constructions. Another example of a deterministic system is a system of two reservoirs connected by a conduit. The behavior of this system may be completely analyzed and accurate behavior predictions derived, if the physical parameters of the reservoirs and conduit, the liquid characteristics, and so on, are all known.

However, early in the design process of a structural system or reservoir system knowledge of the physical parameters may be incomplete. That is, exact lengths, cross sections or loads may not be known, or may be very likely to change. Conventional analysis, which depends upon solution of systems of differential equations, is stymied, if some of the necessary variables are unknown, and must be repeated if any of the variables change in value. Qualitative analysis replaces the ordinary differential equations (ODE) of a deterministic system with qualitative differential equations (QDE), which inherently include uncertainty in their construction, development, and analytic process. The introduction of uncertainty means that the system is not deterministic, and multiple outcomes will result from a simulation analysis.

This approach to machine learning is appropriate for situations where a reasoning system must infer all of the possible outcomes of a deterministic system of phenomena which operates in accordance with known functions of time, but where quantitative knowledge of at least some of the parameters of the system is incomplete. That is, assume first that one is examining, a system of phenomena which can be described by a known system of ordinary differential equations, but for which the values of some of the variables are unknown. If the values of all of the variables were known, then solving the system of equations would fully describe the behavior of the system of phenomena. The underlying mechanisms of the phenomena operating in the system must be well understood, for such a system of equations to be possible. The unknown parameters or variable values are such matters as the size of a tank or a beam, not the fundamental way the system operates. However, even with a complete set of equations, the system with unknown variable values is no longer deterministic; multiple outcomes are possible, dependent on the values of the unknown variables. Such a system may be modeled with a qualitative reasoning framework, to automatically simulate all of the possible outcomes of the non-deterministic system, and to discover the dependencies of events in the system.

In classical qualitative reasoning [Kuipers, 1994] the mathematical functions describing the system are not modeled with the same precision that a system of differential equations models the system. Mathematical functions are grouped into broad classes, and behavior inferred for the entire class. All monotonically increasing functions are grouped together and treated as a sort of meta-function, for example. This implies that full knowledge of the underlying mechanisms may not be necessary to model a system with qualitative reasoning. Nevertheless, this machine learning paradigm may be less useful in a situation where the underlying mechanisms of a system of phenomena are unknown.

Qualitative reasoning was not applied during this research. The underlying mechanisms of the conceptual phase of structural design are, until now, not well understood making it difficult to develop the required qualitative differential equations system.

2.6 Support Algorithms

Support Algorithms could be applied to give justifications to the decisions adopted by the different reasoning mechanisms. Truth Maintenance System is outlined in section 2.6.1.

2.6.1. Truth Maintenance Systems

The fundamental idea of truth maintenance systems is that a problem solver can be decomposed into two parts: an inference engine and a Truth Maintenance System (TMS). This natural partitioning of concerns allows the inference engine to focus on drawing inferences within the task domain and the TMS to focus on beliefs, assumptions, and contexts. Every important inference made by the inference engine is communicated to the TMS as a justification.

There are several different families of truth maintenance systems. Each type partitions problem-solving concerns somewhat differently and hence supports different types of problem solver inference engine interactions. Within each family there remain a large number of design alternatives. As a result there are many unexplored combination of options. Truth maintenance systems (TMS) have five principal uses:

- Identifying responsibilities for conclusions: TMS allow a problem solver to identify responsibility for its conclusions by providing rational explanations of how its conclusions follow from the premises. Generally just providing the answer is not enough. By providing explanations the problem solver enables the user (or itself) to figure what to change when things go wrong.

- Recovering from inconsistencies: In an ideal world all the data would be valid and every constraint imposed would be perfectly satisfied. Neither we not our programs live in such a world. For example, the data we give to our program can be wrong.

- Maintaining a cache of inferences: Most artificial intelligence solvers search. Since they search, they often go over parts of the search space again and again. If a problem solver cached its inferences, then it would not need to retrieve conclusions that it had already derived earlier in the search.

- Guiding backtracking: When the search detects an inconsistency while exploring the solution TMS chronological backtracking, as the name implies, backtracks to the most recent choice the search has made and explores the next alternative.

- Default reasoning: Many artificial intelligence applications require the problem solver to make conclusions based on insufficient information. TMS assumes that the solution is the generic one unless there is some evidence to the contrary.

It is important to realize that TMS are not reasoning engines by itself. They are just additional modules to existing reasoners. A good example of TMS application is the dependence network used as an additional module by VT (an Expert System for designing elevators systems). This dependence network is a TMS developed to keep track of, which values of design variables depend on values determined by earlier decisions and to propagate updates brought about by changing data or assumptions.

M-RAM relies in the description of the outcome of retrieved past experience suggested by a case-based reasoning agent to provide the justification for the solutions that it presents to the design problem being analyzed. In a future implementation a truth maintenance system module could be an interesting addition to the proposed model to provide this feature in a more structured way.

2.7 Distributed Artificial Intelligence (DAI)

Artificial Intelligence has historically been interested in the architecture of single agents. Work on natural language understanding, planning, knowledge representation, reasoning, and learning focused on how an agent would operate carrying out various sophisticated tasks. Even when the tasks themselves where inherently multi-agent (such as understanding natural language), the approach was to study how to design the individual agent. Artificial intelligence has focused on the micro issues of agent architecture.

All this began to change with the emergence of a distinct sub field within artificial intelligence, known as distributed artificial intelligence (DAI). It traces its roots to early work in the Contract Net Protocol [Smith, 1980]. The main research focus of distributed artificial intelligence has been on ways of getting multiple agents to interact appropriately. Early work was exclusively on interacting agents that had been designed by a single designer; to a certain degree, agents could be counted on to act for the greater good of the system, since they could be programmed that way by their designer who was only concerned with increasing the general system's performance, not the performance of individual agents. Utility was measured at the system level.

Now a distinct research direction within DAI is beginning to take shape. Certain researchers began to ask questions related to individually motivated agents, who had been designed by independent designers.

The original stream of DAI research became known as Cooperative Problem Solving (CPS), while the latter stream became known as Multiple Agent Systems (MAS). The approach taken in this research is based on MAS. The question it analyzes is those that are of interest when independently designed and motivated agents interact.

With the advent of large computer and telecommunication networks, the problem of integrating and coordinating many human and automated problem solvers working on multiple simultaneous problems has becoming a pressing concern. Just as traditional AI research has sometimes used individual human psychology or cognition as a model or driving metaphor, DAI considers concepts such as group interaction, social organization, and society as metaphors and problem generators.

Research done by Lenat [1975], Durfee and Lesser [1987], Bond and Gasser [1988], Lander and Lesser [1989], Lesser and Durfee [1989], Sycara [1989], Gasser [1992], Pope et al. [1992], Shoham and Tennenholtz [1993], Zlotkin and Rosenchain [1991], and Chaib-draa [1995], in Distributed Artificial Intelligence (DAI) supported the M-RAM model in defining a broad range of issues related to the distribution and coordination of knowledge and actions in an environment that involves multiple entities. These entities, called agents, can be viewed collective as a society where agents work together to achieve their own goals, as well as the goals of the society as a whole.

Researchers view the focus of DAI studies differently. According to Gasser [1992], DAI is concerned with the study and construction of semi-autonomous automated systems that interact with each other and their environments. It goes beyond the study of individual "intelligent agents" solving individual problems, to consider problem solving that has social components.

Rosenchein and Zlotkin [1994] classify existing DAI research according to the model applied to coordinate agent interaction. According to those authors four main streams of research in DAI have approached the problem of multi-agent coordination in different ways. They categorize it in the general areas of multi-agent planning, negotiation, social laws, and economic approaches.

- **Multi-Agent Planning** - DAI research concentrated on "planning for multiple agents" aims to demonstrate that problem solvers improve collective coherence by synchronizing actions with global plans. Centralized multiagent planning attempts to do this by having one problem solver generate a plan for all of the others [Smith, 1980]. A second focus of research has been "distributed planning" where problem solvers incrementally build a plan by constructing their local subparts, cooperatively, resolving the conflicts, and then aggregating them into one plan [Durfee and Lesser 1987, Zlotkin and Rosenchein, 1991, and Pope et al. 1992].

- **Negotiation** - Interagent collaboration in DAI systems has been explored by different researchers with different approaches. One approach aims at agents exchanging partial solutions at various levels of detail to construct global solutions. This includes a study of effective strategies for communication of data and hypotheses among agents. These negotiation procedures also include the exchange of partial global plans [Lichter et al., 1994], the communication of information intended to alter others agent's goals [Sykara, 1989], and the use of incremental suggestions leading to a joint plan of action [Wilkenfeld and Kraus, 1993].

- **Social Law** - Researchers have suggested applying the "society" metaphor to artificial systems to improve the performance of its agents. Systems are structured so that agents will not arrive at potentially conflicting situations. These systems have a flat organization of problem solvers who are specialists in a given area. Their mode of interaction is directed by rules of behavior, which amount to a protocol of interaction [Lenat 1975 and Shoham and Tennenholtz 1993]. These social laws are seen as a method to avoid the necessity for costly coordination techniques, like planning and negotiation.

- **Economic Approaches** - Control is distributed in a marketplace. Agents interact via competition for tasks and resources through bidding and contractual mechanisms for control, or economic valuation of services and demand [Smith 1980 and Malone et al. 1988].

The M-RAM research and the majority of DAI research have the common goal of building a dynamically self configurable system which can adapt to new environments of circumstances without input or redesign from human user. Existing DAI systems provided good testbed for the development of the M-RAM model but is important to consider that most DAI experiments and theories depend upon closed-systems assumptions such as common communication protocols, a shared global means of assessing coherent behavior, some ultimate commensurability of knowledge, or some boundary to a system. Most DAI systems suppose that all agents are built using the same architecture. These current models cannot emphasize autonomy and heterogeneity and are inadequate for supporting the integration of heterogeneous systems with possibly incompatible internal semantics like the ones that M-RAM integrates.

2.8 Knowledge Discovery in Databases

In the last decade, we have seen an explosive growth in our capabilities to both generate and collect data. Advances in scientific data collection, introduction of bar codes, and computerization of many businesses have generated a flood of data. Advances in data storage technology and better database management systems allowed us to transform the available data in a huge quantity of stored data. These clearly overwhelm the traditional manual methods of data analysis. A significant need exists for a new generation of techniques and tools with the ability to assist humans in analyzing the available data for nuggets of useful knowledge. These techniques and tools are the subject of the emerging field of knowledge discovery in databases (KDD).

Historically the notion of finding useful patterns in raw data has been given various names, including KDD and data mining. Statisticians, data analysts and information systems managers have commonly used the term data mining, while KDD has been mostly used by artificial intelligence and machine learning researchers. In this research we adopt Fayyad et al [1996] view that KDD refers to the overall process of discovering useful knowledge from data while data mining refers to the application of algorithms for extracting patterns from data without the additional steps of the KDD process. In other words data mining is considered to be just one of the steps in the KDD process.

KDD is then defined as the overall process of finding and interpreting patterns from data [Fayyad et al., 1996]. It is typically interactive and iterative, involving the repeated application of specific data mining methods or algorithms and the interpretation of the patterns generated by the algorithms. The KDD process is a multi-step process, that involves data preparation, search for patterns, knowledge evaluation, and refinement involving iteration after modification. KDD is, thus the non-trivial process of identifying valid, novel, potential useful, and ultimately understandable patterns in data.

KDD systems typically draw upon methods, algorithms, and techniques from diverse research fields like machine learning, pattern recognition, databases, statistics, artificial intelligence, knowledge acquisition for expert systems, and data visualization.
Brachman and Anand [1996] subdivide de KDD process in 9 steps:

1. Developing an understanding of the application domain

2. Creating a target data set.

3. Data cleaning and preprocessing with noise removal and deciding on strategies for handling missing data fields.

4. Data reduction and projection

5. Choosing the data mining tool by deciding whether the goal of the KDD process is classification (learning a function that maps a data item into one of several predefined classes), regression (learning a function which maps a data item to a real-valued prediction variable), clustering (seeking to identify a finite set of categories or clusters to describe the data), summarization (finding a compact description for a subset of data), dependency modeling (finding a model which describes significant dependencies between variables), and change and deviation detection (discovering the most significant changes in the data from previously measured or normative values).

6. Choosing the data mining algorithm(s).

7. Data mining by searching for patterns of interest in a particular representational form or a set of such representations: classification rules or tree, regression, clustering, and so forth.

8. Interpreting mined patterns for further iteration.

9. Consolidating discovered knowledge.

Information available from the Knowledge Discovery in Databases (KDD) and Data Mining research proved to be very helpful during the development of M-RAM model that intended to analyze data available in a database of past structural designs with the objective of generating knowledge. However it is important to realize that the majority of the KDD systems are supported by one data mining tool without considering the possible advantages of the interaction between different tools. When different tools are applied they are not developed as intelligent agents and don't interact. The process is managed by a human user that after receiving the results of one data mining tool plans the next step and decides which tool is going to be used next. It is equally important to know that their main objective is data description that focuses on finding human interpretable patterns describing and classifying the data. This objective is different from the main objective of this research that is data prediction that involves using some variables in the database to predict unknown or future values of other variables of interest.

2.9 Summary

Research related to design theory, traditional artificial intelligence, machine learning algorithms, reasoning mechanisms, support algorithms, distributed artificial intelligence, and Knowledge Discovery in Databases was described in this chapter. The need to better understand those different studies surge of the limitations of existing tools to solve complex problems as the design problem. The objective here was first to better understand the available tools to be able to map the most suitable ones to different parts of the decomposed design problem and second to understand what studies were available to support in the definition of communication protocols among cooperative agents.

Chapter 3

3 The M-RAM Model

3.1 Introduction

The M-RAM model is presented in this chapter and used for the modeling of the information relevant to the conceptual phase of structural design for tall buildings. The model is implemented with object-oriented modeling and design theory applying the unified modeling language (UML). In order to understand the M-RAM model and to understand the different tools used for its implementation within this research the object-oriented methodology is explained in Section 3.2 and the UML language is explained in Section 3.3. Finally, Section 3.4 presents the M-RAM model demonstrating the way in which the multi-reasoning distributed artificial intelligent system selects the characteristics of an artifact like the structural material, the structural form, the structural system, and the structural parameters for tall building design. It does so by, first, choosing the most suitable structural system using a classification agent (C4.5), then, by providing different possible structural forms and coefficients presenting cases provided by the case base reasoning (CBR) agent and, finally, by adapting the retrieved cases applying a genetic algorithm (GA) agent.

3.2 Object-Oriented Methodology

In Chapter 1 conceptual design was described as a complex problem. According to Booch [1994], the technique of mastering complexity has been known since ancient times: *"divide et impera"* (divide and rule). When designing a complex system, it is essential to decompose it into smaller and smaller parts, each of which we may then refine independently. Object oriented decomposition, where each object in the system embodies its own unique behavior and each one models some object in the real world, help us to cope with this complexity.

In the object oriented paradigm models are organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity.

A model is an abstraction of something for the purpose of understanding it before building it. Models serve several purposes as testing a physical entity before building it, communication with customers, visualization, and reduction of complexity. An object model captures the static structure of a system by showing the objects in the system, relationships between the objects, and the attributes and operations that characterize each class of objects. Object models provide an intuitive graphic representation of a system.

There is some disputes about exactly what components, characteristics, and properties are required by an object-oriented approach, but it generally includes:

1. **Object:** An object can be defined as a concept, abstraction, or thing with crisp boundaries and meaning for the problem in hand. An object has state, behavior, and

identity. The state of an object encompasses all of the properties of the object plus the current values of each of these properties, behavior is how an object acts and reacts, in terms of its state changes and message passing, and identity is the property of an object which distinguishes it from all other objects.

2. **Class:** An object class describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics [Rumbaugh et. al, 1991]. Each object is said to be an instance of its class. Each instance of the class has its own value for each attribute, but shares the attribute names and operations with other instances of the class.

3. **Attribute:** Attribute is a data value held by the object in a class. Each attribute has a value for each object instance and each attribute name is unique within a class.

4. **Operations and methods:** Operation is a function or transformation that may be applied to or by objects in a class. All objects in a class share the same operations. A method is the implementation of an operation for a class.

5. **Abstraction:** Abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. An abstraction focuses on the outside view of an object, and so serves to separate an object essential behavior from its implementation.

6. **Encapsulation:** Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior [Booch, 1994]. It serves to separate the contractual interface of an abstraction and its implementation. The abstraction of an object should precede the decisions about its implementation. Once

an implementation is selected, it should be treated as a secret of the abstraction and hidden from most clients. No part of a complex system should depend on the internal details of any other part. Encapsulation allows programs changes to be reliable made with limited effort.

7. **Modularity:** Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules. System details that are likely to change independently should be the secrets of separate modules. The only assumptions that should appear between modules are those that are considered unlikely to change.

8. **Hierarchy:** Hierarchy is a ranking or ordering of abstractions. The two most important hierarchies in a complex system are inheritance and aggregation. Inheritance defines a relationship among classes wherein one class shares the structure or behavior defined in one or more classes (denoting single inheritance and multiple inheritance, respectively). Inheritance thus represents a hierarchy of abstractions, in which a sub class inherits from one or more super classes. Typically, a subclass augments or redefines the existing structure and behavior of its super classes. Semantically, inheritance denotes an "is a" relationship implying a generalization/specialization hierarchy, wherein a sub class specializes the more general structure or behavior of its super classes. Aggregation, in the other hand, permits the physical grouping of logically related structures denoting a "part of" relationship.

9. **Polymorphism:** Polymorphism means that the same operation may behave differently in different classes. Polymorphism allows programs to automatically select the correct method to implement an operation based on the name of the operation and the class of the object being operated on.

3.3 Unified Modeling Language (UML)

The Unified Modeling Language (UML) is the successor to the object-oriented analysis and design methods that appeared in the late 80s and early 90s. Before UML the object oriented methods scene was pretty split and competitive. Different authors had methods that were very similar, yet they contained a number of often annoying minor differences among them. Talk of standardization had surfaced, but nobody seemed willing to do anything about it [Fowler, 1997].

This situation began to change in 1994 when Jim Rumbaugh joined Grady Booch at Rational Software, with the intention of merging their methods. In 1995 Ivar Jacobson joined Rational fact that allowed the unification of the three most used object-oriented methods and during 1996, Grady, Jim, and Ivar, worked on their unified method. At this time the consortium called Object Management Group (OMG) – that includes over 700 companies, representing the entire spectrum of the computer industry – created a task force to work on standardization in the methods area. Finally in 1997 Rational released version 1.0 of UML documentation as their proposal to the OMG task force. After proposing some small adaptation, OMG supported the method that soon obtained wide industry support.

The UML is a modeling language, not a method. The UML has no notion of the process. The modeling language is the (mainly graphical) notation that methods use to express designs. The process is the advice on what steps to take in doing a design. As this thesis is being written, Jim, Grady, and Ivar are working on merging their processes and the result will be called the Rational Objectory Process.

3.4 M-RAM Model

The M-RAM model is an object model that describes the structure of the objects in the system, their identity, their relationships to other objects, their attributes, and their operations. The various components of the M-RAM model shown in figure 4 are discussed here; the graphical notation is the UML method [Rational Software Corporation, 1997a] [Rational Software Corporation, 1997b] [Rational Software Corporation, 1997c] [Rational Software Corporation, 1997d].

3.4.1 Designer

A designer represents the entity (human, e.g., a structural engineer; or a computer, e.g., a structural engineer assistant) that presents a proposal, based on a design intent and/or an artifact characteristic that need to be satisfied. An example would be a designer that after receiving from the architect information and specifications of the building to be built defines structural coefficients like loads (life, wind, and earthquake), design fundamental period, and maximum lateral deflection based on past experience, local standards, and some structural formulae. All this information is used as input for the next structural design stage: the dimensioning of the structural components.

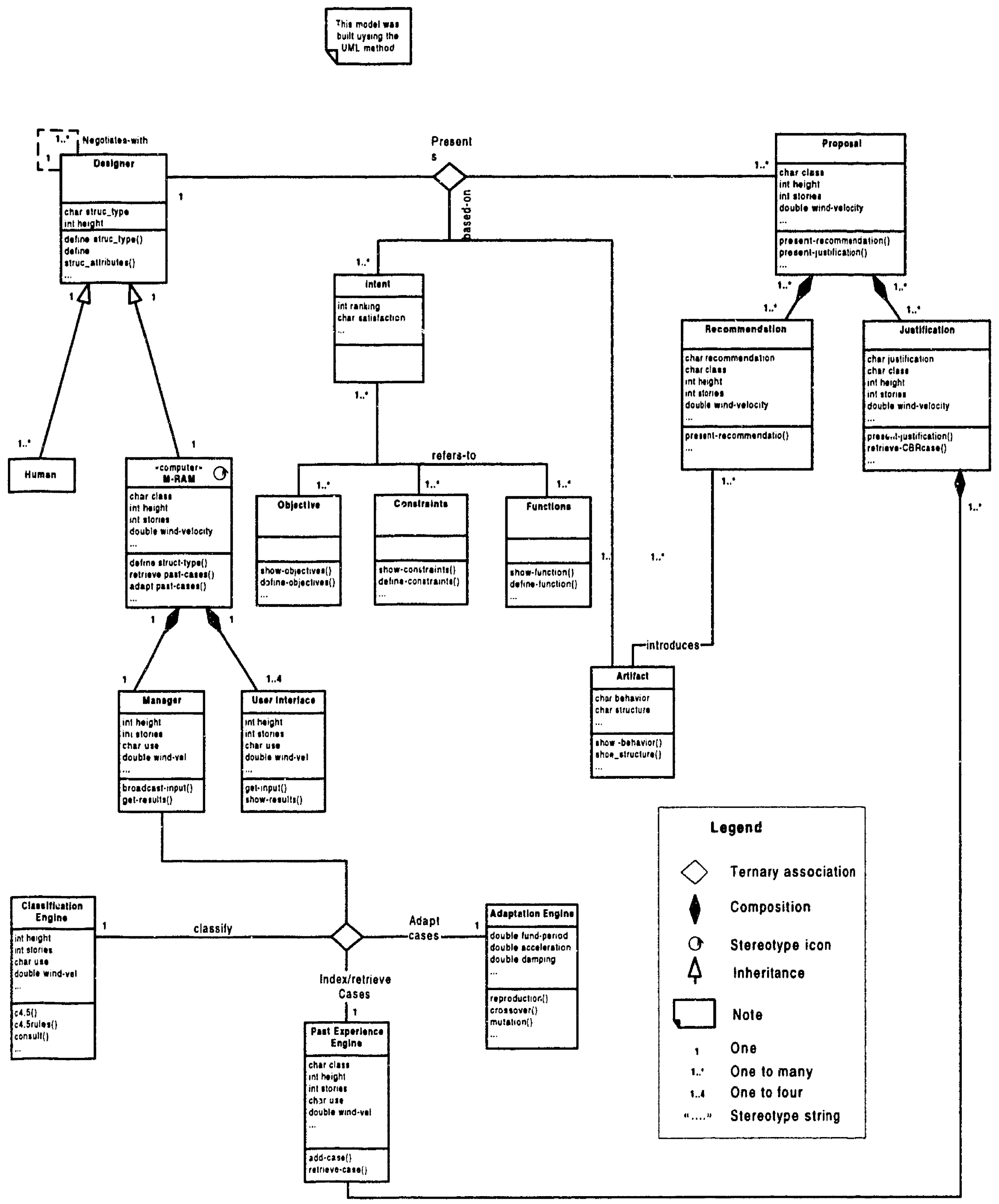


Figure 4 - M-RAM object model

[Graphical notation of the UML model (Rational Software Corp. 1997)]

3.4.2 Proposal

A proposal represents the statement given by the computer designer. A proposal includes two important elements: (1) The recommendation believed to satisfy a design intent and a required artifact characteristics, and (2) the justifications of why the recommendation fulfills that design intent and/or artifact characteristics.

3.4.3 Intent

Design intent refers to what the designer needs to achieve or satisfy [Peña-Mora et al., 1995]. Design intent has ranking that specifies its importance and a satisfaction measure that specifies the extent to which the design intent has been satisfied by a recommendation. Design intent refers to the following attributes of a design: (1) Objectives, (2) constraints, and (3) functions.

1. **"Objective"** is a characteristic to be optimized by the artifact. It presents a measure against which the design is checked (e.g., minimize the construction cost). Designers tend to use this class of intents as an evaluation, which requires comparison among several competing designs.

2. **"Constraints"** is a confinement of an artifact (e.g., lack of skilled labor in certain regions making it very difficult to build concrete structures). In this class of intents, designers only need to test if the criterion is met by the design recommendation without the need to compare it to other design alternatives (see Section 2.2.1).

3. **"Function"** is an action or activity performed by an artifact (e.g., to safely withstand the earthquake loads during the building lifetime). This class establishes the performance criteria that later translates to the behavior of the system and specific constraints.

3.4.4 Recommendation

Recommendation refers to the entity that satisfies the design intents and/or the required artifact characteristics. A recommendation is a proposed structural system presented by the C4.5 agent, the cases presented by the CBR agent, and the adaptation of the retrieved cases presented by the GA agent. Several alternate structural systems could be proposed for the same tall building.

3.4.5 Justification

Justification refers to the reason that partially explains why a recommendation will satisfy a design intent. According do Peña-Mora [1994] a justification could be a *rule* (e.g., if the number of Floors < 40 then braced frame structural system is feasible), a *case* (e.g., this building is similar to the Sears Tower in Chicago), a *catalog* (e.g., this structural system solution was taken from an entry in the "Structural Systems for Tall Buildings" published by the Council on Tall Buildings and Urban Habitat), a *principle* (e.g., the relationship between force and deformation is supported by Hoke's law), an *authority* (e.g., this structural system has been suggested by someone who is an authority in the field), a *trade-off* (e.g., this is the best possible design based on the trade off between minimizing the cost and minimizing the deflection), a *prototype* (e.g., the model of the proposed structure produced these measurements in the wind tunnel), a *constraint network* (e.g., this condition satisfies all the constraints on the system), or a *Paretto Optimal Surface* (e.g., this design fall on the surface of best possible design when optimizing cost, schedule, and ecological impact).

A justification may support other justifications by providing supporting evidence or assumptions. Cases of past experience provided by the CBR engine are used as justification in the M-RAM implementation.

3.4.6 Computer - M-RAM

The M-RAM computer implementation is a distributed artificial intelligence computer system developed as a decision support tool for the preliminary design of tall building structures. Decisions are made on the basis of such information as height of the building, building use, region, typical live load, wind velocity, earthquake loading, design fundamental period, design acceleration, maximum lateral deflection, spans, story height, and other client requirements. It runs on the Internet using a World Wide Web interface (see URL: http://cee-ta.mit.edu).

3.4.7 Manager and User Interface

The Internet is being used as a communication backbone among the different systems that implement the reasoning mechanisms being employed. Each different reasoning mechanism considered as an autonomous module can be installed in different severs dispersed in diverse locations around the world (Figure 5). Each distributed server acts as an intelligent agent. A World Wide Web (WWW) interface receives the expected data as input and launches a manager program that broadcast the problem to all existing modules. Each module solves a part of the problem and sends back their partial solution to the manager program. This program sends the solution or possible solutions back to the WWW interface that display it to the user (Figure 6).

In the current implementation M-RAM is being developed with three modules: The classification module implemented by the C4.5 agent, the past cases/experience module implemented by the CBR agent, and the adaptation module implemented by the GA agent.

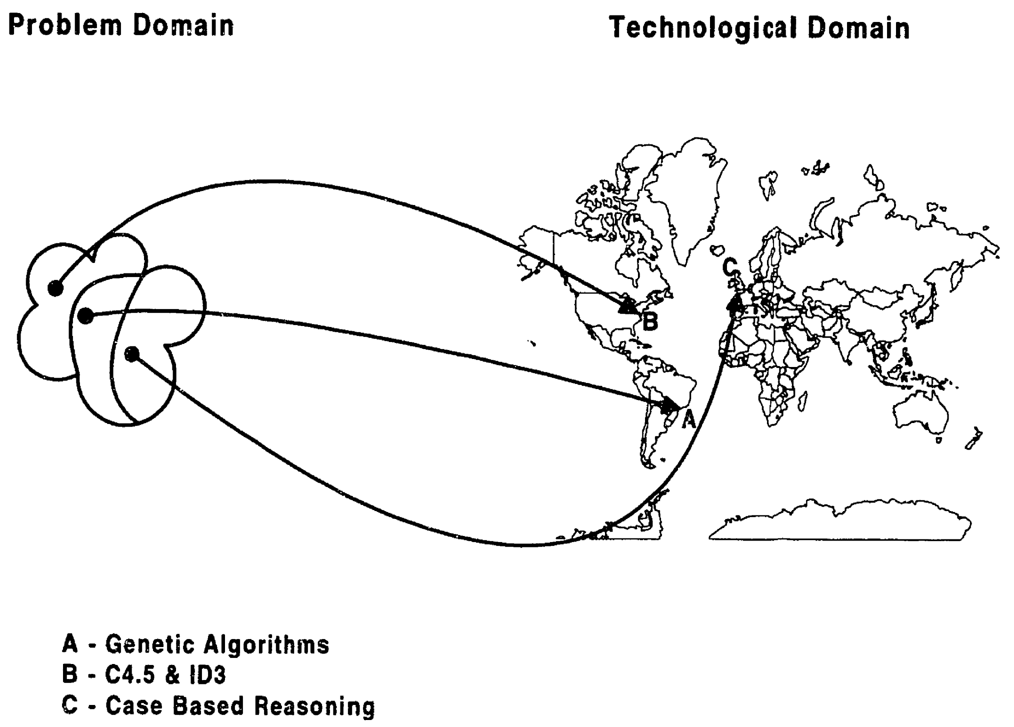
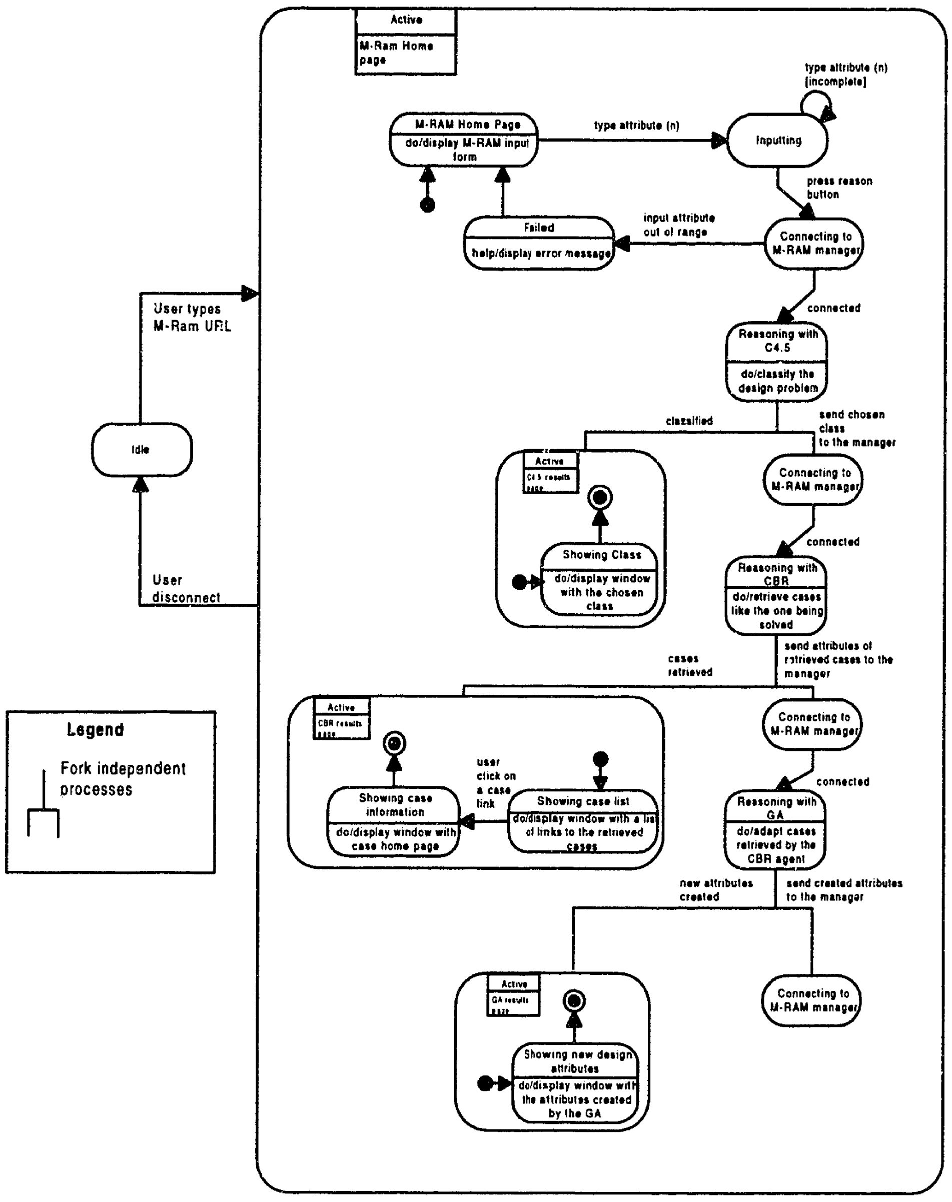


Figure 5 - M-RAM components



Active
M-Ram Home page
type attribute (n) [incomplete]
M-RAM Home Page
do/display M-RAM input form
type attribute (n)
Inputting
press reason button
Failed
help/display error message
input attribute out of range
Connecting to M-RAM manager
User types M-Ram URL
connected
Reasoning with C4.5
do/classify the design problem
classified
send chosen class to the manager
Idle
Active
C4.5 results page
Connecting to M-RAM manager
Showing Class
do/display window with the chosen class
connected
Reasoning with CBR
do/retrieve cases like the one being solved
User disconnect
cases retrieved
send attributes of retrieved cases to the manager
Active
CBR results page
Connecting to M-RAM manager
Legend
Fork independent processes
user click on a case link
Showing case information
do/display window with case home page
Showing case list
do/display window with a list of links to the retrieved cases
connected
Reasoning with GA
do/adapt cases retrieved by the CBR agent
new attributes created
send created attributes to the manager
Active
GA results page
Connecting to M-RAM manager
Showing new design attributes
do/display window with the attributes created by the GA

Figure 6 - M-RAM state diagram

[Graphical notation of the UML model (Rational Software Corp. 1997)]

3.4.8 Classification Engine

In the M-RAM implementation a C4.5 agent implemented the classification engine. In C4.5, recorded classifications are examined and a model is constructed inductively by generalization from specific examples. This machine-learning algorithm can be defined as a set of computer programs that construct classification models by discovering and analyzing patterns found in given records. The algorithm that has fundamental importance in C4.5 is the one that generates the initial decision tree from a set of training cases. Since the cases do not all belong to the same class, a divide-and-conquer algorithm attempts to split them into subsets using statistical evaluation tests, and an information gain ratio criterion to find the best possible tree structure [Quinlan, 1993].

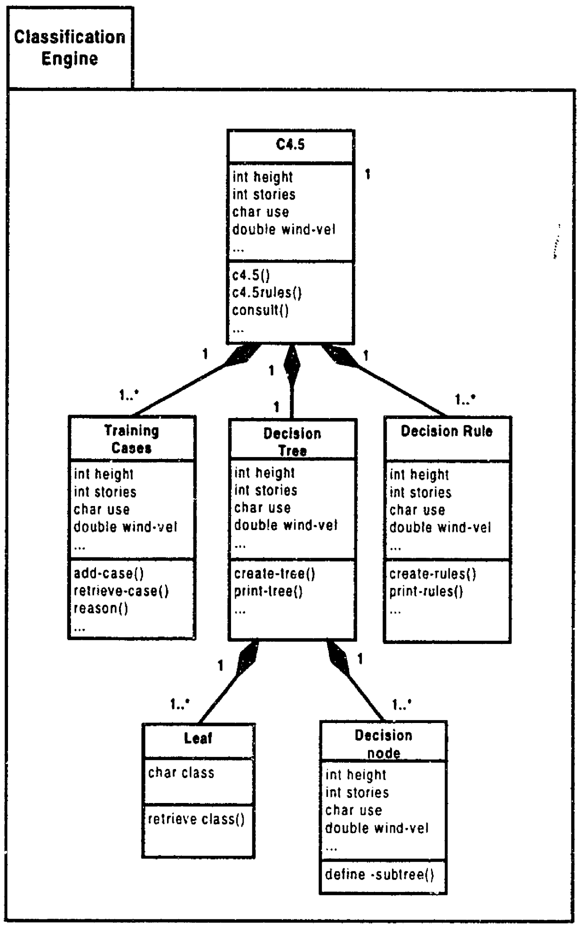


Figure 7 – classification engine

[Graphical notation of the UML model (Rational Software Corp. 1997)]

C4.5 can be divided in a set of training cases, in a decision tree, and in a set of decision rules generated by simplification of the decision tree (Figure 7). The decision tree can be subdivided in leafs that indicate classes and in decision nodes that specifies some test to be carried out on a single attribute value (see Figure 23).

3.4.9 Past Experience Engine

In the M-RAM implementation Caspian, a Case-Based Reasoning agent, implemented the past experience engine. Case-based reasoning is the process of "remember and compare" [Kolodner, 1993]. A reasoner remembers previous situations similar to the current one and uses them to help solve a new problem. It is a model of reasoning that incorporates problem solving, understanding and learning as well as integrates all with the memory processes. Learning occurs as a natural consequence of reasoning where procedures applied to the new problems are indexed in the memory. Feedback, and analysis of feedback through follow-up procedures and explanatory reasoning, are necessary parts of the complete reasoning-learning circle. The knowledge of a case-based reasoner is constantly changing as new experience give rise to new cases that are stored for future use. Case-based reasoner learns from experience to exploit prior successes and avoids repeating past failures [Leake, 1996].

The CBR agent can be divided in three parts (see figure 8): a case library, and indexing engine, and a retrieving engine. The case library is organized in the cases problem description (the state of the world at the moment the case was happening and, if appropriate, what problem needed solving at the time), the cases solution description (the state or derived solution to the problem specified in the problem description, or the reaction to the situation), and the cases outcome (the resulting state of the world after solution was carried out).

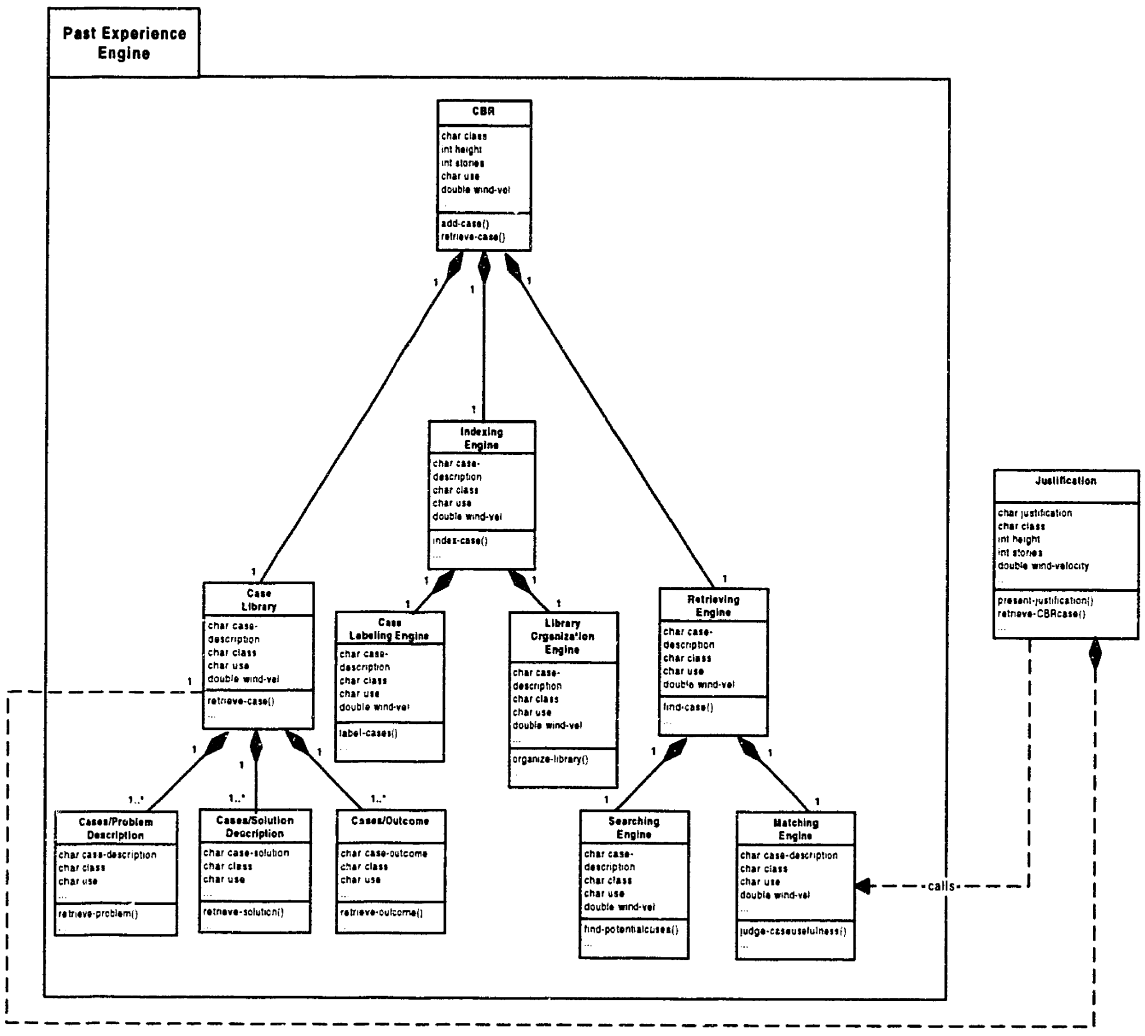


Figure 8 – Past experience engine

[Graphical notation of the UML model (Rational Software Corp. 1997)]

The biggest issue in CBR is retrieval of appropriate cases. How can the computer remember the right cases at the right time? This is known as the indexing problem. The indexing engine can be subdivided in two parts: the case labeling engine and the library organization engine. The case-labeling engine ensures that cases can be retrieved at appropriate times. In general, labels designate under what circumstances the case might have a lesson to teach and therefore when it is likely to be useful. The library organization engine organizes the cases so that the search trough the case library can be done efficiently and accurately. Related, of course, is the problem of retrieval algorithm itself.

The case library can be considered a special kind of database. Like those of a database, whose retrieval algorithms must be able to find appropriate records when queried, and like a database, it must be able to do its job fast and efficiently. Like databases, retrieval of cases can be seen as a massive search problem but with a important difference: no case in the library can ever be expected to match a new situation exactly, so search must result in retrieval of a close partial match. The retrieving engine can be subdivided in the searching engine (finding potentially matching cases) and a matching engine (each case is judged for its potential usefulness). In some schemes, search and matching are sequential; in others, they are interleaved.

3.4.10 Adaptation Engine

In the M-RAM implementation a Genetic Algorithm (GA) Agent implemented adaptation engine. In nature, variety is manifested as a variation in the chromosomes of the entities in the population [Koza, 1996]. Entities that are better able to perform tasks in their environment survive and reproduce at a higher rate. Over time, the structure of individuals in the population changes because of natural selection. When modifications in structure that arose from differences in fitness (how suitable the entity is to the surrounding environment) are visible and measurable is said that the population has evolved. In this process, structure arises from fitness.

As shown in figure 9 GA can be subdivided in phenotype (a design solution) and a genotype (a way of representing or encoding the information, which is used to produce de phenotype). The principle of the survival of the fittest determines whether a genotype survives to reproduce. The first step in preparing to solve a GA problem is the identification of a suitable representation scheme. After the phenotype is converted into a genotype the GA solves the problem with the help of three basic operations:

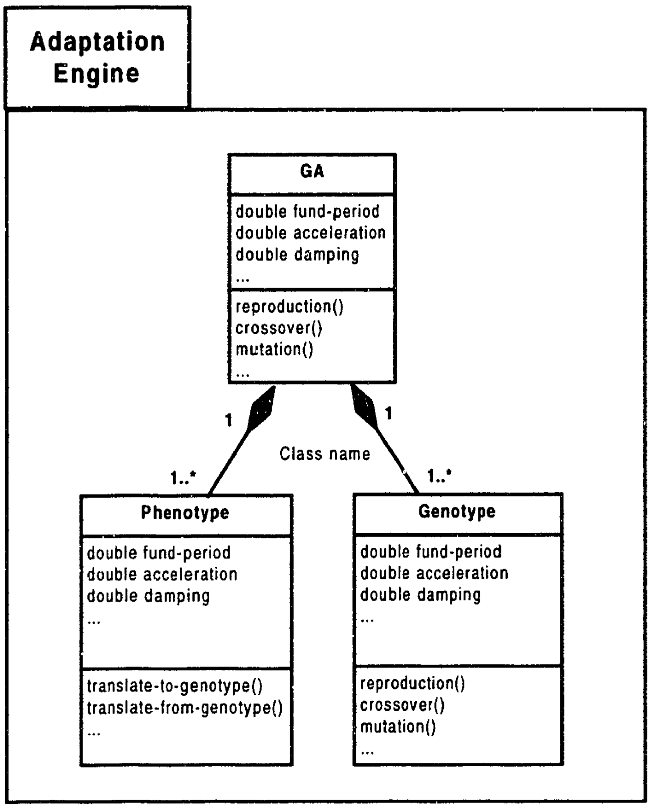


Figure 9 - Adaptation engine

[Graphical notation of the UML model (Rational Software Corp. 1997)]

reproduction, crossover, and mutation.

Reproduction performs the operation of fitness-proportionate reproduction by copying individuals in the current population into the next generation with a probability proportional to their fitness. The effect of the operation of fitness-proportionate reproduction is to improve the average fitness of the population. Improvements in the population come at the expense of the genetic diversity of the population.

Crossover allows new individuals to be created. Whereas the operation of reproduction acted on only one individual at a time, the operation of crossover starts with two parents. As with the reproduction operation, the individuals participating in the crossover operation are selected proportionate to fitness. The crossover operator uses two parents and produces two offsprings. Crossover creates something new and new individual can have higher fitness value than its parents.

The mutation operation is an asexual operation in that it operates on only one individual. A single character at the selected mutation point is changed. The mutation operation has the effect of increasing the genetic diversity of the population by creating a new individual and is potential useful in restoring lost diversity in a population.

The GA then interactively performs those operations on each generation of individuals to produce new generation of individuals until some termination criterion is satisfied.

3.5 Summary

This chapter presented the M-RAM model that allows us to better understand the conceptual phase of the structural design. The model captures all the steps which designers go through during this design process. It demonstrates how a modular computer system based in this model could be built applying autonomous agents to perform the different tasks required by the decomposed problem.

Another topic discussed in this chapter is the use of object-oriented methodology to represent items in the real world. This methodology allows the modeling of complex systems by modeling entities that take different roles during their lifetime, as usually happens with real world entities. In addition, the chapter presents UML, the Unified Modeling Language, an industry wide effort towards standardization of different object-oriented methodologies.

Chapter 4

4 M-RAM Architecture and Implementation

4.1 Introduction

Getting information processes, especially AI processes, to share a common syntax is a major problem. There is no universally accepted language in which to represent information queries. According to Destrouzos [1997], hardly anyone is paying attention to shared conventions that will allow interconnected machines to understand and work with one another without the constant intervention of a human being.

Figure 10 demonstrates how this research deals with this problem presenting the architecture that was applied in the M-RAM prototype development. One of the most important characteristics of the proposed architecture is the possibility of geographical dispersion of the reasoning mechanisms. The idea was to allow the individual agents to be developed and maintained in different parts of the world using a "plug-and-play" architecture leaving the agents where they were developed.

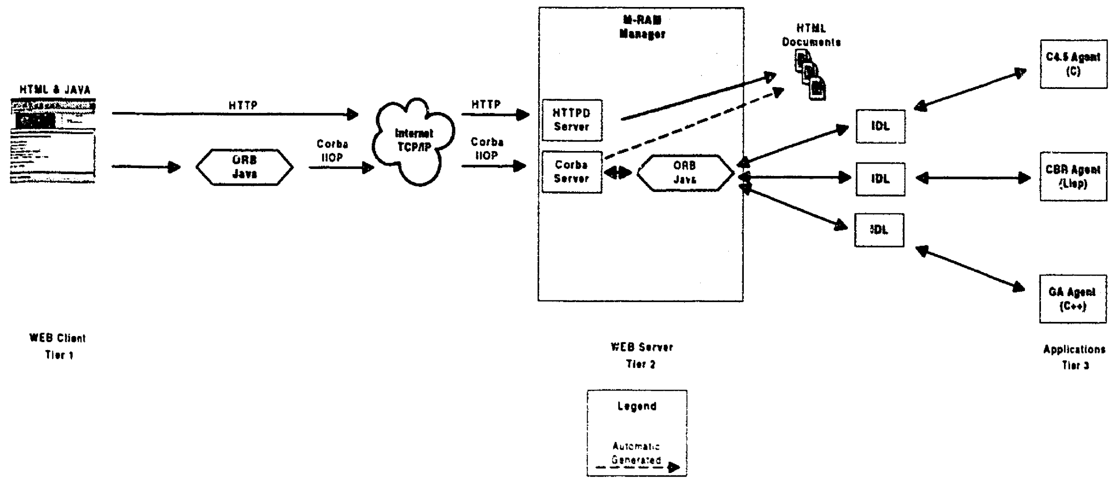
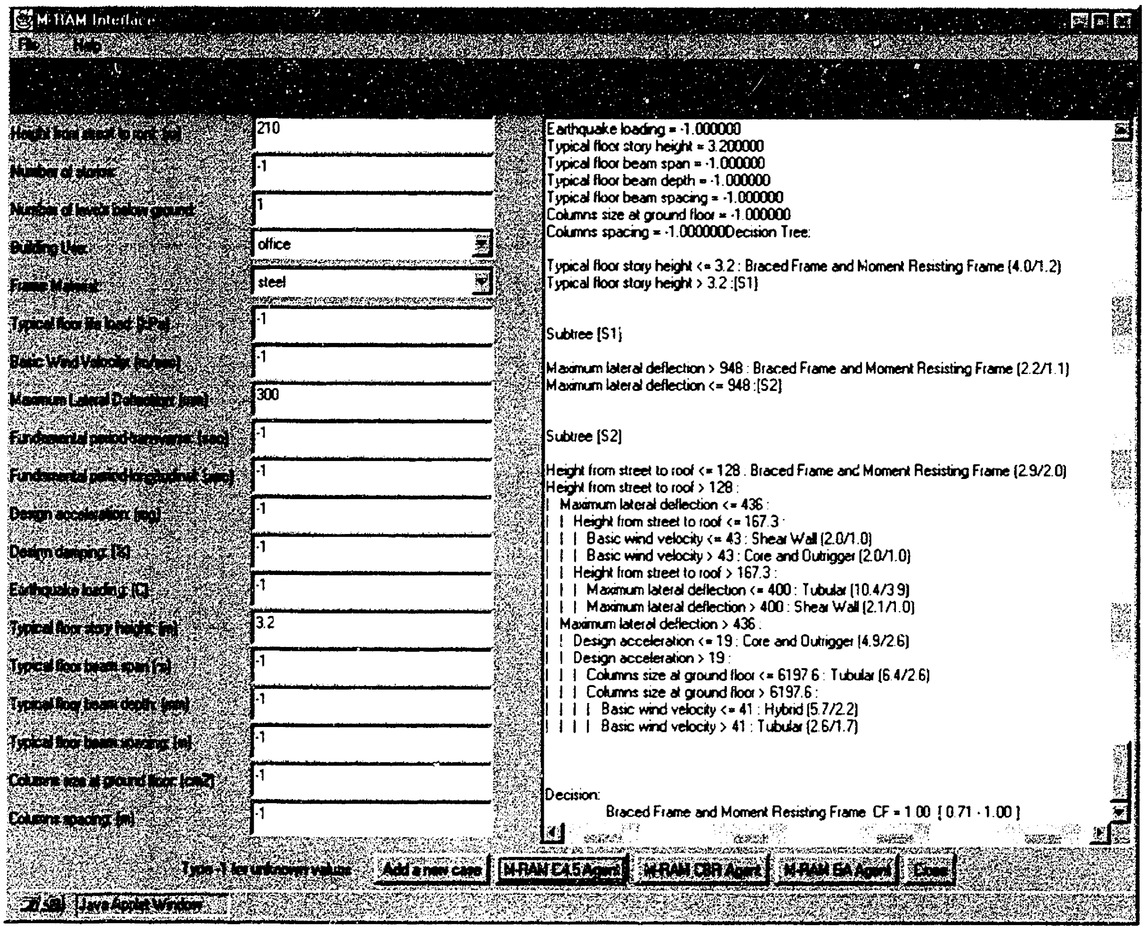


Figure 10 – M-RAM Architecture

4.2 User Interface

In the actual M-RAM implementation a WWW interface written in Java (first tier) receives the expected data as input (Figure 11), launches the manager program (second tier) that decides which server will perform which operation on the data sending the required data to the location where the application is situated (third tier). Each application solves a part of the problem sending back this partial solution to the manager program. The manager coordinates those partial solutions assuring a consistent solution for the problem as a whole. Finally the manager program send the solution or possible solutions of the structural design problem back to a WWW interface that displays it to the user.

In this implementation Java provides the mobile code foundation and the Common Object Broker Architecture (CORBA) provides the distributed object infrastructure.



Height from street to roof: [m]	210
Number of stories:	-1
Number of levels below ground:	1
Building Use:	office
Frame Material:	steel
Typical floor live load: [kPa]	-1
Basic Wind Velocity: [m/sec]	-1
Maximum Lateral Deflection: [mm]	300
Fundamental period-transverse: [sec]	-1
Fundamental period-longitudinal: [sec]	-1
Design acceleration: [%g]	-1
Design damping: [%]	-1
Earthquake loading: [C]	-1
Typical floor story height: [m]	3.2
Typical floor beam span: [m]	-1
Typical floor beam depth: [mm]	-1
Typical floor beam spacing: [m]	-1
Columns size at ground floor: [cm2]	-1
Columns spacing: [m]	-1

```
Earthquake loading = -1.000000
Typical floor story height = 3.200000
Typical floor beam span = -1.000000
Typical floor beam depth = -1.000000
Typical floor beam spacing = -1.000000
Columns size at ground floor = -1.000000
Columns spacing = -1.000000Decision Tree:

Typical floor story height <= 3.2 : Braced Frame and Moment Resisting Frame (4.0/1.2)
Typical floor story height > 3.2 :[S1]


Subtree [S1]

Maximum lateral deflection > 948 : Braced Frame and Moment Resisting Frame (2.2/1.1)
Maximum lateral deflection <= 948 :[S2]


Subtree [S2]

Height from street to roof <= 128 : Braced Frame and Moment Resisting Frame (2.9/2.0)
Height from street to roof > 128 :
|   Maximum lateral deflection <= 436 :
|   |   Height from street to roof <= 167.3 :
|   |   |   Basic wind velocity <= 43 : Shear Wall (2.0/1.0)
|   |   |   Basic wind velocity > 43 : Core and Outrigger (2.0/1.0)
|   |   Height from street to roof > 167.3 :
|   |   |   Maximum lateral deflection <= 400 : Tubular (10.4/3.9)
|   |   |   Maximum lateral deflection > 400 : Shear Wall (2.1/1.0)
|   Maximum lateral deflection > 436 :
|   |   Design acceleration <= 19 : Core and Outrigger (4.9/2.6)
|   |   Design acceleration > 19 :
|   |   |   Columns size at ground floor <= 6197.6 : Tubular (6.4/2.6)
|   |   |   Columns size at ground floor > 6197.6 :
|   |   |   |   Basic wind velocity <= 41 : Hybrid (5.7/2.2)
|   |   |   |   Basic wind velocity > 41 : Tubular (2.6/1.7)



Decision:
        Braced Frame and Moment Resisting Frame  CF = 1.00  [ 0.71 - 1.00 ]
```

Figure 11 - M-RAM Interface

4.3 The Corba Bus

According to Mawbray and Ruh [1997] integration is often viewed as simpler process than new development. Unfortunately integration has not resulted in the deployment of new capability at either a faster or a cheaper rate. This is primarily due to the need to customize interfaces for components that were not originally designed to work together. In addition, a networked environment requires communications across heterogeneous hardware platforms. The time and energy spent to develop the interfaces can easily exceed the effort required to develop the code for the functions themselves.

While system software manufacturers have long recognized the need for tools that support system integration, many have fallen short of satisfying the need of the system implementators. The tools that do exist consist primarily of scripting languages, code libraries, and clipboard functions. Most of these provide only low-level communications-oriented functions, such as sockets and remote procedure calls.

In recognition of this technology vacuum, the Object Management Group (OMG) defined the CORBA standard. The CORBA specification has been implemented by numerous hardware and system software manufacturers, creating a rich and robust framework that successfully operates across heterogeneous computing platforms.

CORBA objects can be defined as a binary component that remote clients can access via method invocations. Clients don't need to know where the distributed object resides, what operation a system executes on, or how the server object is implemented. What the client needs to know is the interface the server object publishes. This interface serves as a binding contract between client and servers [Orfali and Harkey, 1997]. CORBA distributed object technology allows us to put together complex client/server information systems by simply assembling and extending components.

CORBA is based on a client-server model of distributed computing. In the client-server model, a request for service is made from one software component to another on a network. CORBA adds an additional dimension to this model by inserting a broker (ORB – Object Request Broker) between the client and the server components.

The broker plays two hey roles. First, it provides common services, including basic messaging and communication between client and server, directory services, meta-data description, security services, and location transparency. Second, it insulates the application from the specifics of the system configuration, such as hardware platforms and operating systems, network protocols, and implementation languages.

With a broker each interface is defined just once and the broker handles subsequent interactions. With CORBA IDL (Interface Definition Language) these interfaces can be defined in a standardized, platform-independent fashion. The advantage to the software developer is a significant reduction in complexity. Without the broker, the number of interfaces is an N-squared problem; with the broker, only N interfaces are required (see Figure 12).

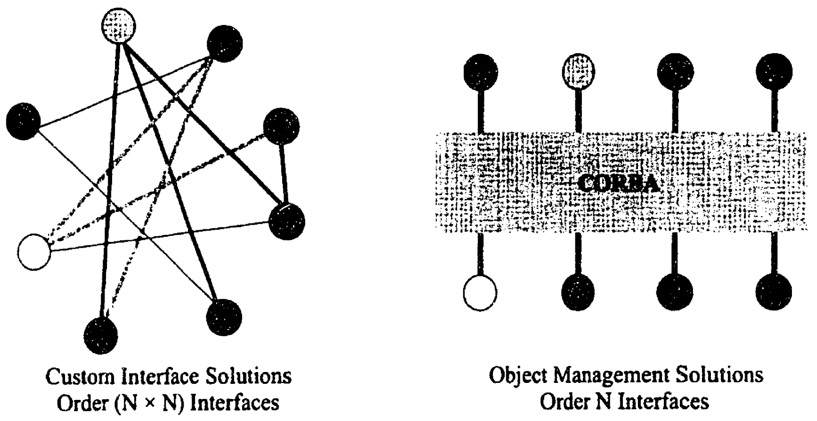


Figure 12 – Custom interfaces versus CORBA solutions

The most important benefits provided by CORBA ORB are:

1. **Static and dynamic method invocations**: A CORBA ORB lets the programmer either statically define his/her method invocations at compile time, or it lets him/her dynamically discover them at run time.

2. **High-level language bindings:** A CORBA ORB lets the programmer invoke methods on server objects using the high-level language of choice. It doesn't matter what language server objects are written in. CORBA separates interface from implementation and provides language-neutral data types that make it possible to call objects across language and operating systems boundaries.

3. **Self-describing system**: CORBA provides run time meta-data for describing every server interface known to the system. Every CORBA ORB must support an *Interface Repository* that contains real-time information describing the functions a server provides and their parameters. The client use meta-data to discover how to invoke services at run time.

4. **Local/remote transparency**: An ORB can run in stand alone mode or it can be interconnected to every other exiting ORB using CORBA Internet Inter-ORB Protocol (IIOP) services. In general, a CORBA client/server programmer does not have to be concerned with transports, server locations, object activation, byte ordering across dissimilar platforms, or target operating system. CORBA makes it al transparent.

5. **Built-in security and transactions**: The ORB includes context information in its messages to handle security and transactions across machine and ORB boundaries.

6. **Polymorphic messaging**: ORB does not simply invoke a remote function – it invokes a function on a target object. This means that the same function call will have different effects depending on the object that receives it.

7. **Coexistence with existing systems**: CORBA's separation of an object definition from its implementation is perfect for encapsulating existing applications. The programmer can write applications as pure objects and encapsulate existing applications with IDL wrappers.

The key to integrating applications objects is the specification of standard interfaces using IDL. Once all applications and data have an IDL-compliant interface, communication is independent of physical location, platform type, networking protocol, and programming language.

IDL is purely declarative. This means that it provides no implementation details. It is language independent supporting multiple language bindings from a single IDL specification. Standard bindings include C, C++, SmallTalk, Ada95, and others.

Figure 13 shows the principal CORBA interfaces.

In the client side CORBA has the client stubs, the dynamic invocation interface, and the ORB interface:

1. **Client Stubs** provide the static interfaces to the object services. These precompiled stubs define how clients invoke corresponding services on the server. From a client's perspective, the stubs acts like a local call – it is a local proxy for a remote server object. The services are defined using IDL, and both the client and the server

stubs are generated by the IDL compiler. A client must have an IDL stub for each interface it uses on the server.

2. **Dynamic Invocation Interface** lets the programmer discover methods to be invoked at run time.

3. **ORB Interface** consists of a few local services that may be of interest to an application.

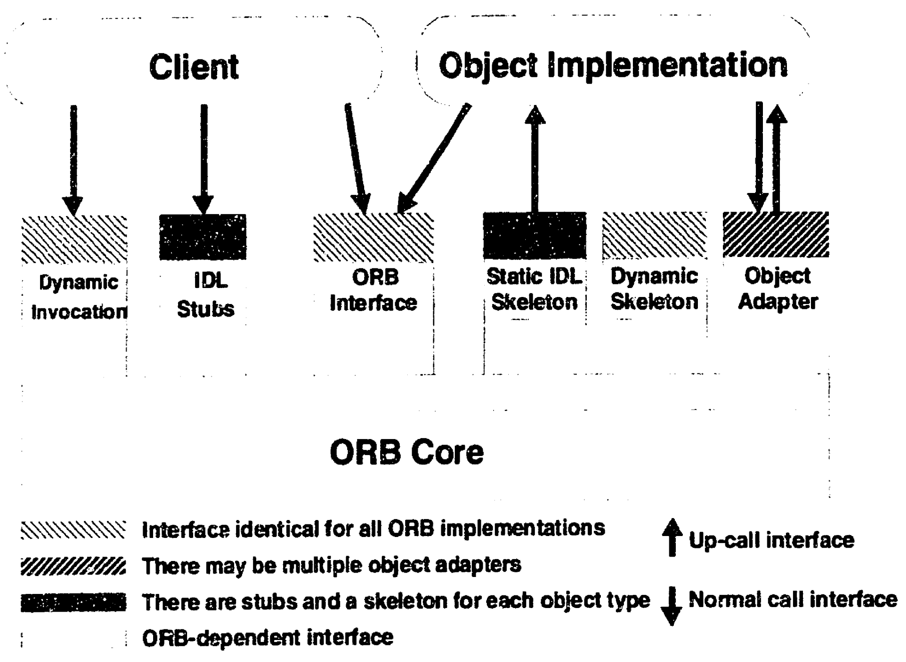


Figure 13 – The principal CORBA interfaces [adapted from OMG, 1997]

The server side cannot tell the difference between static and dynamic invocation; they both have the same message semantics. In both cases, the ORB locates a server object adapter, transmit the parameters, and transfer control to the object implementation through the server IDL stub (or static skeleton). Here is what CORBA principal elements do in the server side (Figure 13):

1. **Static Skeleton** (also known as Server IDL Stubs) provides static interface to each service exported by the server. These stubs, like the ones on the client, are created using an IDL compiler.

2. **Dynamic Skeleton Interface** provides a run-time binding mechanism for servers that need to handle incoming methods calls for components that do not have IDL-based compiled skeletons.

3. **The Object Adapter** sits on top of the ORB's core communication services and accepts requests for services on behalf of the server's object. It provides the run time environment for instantiating server objects, passing requests to them, and assigning them objects Ids – CORBA calls the Ids object references. The object adapter also registers the classes it support and their run-time instances with *the Implementation Repository.*

4. **The ORB Interface** consists of a few services that are identical to those provided on the client side.

In the M-RAM architecture, CORBA's Object Request Broker (ORB) is the object bus. It lets objects transparently make requests to - and receive responses from - other objects locally or remotely. It contains real-time information describing the function a server provides and their parameters.

The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its methods, and return the results. The client just needs to know the object interface.

On the server side, M-RAM agents provide, an IDL interface and a CORBA IIOP server next to the existing HTTP server. On the client side M-RAM provides a CORBA ORB by downloading a Java ORBlet from the server. An ORBlet is simply an ORB written in Java bytecodes

M-RAM enforces collaboration by writing code for the two sides of the collaboration. The next step would be the development of intelligent components that more than just inter operate but instead collaborate at the semantic level to get a job done. The trick, however is to get components that have no previous knowledge of each other collaborate. To get to that point, we will need standards that set the rules of engagement for different component interaction boundaries. The Knowledge Query and Manipulation Language (KQML) [Finin et al., 1992] that is both a message format and a message-handling protocol to support run-time knowledge sharing among agents can provide this standards for the M-RAM model because it focuses on an extensible set of performatives, which defines the permissible "speech acts" agents may use and compromise a substrate on which to develop higher level models of interagent interaction [Finin et al., 1994].

4.4 The C4.5 Agent

The C4.5 computer program construct classification models by discovering and analyzing patterns found in recorded data.

4.4.1 C4.5 Requirements

The key requirements of the C4.5 program are:

1. **Attribute value description:** The data to be analyzed must be what is sometimes called a flat file – all information about one object or case must be expressed in terms of a fixed collection of properties or attributes. Each attribute may have either discrete or numeric values, but the attributes used to describe a case must not vary from one case to another

2. **Predefined classes:** The categories to which cases are to be assigned must have been established beforehand. In the terminology of machine learning, this is defined as supervised learning.

3. **Discrete classes:** The classes must be shapely delineated – a case either does or does not belong to a particular class – and there must be far more cases than classes.

4. **Sufficient data:** Inductive generalization proceeds by identifying patterns in data. As differentiation depends on statistical tests there must be sufficient cases to allow these tests to be effective. The amount of data required is affected by factors such as the number of attributes and classes. A detailed classification model usually requires hundreds or even thousands of training cases.

5. **Logical classification models:** C4.5 construct only classifiers that can be expressed as decision trees or set of production rules.

4.4.2 M-RAM Classes

The demand for tall buildings has led to the continuing development of innovative structural systems which are necessary in order to provide economic solutions. Cost is the dominant factor in the optimization process, and it is essential that reasonable optimal solutions are established early in the design process so that meaningful decisions can be made. The large variety of structural systems and construction materials makes the task of obtaining the optimum solution difficult for the designers.

Structural systems can be defined as an arrangement of structural components and subsystems to properly transmit the forces from the superstructure to the substructure subject to certain constraints.

Due to relative significance of structural cost for tall buildings construction, frequently in order of 20% to 30% of the total cost, an improperly planned and designed structural system may rise the total cost of construction by as much as 5% to 10% [Pouangare, 1986]. Even worse is that the system may not serve its purpose safely and properly.

For the practicing structural engineer, the cataloging of structural systems for tall buildings has historically recognized the primary importance of the system to resist lateral loads [Kowalczyk et al., 1995]. At one end of the spectrum are the moment resisting frames which are efficient for buildings in the range of 20 to 30 stories; at the other end is the generation of tubular system with high cantilever efficiency which are good for 70 to 80 stories. Any particular form is economical only over a limited range of building heights.

Recognizing this historical knowledge, M-RAM applies C4.5 to create a decision tree and a set of rules with the objective of clustering the database and consequently reducing the search space. The database, obtained from the monographs edited by the Council on Tall Buildings and Urban Habitat [Kowalczyk et al. [1995], was divided in five classes according to the system adopted to resist lateral loads: Braced frame and moment resisting frame, shear wall, core and outrigger, tubular and hybrid.

4.4.2.1 Braced Frame and Moment Resisting Frame

Braced frames and moment resisting frames are normally organized as planar assemblies in orthogonal directions to create planar frames or a tube frame system. Both systems are commonly used today as effective means of resisting lateral forces in high-rise construction for building of up to 40 or 50 stories (Figure 14).

- **Braced frames** are cantilevered vertical trusses resisting lateral loads primarily through the axial stiffness of the frame member. The effectiveness of the system, as characterized by a high ratio of stiffness to material quantity, is recognized for multistory buildings in the low to medium height range. Braced frame geometries are grouped, based on their ductility characteristics, as either concentric braced frames (CBF) or eccentric braced frames (EBF). In CBFs the axes of all members intersect at a point such that the member forces are axial. CBFs have a great amount of stiffness but low ductility. Thus in areas of low seismic activity, where high ductility is not essential, CBFs are the first choice on engineers for lateral load resistance. EBFs, on the other hand, utilize axis offsets to introduce flexure and shear into the frame, which lowers the stiffness-to-weight ratio but increases ductility.

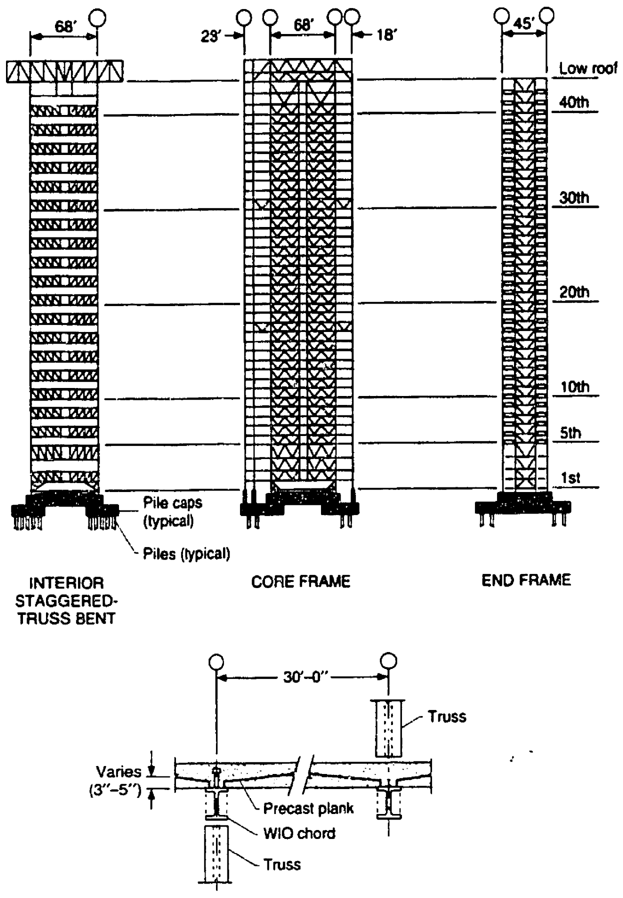


Figure 14 - Framework and typical bay section; Taj Mahal Hotel-USA [adapted from Kowalczyk et al., 1995]

- **Moment resisting frame** consists of horizontal and vertical members rigidly connected together in a planar grid form, which resists lateral loads primarily through the flexural stiffness of the members. Moment resisting frames have the advantages in high-rise construction due to their flexibility in architectural planning. A moment resisting frame may be placed in or around the core, on the exterior, or throughout the interior of the building with minimal constraint on the planning module. Stiffness rather than strength often control the size of the members in a moment resisting frame in order to develop an acceptable drift control under lateral loads. The lateral drift is a function of both the column stiffness and beam stiffness. Moment resisting frames are normally efficient for buildings no more than 20 or 30 story height. The lack of efficiency for taller buildings is due to the moment resisting frame's reliance on lateral load resistance derived primarily through flexure of its members.

The strength and stiffness of the frame is proportional to the column and beam size and inversely proportional to the story height and the column spacing. In general an increase in beam stiffness has a greater effect on the frame stiffness than an increase in column stiffness.

Internally located frames have serious disadvantages that severely limit their usefulness in tall buildings. The floor space requirements of most buildings limit the number of interior columns available for frames. Also the floor beams are generally of long span and limited depth which means that their stiffness is very limited. Externally located frame do not necessary has this disadvantages and it is often possible and even desirable to provide closely spaced columns and deep spandrel beams.

- **Frame-Truss Interacting Systems** are produced when shear trusses are combined with moment resisting frames. The linear wind sway of the moment frame, when combined with the cantilever parabolic sway of the truss, produces enhanced lateral stiffness. Frame-truss interacting systems have a wide range of application to buildings of up to 40 stories in height.

4.4.2.2 Shear wall

Shear wall is a thin slender beam cantilevered vertically to resist lateral loads (Figure 15). For a linear member to be considered as a shear wall the aspect ratio of its section must be greater than 5 and the length greater than 10 times the smaller dimensions of the section.

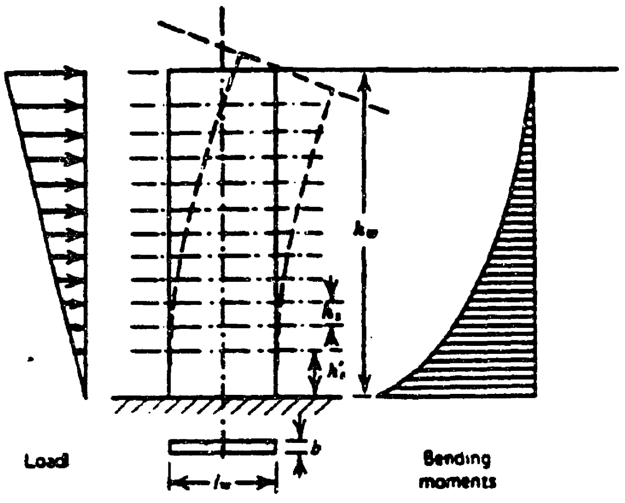


Figure 15 – Shear wall

Although some pure shear-wall systems have been built in the past it is very unusual nowadays to have structural system consisted only of shear walls. Usually rigid frames are connected with the shear walls this way evolving the frame-shear-wall system.

With the advent of reinforced concrete, shear wall systems have become widely used to stabilize efficiently high rise buildings. A common shear wall system groups shear walls around services cores, elevators shafts, and stairwells to form a stiff box-type structure (Figure 16). The shear walls are designed to cantilever from the foundation level. The bending, shear, and warping stresses due to wind or earthquake loads are combined with stresses due to gravity loads. Individual walls within the box system can then be designed as unit-length walls spanning either floor to floor or between return walls. Multiple shear walls throughout a tall building may be coupled to provide additional frame action and hence increase overall building stiffness.

Frame-shear-wall structural system becomes inefficient above 40 stories since large amount of material is needed to make the shear wall (either concrete or steel braces) sufficient stiff and strong

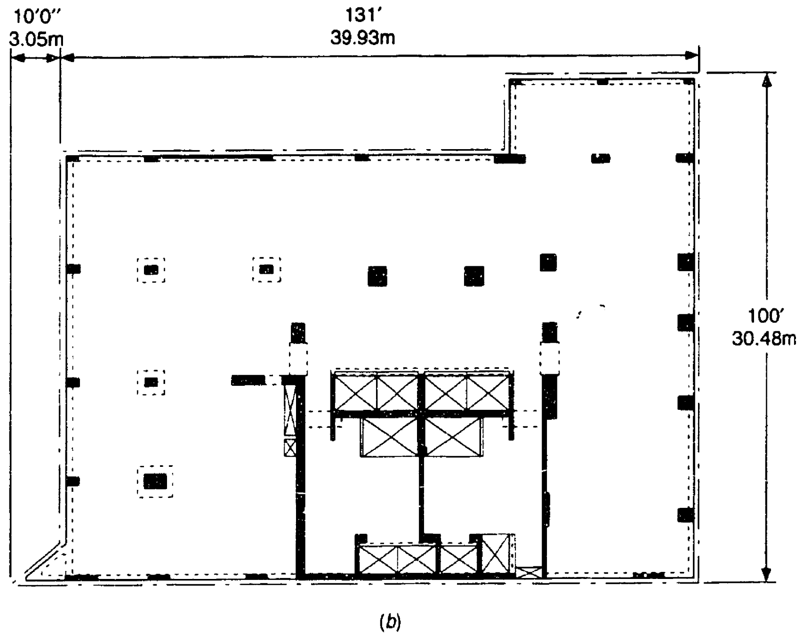


Figure 16 - Meandering shear wall; Metropolitan Tower-USA [adapted from Kowalczyk et al., 1995]

4.4.2.3 Core and outrigger

In a high-rise building the core can be related to the mast of a ship, with the outrigger acting like the spreaders and the exterior columns like the stays or shrouds. Outriggers serve to reduce the overturning moment in the core that would otherwise act as pure cantilever, and to transfer the reduced moment to columns outside the core by way of tension-compression couple, which takes advantage of the increased moment arm between these columns (Figure 17).

Ass said in section 4.4.2.2 frame-shear-wall structural systems becomes inefficient above 40 stories. The efficiency of the structural system can be improved up to 30% by using belt trusses or outrigger trusses to tie the frame and the core.

For many buildings, the answer to the problems and restrictions of core-only or tubular structures is the incorporation of one or more levels of outriggers. Typical outrigger organization consists of linking the core of a high-rise building to the exterior columns on one or more building faces with truss or wall elements. The outrigger system may be formed in any combination of steel, concrete, or composite construction.

The trusses are fixed rigidly to the core and are simply connected or fixed to the exterior columns. When the shear core tries to bend, the belt or outrigger trusses act as levers arms that induces axial forces into the perimeter columns. These columns in turn act as struts to resist the deflection of the core. The core fully develops the horizontal shears and the truss transfer the vertical shear from the core to the façade frame.

The frame with shear and outrigger trusses systems is applicable to building from 30 to 55 floors.

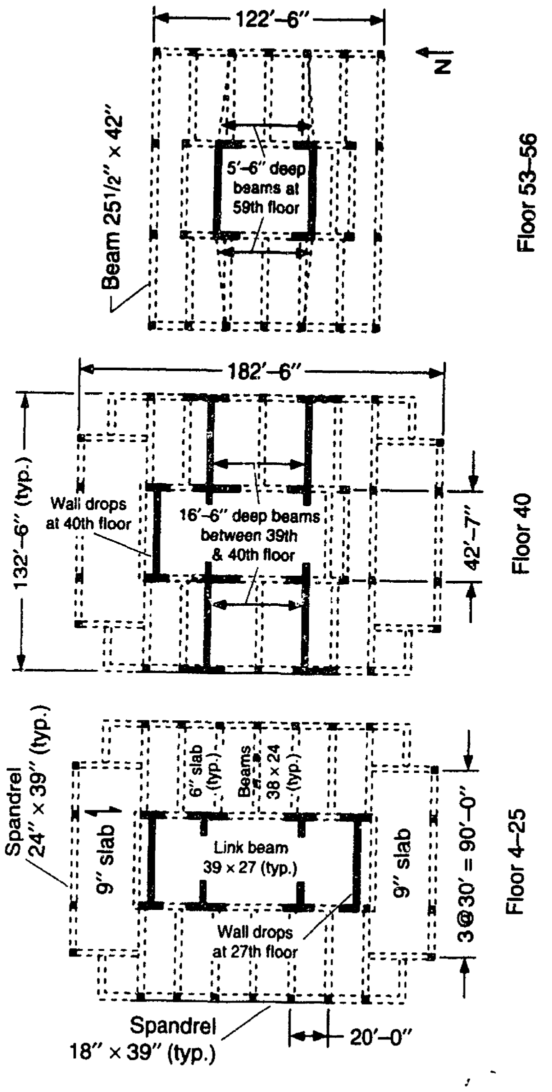


Figure 17 - Typical floor plans; Two Prudential Plaza-USA [adapted from Kowalczyk et al., 1995]

4.4.2.4 Tubular

Tubular systems (Figure 18) can be subdivided in frame tube, trussed tube, and bundled tube:

- **Framed tube:** Framed tube is a logical extension of the moment resisting frame, whereby the beam and column stiffness are increased dramatically by reducing the clear span dimensions and increasing the member depths. The organization of the

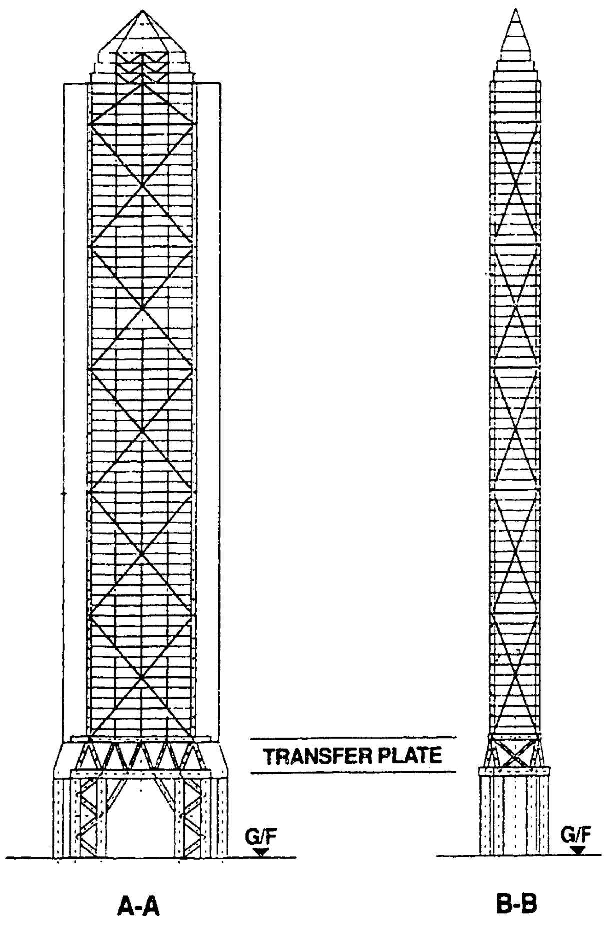


Figure 18 - Structural steel schema; Central Plaza-Hong Kong [adapted from Kowalczyk et al., 1995]

framed tube system is generally one of closely spaced exterior columns and deep spandrel beams rigidly connected together, with the entire assemblage continuous along each façade and around the building corners.

Framed tubular design assumes that the façade structure responds to lateral loads as a closed hollow box cantilevering out of the ground. This façade structure looks like a perforated wall. The rigidity of the tube is so hight that it respondes to lateral loading similar to a cantilever beam. The interior columns are assumed to carry gravity loads only and do not contribute to the exterior tube's stiffness. The monolithic nature of reinforced concrete construction is ideally suited for such system that involves fully continuous interconnections of the frame members.

Framed tube system is economically applicable from40 to 70 floors.

- **Trussed tube:** Trussed tube is a solution adapted to the qualities and characteristics of structural steel. The objective of these systems is to form a rigid box to resist lateral shears by applying axial forces in its members rather than through flexure. This rigid box is achieved by introducing a minimum number of diagonals on each facade and making the diagonals intersect at the same point at the corner column. The system is tubular in that the fascia diagonals not only form a truss in the plane but also interact with the trusses on the perpendicular faces to affect the tubular behavior. This creates the X form between corner columns on each façade.

  Trussed tube system is economically applicable from 40 to 80 stories.

- **Bundled tube:** The need for vertical planning modulation and the control of shear lag for very tall buildings led to the development of the bundled, or modular, tube concept. This structural system rely in the notion that if the overall tubular shape is structured into small cells by the inclusion of interior tubular lines, then the overall structure would have the cantilever efficiency associated with that of a single cell with high aspect ratio. The interior tubular web line serves to reduce the effects of shear lag on the column in the middle of the windward and leeward flange frames.

The exterior framed tube is stiffened by interior cross diaphragms in both directions. This way as assemblage of cell tubes is formed. These individual tubes are independently strong therefore may be bundled in any configuration and discontinued at any level. A further advantage of this system lies in the extremely large floor areas that may be enclosed.

The interior tube diaphragms act as webs of huge cantilever beam resisting shear force thus minimizing shear lag and in addition they provide strength against bending.

This structural system is economically applicable from 55 to 110 stories.

4.4.2.5 Hybrid

Tall buildings have been traditionally designed to make use of a single type of lateral load resisting systems mainly because until recently structural systems had to be amenable to hand calculation or computer analysis using limited-capacity machines. Nowadays with large computational power decision on structural systems can be made on the basis of their effects on the appearance and functioning of the building and on its constructability. Regardless of this fact designer must still be aware of the risks and pitfalls of creating abrupt discontinuities in building stiffness, the long-term effects of differential axial shortening, and other possible side effects when using mixed systems and multiple materials. Examples are composite or mixed systems with steel and concrete. Figure 19 show a structural detail of a building that was designed with a Hybrid structural system.

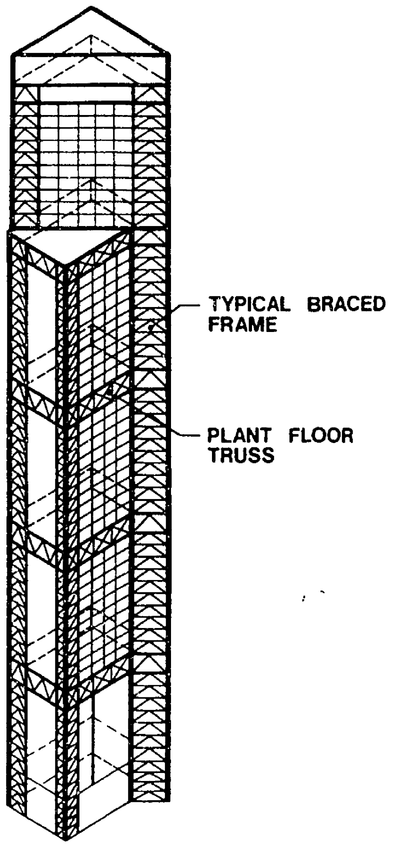


Figure 19 - Structural steel system; Overseas Union Bank Center-Singapore [adapted from Kowalczyk et al., 1996]

4.4.3 The M-RAM C4.5 Implementation

In the M-RAM C4.5 [Quinlan, 1993] implementation each case concerns one design. Nineteen properties have been used to describe each case: (1) Height from street to roof, (2) number of stories, (3) number of levels below ground, (4) building use, (5) frame material, (6) typical floor life load, (7) basic wind velocity, (8) maximum lateral deflection, (9) design fundamental period – transverse, (10) design fundamental period – longitudinal, (11) design acceleration, (12) design damping, (13) earthquake loading, (14) typical floor story height, (15) typical floor beam span, (16) typical floor beam depth, (17) typical floor beam spacing, (18) columns size at ground floor, and (19) columns spacing. The outcome of each case is given as braced frame and moment resisting frame, shear wall, core and outrigger, tubular or hybrid depending on which kind of structural system was applied to resist lateral loads.

From the available cases, C4.5 acting as a learning system constructed a classification model that relates the type of structural system to resist lateral loads to the values of the recorded properties. There are 46 cases of actual built tall buildings, in the dataset. We recognize the fact that this is not sufficient data to induce a good generalization. Robust pattern cannot be distinguished from chance coincidences but the objective in this stage of the research was to define the viability of the proposed system architecture. The next step in this research is to augment the dataset; consulting engineers and computer simulation can be used to accomplish this.

The first step required to run C4.5 is the creation of a file with definition of classes and attributes. The file tese-design.names (figure 20) specifies the classes and the name and

```
|Date: Thursday, 01 May 1997
|Author: Lucio Soibelman
|E-mail: lucio@mit.edu
|
| Title: M-RAM - C4.5 Module
|
| Source of Information
|
| Tall Buildings and Urban Environment Series
| Structural Systems for Tall Buildings
| Council on Tall Buildings and Urban Habitat
|                                    Committee 3
| McGraw-Hill, Inc
|----------------------------------------------------------------
|  Classes - Classifies according lateral load resisting Systems
|  -------

Braced Frame and Moment Resisting Frame, Shear Wall, Core and
Outrigger, Tubular, Hybrid.

|  Attributes
|  ----------
Height from street to roof:                 continuous
Number of stories:                          continuous
Number of levels below ground:              continuous
Building Use:                               office, hotel,
residential, garage, mixed
Frame material:                             steel, concrete, mixed
Typical floor live load:                    continuous
Basic wind velocity:                        continuous
Maximum lateral deflection:                 continuous
Design fundamental period-transverse:       continuous
Design fundamental period-longitudinal:     continuous
Design acceleration:                        continuous
Design Dumping:                             continuous
Earthquake loading:                         continuous
Typical floor story height:                 continuous
Typical floor beam span:                    continuous
Typical floor beam depth:                   continuous
Typical floor beam spacing:                 continuous
Columns size at ground floor:               continuous
Columns spacing:                            continuous
```

Figure 20 – File tese-design.names

description of each attribute. Some attributes have numerical values and are described just as continuous; others have a small set of possible values.

The next required step is to create a file with information on all individual cases available in the dataset. This involves spelling out the attributes for each case, separated by commas, and followed by the case's class. Four of the M-RAM cases are:

```
99.7,25,4,office,steel,3,?,?,3,3,40,2,0.1,3.84,24,850,3.15,1600,3.15,Braced Frame and
Moment Resisting Frame
218,68,2,mixed,concrete,2.5,47,436,4,5,15,2.5,?,3.45,?,508,?,?,?,Shear Wall
342,82,5,office,steel,4,?,855,?,?,?,?,?,3.86,13.7,965,3.05,?,3.05,Tubular
288,76,6,mixed,mixed,2.5,34,483,5.3,5.3,20,2.5,0.03,3.5,?,?,?,89304,?,Hybrid
```

Unknown or inapplicable values are indicated by question marks. The first case has height from the street to the roof equal to 99.7 meters, 25 stories, 4 underground levels, is an office building, and so on.

The C4.5 program generates a classifier in the form of a decision tree (Figures 21, 22 and 23) where a leaf indicates classes and decision nodes specifies some test to be carried out on a single attribute value, with one branch and sub-tree for each possible outcome of the test. A decision tree can be used to classify a case by starting at the root of the tree and moving through it until a leaf is encountered. The output of the decision tree generator appears in figures 21 and 22. Figure 23 shows a graphical representation of this tree.

The number in parentheses following each leaf (Figures 21 and 22) indicate the number of training cases associated with each leaf and the number of them misclassified by the leaf.

The program also contains heuristic methods for simplifying decision trees, with the aim to producing more comprehensible structures without compromising accuracy on unseen cases. The second section of the output (Figure 22) gives the simplified tree.

```
C4.5 [release 5] decision tree generator     Sun Mar 22 05:00:46 1998
----------------------------------------

    Options:
        File stem <tese-design>

Read 46 cases (19 attributes) from tese-design.data

Decision Tree:

Typical floor story height <= 3.2 : Braced Frame and Moment Resisting Frame (4.0)
Typical floor story height > 3.2 :[S1]


Subtree [S1]

Maximum lateral deflection > 948 : Braced Frame and Moment Resisting Frame
(2.2/0.1)
Maximum lateral deflection <= 948 :[S2]


Subtree [S2]

Height from street to roof <= 128 : Braced Frame and Moment Resisting Frame
(2.9/0.9)
Height from street to roof > 128 :
|   Height from street to roof <= 160 : Shear Wall (3.0/1.0)
|   Height from street to roof > 160 :
|   |   Typical floor story height > 4 : Hybrid (2.9/0.9)
|   |   Typical floor story height <= 4 :
|   |   |   Maximum lateral deflection <= 436 :
|   |   |   |   Maximum lateral deflection > 400 : Shear Wall (2.0)
|   |   |   |   Maximum lateral deflection <= 400 :
|   |   |   |   |   Maximum lateral deflection <= 185 : Shear Wall (2.0/1.0)
|   |   |   |   |   Maximum lateral deflection > 185 :
|   |   |   |   |   |   Columns spacing <= 6.8 : Tubular (5.0)
|   |   |   |   |   |   Columns spacing > 6.8 :
|   |   |   |   |   |   |   Design acceleration <= 16 : Tubular (2.0)
|   |   |   |   |   |   |   Design acceleration > 16 : Core and Outrigger (2.0)
|   |   |   Maximum lateral deflection > 436 :
|   |   |   |   Design acceleration <= 19 : Core and Outrigger (4.9/1.4)
|   |   |   |   Design acceleration > 19 :
|   |   |   |   |   Columns size at ground floor <= 6197.6 : Tubular (6.3/1.1)
|   |   |   |   |   Columns size at ground floor > 6197.6 :
|   |   |   |   |   |   Basic wind velocity <= 41 : Hybrid (4.3/0.6)
|   |   |   |   |   |   Basic wind velocity > 41 : Tubular (2.4/0.7)
```

Figure 21 - Output of the decision tree generator – part I

```
Simplified Decision Tree:

Typical floor story height <= 3.2 : Braced Frame and Moment Resisting
Frame (4.0/1.2)
Typical floor story height > 3.2 :[S1]


Subtree [S1]

Maximum lateral deflection > 948 : Braced Frame and Moment Resisting Frame
(2.2/1.1)
Maximum lateral deflection <= 948 :[S2]


Subtree [S2]

Height from street to roof <= 128 : Braced Frame and Moment Resisting
Frame (2.9/2.0)
Height from street to roof > 128 :
|   Height from street to roof <= 160 : Shear Wall (3.0/2.1)
|   Height from street to roof > 160 :
|   |   Typical floor story height > 4 : Hybrid (2.9/2.0)
|   |   Typical floor story height <= 4 :
|   |   |   Maximum lateral deflection <= 436 :
|   |   |   |   Maximum lateral deflection > 400 : Shear Wall (2.0/1.0)
|   |   |   |   Maximum lateral deflection <= 400 :
|   |   |   |   |   Columns spacing <= 6.8 : Tubular (6.0/1.2)
|   |   |   |   |   Columns spacing > 6.8 :
|   |   |   |   |   |   Design acceleration <= 16 : Tubular (3.0/2.1)
|   |   |   |   |   |   Design acceleration > 16 : Core and Outrigger
(2.0/1.0)
|   |   |   Maximum lateral deflection > 436 :
|   |   |   |   Design acceleration <= 19 : Core and Outrigger (4.9/2.6)
|   |   |   |   Design acceleration > 19 :
|   |   |   |   |   Columns size at ground floor <= 6197.6 : Tubular
(6.3/2.4)
|   |   |   |   |   Columns size at ground floor > 6197.6 :
|   |   |   |   |   |   Basic wind velocity <= 41 : Hybrid (4.3/1.8)
|   |   |   |   |   |   Basic wind velocity > 41 : Tubular (2.4/1.7)


Tree saved


Evaluation on training data (46 items):

        Before Pruning          After Pruning
       ----------------  --------------------------
       Size      Errors  Size        Errors  Estimate

         27    7(15.2%)     25    7(15.2%)    (48.2%)   <<
```

Figure 22 - Output of the decision tree generator – part II

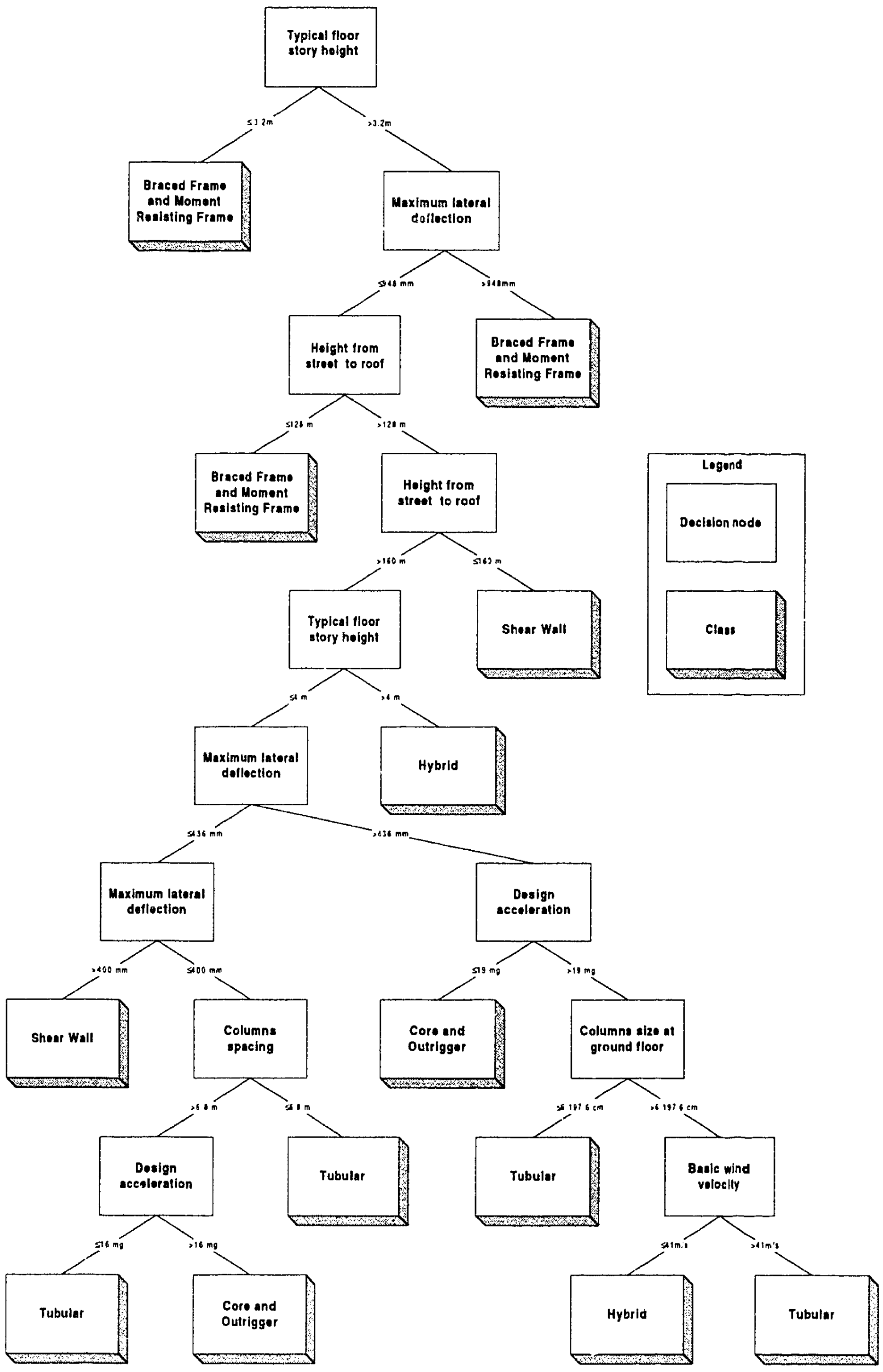


Figure 23 – Graphical representation of the M-RAM C4.5 tree

After presenting the simplified tree, the output shows how both the original and simplified trees perform on the training set from which they were constructed (Figure 22). The original tree of 27 nodes misclassifies 7 of 46 cases. The simplified tree also misclassifies just 7 training cases but the program predicts that it will have a much larger error rate of 48.2% on unseen cases.

4.4.4 C4.5 Decision Tree Construction Method

Quinlan [1993] applied in C4.5 an elegantly simple method for constructing a decision tree from a set T of training cases. Let the classes be denoted $\{C_1, C_2,...,C_K\}$. There are then 3 possibilities:

1. T contains one or more cases, all belonging to a single class C_j:

   The decision tree for T is a leaf identifying class C_j.

2. T contains no case:

   The decision tree is again a leaf, but the class to be associated with the leaf must be determined from information other than T. For example, the leaf might be chosen in accordance with some background knowledge of the domain, such as overall majority class. C4.5 uses the most frequent class at the parent of this node.

3. T contains cases that belong to a mixture of classes:

   In this situation, the idea is to refine T into subsets of cases that are, or seem to be heading towards, single-class collections of cases. A test is chosen, based on a single attribute, that has one or more mutually exclusive outcomes $\{O_1, O_2, ..., O_n\}$. T is partitioned into subsets $T_1, T_2, ..., T_n$ where T_i contain all the cases in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test, and one branch for each possible outcome. The same tree-building machinery is applied recursively to each subset of training cases, so that the *i*th branch leads to the decision tree constructed from the subset T_i of training cases.

The above method depends in a great deal on the choice of appropriate evaluating test. First there is the necessity of a test to decide which attribute will be tested. This is done by a gain ration criterion that will be described bellow. Then, if the chosen attribute is discrete, the test create one outcome and branch for each possible value of that attribute or, if the chosen attribute A has a continuous numeric value, a binary test with outcome $A < Z$ and $A > Z$, based on comparing the value A against a threshold value Z.

To calculate the value of the threshold Z the training cases T are first sorted on the values of the attribute A being considered. There are only a finite number of those values, so let denote them as $\{v_1, v_2, \ldots, v_m\}$. There are thus only $m-1$ possible splits on A, all of which are examined by C4.5. The midpoint of each interval is chosen as the representative threshold, the *i*th such being

$$\frac{v_i + v_{i+2}}{2}$$

The gain criterion ratio applied to define the attribute in which the test is going to be carried out is based in the idea that the information conveyed by a message depends on its probability and can be measured in bits as minus the logarithm to base 2 of that probability. Selecting one case at random from a set S of cases and announcing that it belongs to some class C_j has the probability of

$$\frac{freq\ (C_j,S)}{|S|}$$

Where $freq\ (C_j,S)$ stand for the number of cases in S that belong to class C_j and $|S|$ denotes the number of cases in the set S.

The information that this message conveys is

$$-\ log_2\ (freq\ (C_j,S)\ /\ |S|)\ \text{bits}$$

To find the expected information from such a message pertaining to class membership, C4.5 sum over the classes in proportion to their frequencies in S, giving

$$Info(S) = -\sum_{j=1}^{k} (freq\ (C_j,S)\ /\ |S|)\ x\ log_2\ (freq\ (C_j,S)\ /\ |S|)\ \text{bits}$$

When applied to the set of training cases $info(T)$ measures the average amount of information needed to identify the class of a case in T.

Considering a similar measurement after T has been partitioned in accordance to the n outcomes of the test X. The expected information required can be found as the weighted sum over the subsets, as

$$info_X(T) = \sum_{i=1}^{n} |T_i|\ /\ |T|\ x\ info(T_i)$$

The quantity

$$gain(X) = info(T) - info_X(T)$$

Measures the information that is gained by partitioning ***T*** in accordance with the test ***X***.

Finally, a normalization is carried out in the ***gain(X)*** equation to avoiu a bias in favor of tests with many outcomes. So,

$$split\ info(X) = - \sum_{i=1}^{n} |T_i| / |T| \times \log_2(|T_i| / |T|)$$

This represents the potential information generating by dividing T into a subset. Then,

$$Gain\ ratio(X) = gain(X) / split\ info(X)$$

Expresses the proportion of information generated by the split that is useful, or in other words, that is helpful for classification.

4.5 The CBR Agent

In the M-RAM implementation, the manager, after receiving the classification from the C4.5 agent, requests the CBR agent to retrieve the most useful case. The CBR agent returns to the manager the home page of the retrieved case (Figure 24). From this page, the designer can follow the links to a complete information about the retrieved case (Figure 25, 26 and 27).

Caspian, a case-based reasoning system written in C and developed in the University of Wales – Aberrystwith, was chosen to be implemented as the M-RAM CBR Agent.

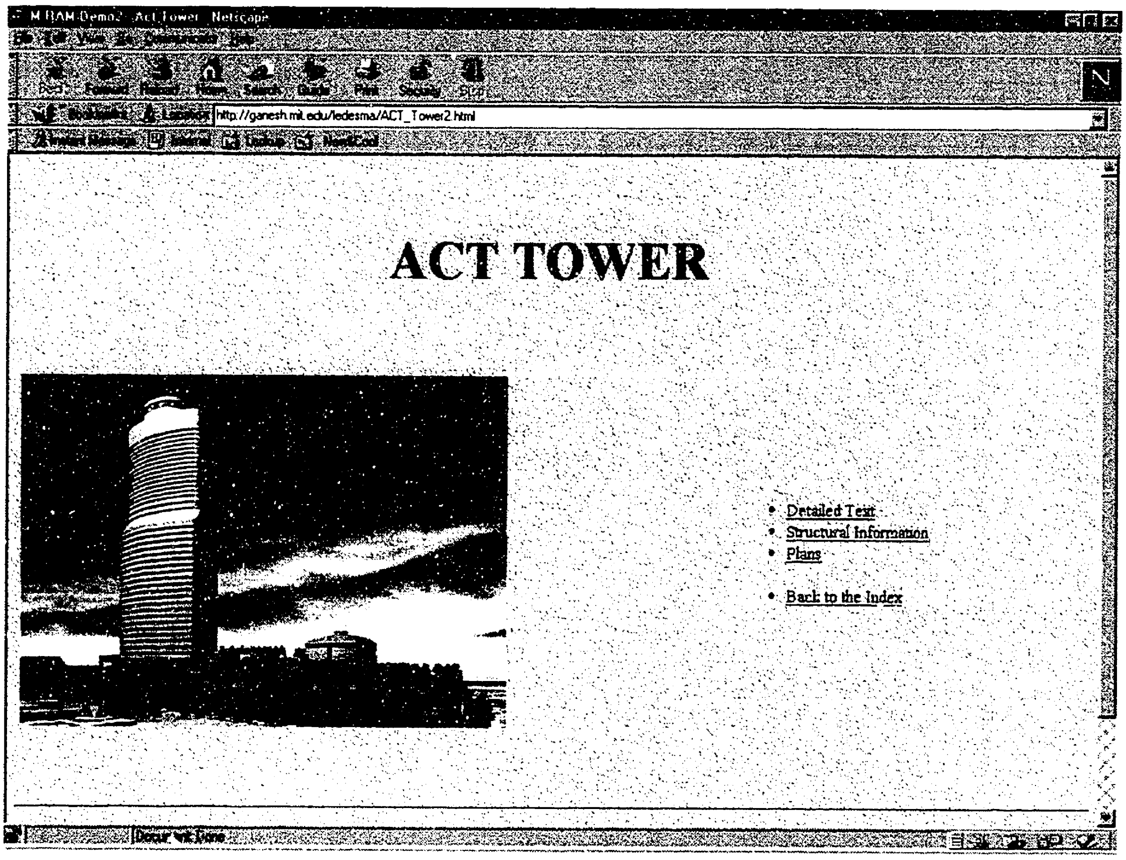
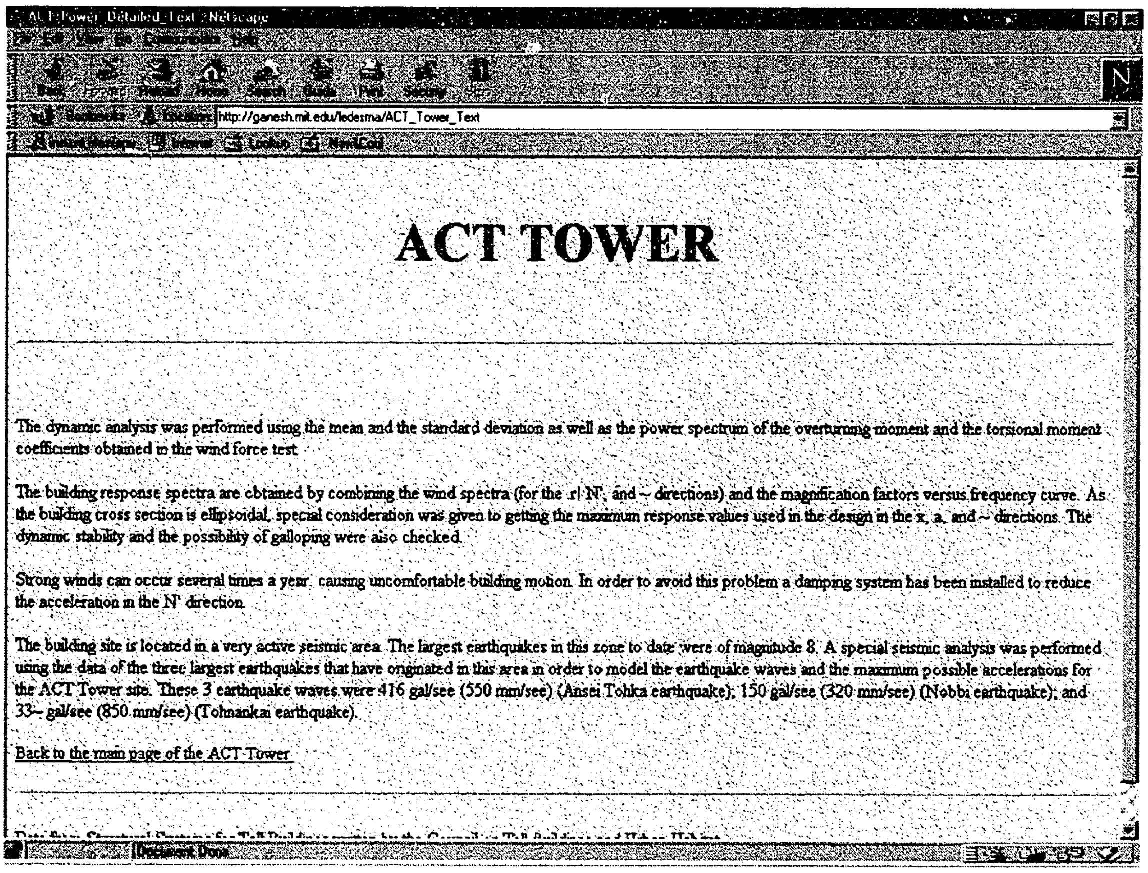


Figure 24 – Home page of the retrieved case


http://ganesh.mit.edu/ledesma/ACT_Tower_Text

ACT TOWER

The dynamic analysis was performed using the mean and the standard deviation as well as the power spectrum of the overturning moment and the torsional moment coefficients obtained in the wind force test.

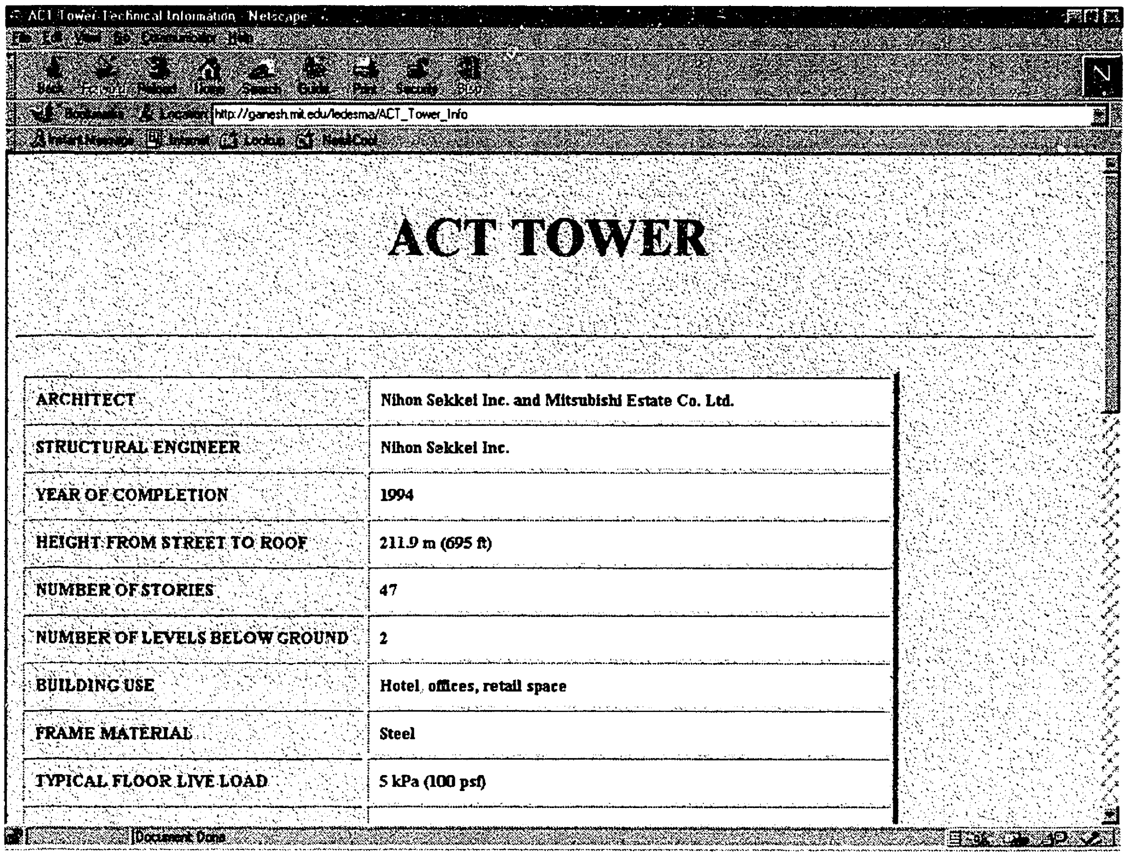
The building response spectra are obtained by combining the wind spectra (for the .r| N', and ~ directions) and the magnification factors versus frequency curve. As the building cross section is ellipsoidal, special consideration was given to getting the maximum response values used in the design in the x, a, and ~ directions. The dynamic stability and the possibility of galloping were also checked.

Strong winds can occur several times a year, causing uncomfortable building motion. In order to avoid this problem a damping system has been installed to reduce the acceleration in the N' direction.

The building site is located in a very active seismic area. The largest earthquakes in this zone to date were of magnitude 8. A special seismic analysis was performed using the data of the three largest earthquakes that have originated in this area in order to model the earthquake waves and the maximum possible accelerations for the ACT Tower site. These 3 earthquake waves were 416 gal/see (550 mm/see) (Ansei Tohka earthquake); 150 gal/see (320 mm/see) (Nobbi earthquake); and 33~ gal/see (850 mm/see) (Tohnankai earthquake).

Back to the main page of the ACT Tower

Figure 25 – Text description of retrieved case



ACT Tower Technical Information - Netscape

http://ganesh.mit.edu/ledesma/ACT_Tower_Info

ACT TOWER

ARCHITECT	Nihon Sekkei Inc. and Mitsubishi Estate Co. Ltd.
STRUCTURAL ENGINEER	Nihon Sekkei Inc.
YEAR OF COMPLETION	1994
HEIGHT FROM STREET TO ROOF	211.9 m (695 ft)
NUMBER OF STORIES	47
NUMBER OF LEVELS BELOW GROUND	2
BUILDING USE	Hotel, offices, retail space
FRAME MATERIAL	Steel
TYPICAL FLOOR LIVE LOAD	5 kPa (100 psf)

Document: Done

Figure 26 – Structural information of retrieved case

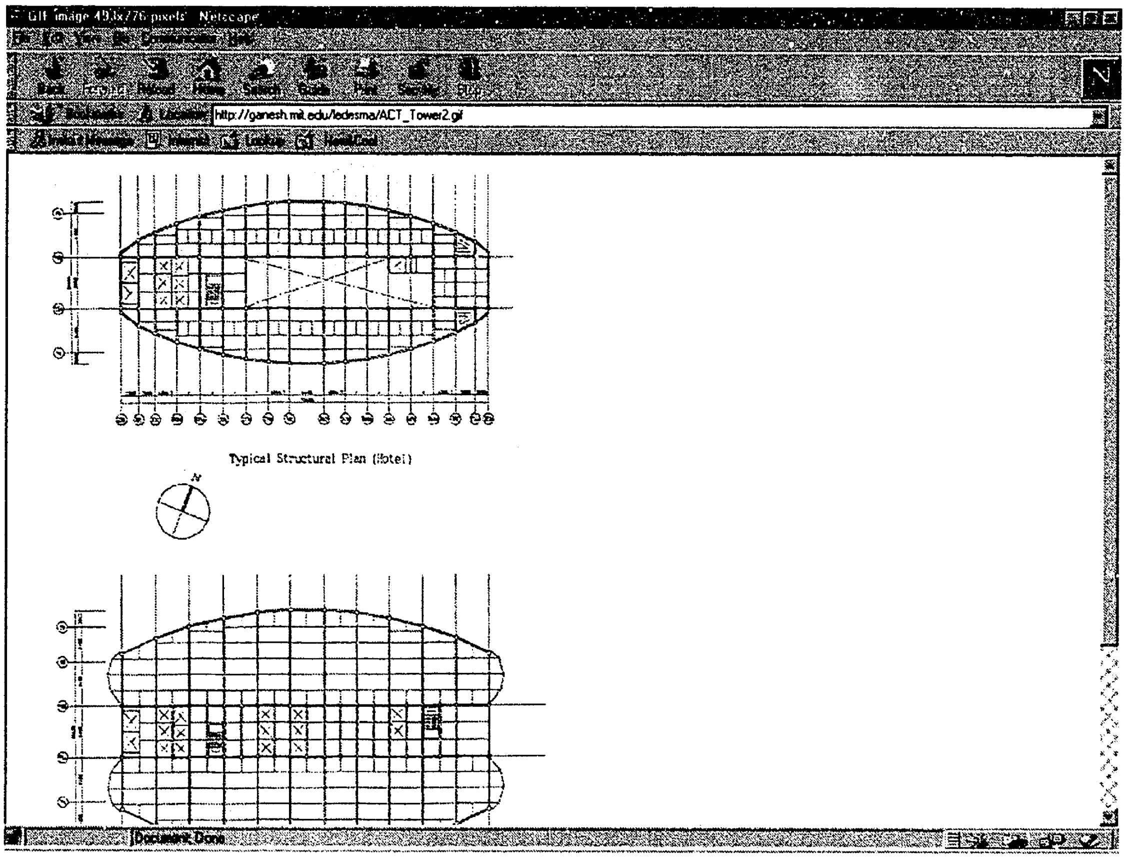

GIF image 493x776 pixels - Netscape
http://ganesh.mit.edu/ledesma/ACT_Tower2.gif
Typical Structural Plan (Hotel)
N
Document: Done

Figure 27 – Structural details of retrieved case

4.5.1 Caspian

Caspian creates a case-base from a case file written in the language CASL [UW, 1996a and UW, 1996b]. The contents of a case-base are described in a file known as a case file. The program Caspian uses this case file to create a case-base in the computer memory, which can then be accessed and modified in order to retrieve the most suitable design solution to the problem being solved using case-based reasoning techniques. When new cases are added to the case-base they are appended to the end of this case file.

Caspian requires the cases to be indexed according to one or more attributes. The program searches the case-base for the subset of cases that matches all the index constraints exactly. In the M-RAM implementation the cases were indexed according to its structural system type. The program then searches the case-base for all cases that match the class previously defined by the M-RAM classification agent.

After the index search has been completed successfully Caspian can be run in two different modes: with or without case matcher. If the case matcher is invoked the program scans the subset of cases to find the one with the highest weight value. If the case matcher is not invoked the program retrieves the whole subset of cases that matches the indexed constraints. In the M-RAM implementation Caspian is run in both modes. First the program invokes the case matcher to retrieve the case that is the best fit to the design problem being analyzed then the program is run again without the case matcher with the objective to retrieve all cases that matches the class defined by M-RAM's classification agent. The retrieved cases are then used as the initial generation in the M-RAM adaptation agent.

4.5.2 Caspian CASL Language

In CASL, a case is similar to a record in a database. The basic unit is a field, which may contain a string, a number, an enumeration symbol or list. A list may be of any length and an element of a list may be of any of the four basic types.

There are a number of differences between a CASL case and an ordinary database record:

1. There are two groups of fields in a case. The first group describes the situation and the nature of the problem. The second group describes the solution to the problem. In the M-RAM implementation the solution is a URL address linking to a World Wide Web page with textual and graphical information about the retrieved case.

2. It is possible for a field to be omitted. This is only true for those fields, which are not used for indexing. Both groups must contain at least one field and the problem section must use at least one field as index.

3. A case has a name associated with it.

4. There are some extra (optional) sections which may be associated with a case

4.5.3 The Case File

The case file consists of a number of blocks of code. The overall syntax is:

- [Introduction]
- Case definition
- Index definition
- [Modification definition]
- [Pre-processing rule definition]
- [Repair rule definition]
- Case Instance {case instance}

[] Representing optional blocks

The introduction block contains introductory text, which gets displayed when the program Caspian has finished checking the case file (Figure 28).

```
introduction is

       '\t\t\tM-RAM CBR Demo',
       '\t\t\t-------- ------ ----\n',
       '\tthis case base reasoning system',
       '\tdemonstrates the conceptual design',
       '\tfor tall buildings.'

end;
```

Figure 28 – M-RAM CASL introduction block

The case definition block defines the types and the weights of the problem fields that may appear in a case. The information in the case definition block is used for type checking cases while weights are used to aid the case-matching process (Figure 29).

```
case definition is

        field height_from_street_to_roof type is number weight is 3;

        field number_of_stories type is number weight is 1;

        field number_of_levels_below_ground type is number weight is 0;

        field building_use type is
                (office,hotel,residential,garage,mixed) weight is 1;


        field frame_material type is
                (steel,concrete,mixed) weight is 1;

        field typical_floor_life_load type is number weight is 0;

        field basic_wind_velocity type is number weight is 0;

        field maximum_lateral_deflection type is number weight is 0;

        field design_fundamental_period_transverse type is number weight is 0;

        field design_fundamental_period_longitudinal type is number weight is 0;

        field design_acceleration type is number weight is 0;

        field design_dumping type is number weight is 0;

        field earthquake_loading type is number weight is 0;

        field typical_floor_story_height type is number weight is 2;

        field typical_floor_beam_span type is number weight is 0;

        field typical_floor_beam_depth type is number weight is 0;

        field typical_floor_beam_spacing type is number weight is 0;

        field columns_size_at_ground_floor type is number weight is 0;

        field columns_spacing type is number weight is 0;

        field structural_system type is

        (braced_frame_and_moment_resisting_frame,shear_wall,core_and_outrigger,
                 tubular,hybrid) weight is 0;


end;
```

Figure 29 – M-RAM CASL case definition block

The purpose of the case definition block is to define the problem fields contained in a case. It consists of a series of field definitions, each of which defines the name, type and optionally the weight of each field and also the prompt.

Only the fields defining the problem part of a case are defined in the case definition. Fields belonging to the solution part of a case may have any name (except those belonging to problem fields) and any type. Whenever Caspian encounters a field name not defined in the case definition it assumes that it is either a solution field or an anonymous field. Hence the fields contained in the solution part of a case may vary from one case to another.

The weight value is used in matching cases. The larger the weight, the more important the field is.

The index definition block defines the field used as indexes when searching for a marching case. A case base should have at least one field as an index. The type of an index field must be an enumeration type (Figure 30).

```
index definition is


      index on structural_system;


end;
```

Figure 30 – M-RAM CASL index definition block

The purpose of the index definition is to define which fields are to be used as indexes. This information is used by Caspian to generate an index structure to improve the search. Indexes are intended to prune the matching process so that many cases are rejected early.

The modification definition block defines the modification rules. The purpose of the modification definition is twofold:

1. It provides the means of specifying that certain symbols or numbers are similar for matching purpose.

2. It provides a means of specifying symbols as abstractions of others. This is useful for making the search more general or for defining generalized cases.

The modification definition block (Figure 31) consists of a list of definitions known as the modification rules. The concern here is with defining certain values to be similar,

```
modification definition is


      field height_from_street_to_roof similar range 0 to 90;
      field height_from_street_to_roof similar range 80 to 120;
      field height_from_street_to_roof similar range 110 to 150;
      field height_from_street_to_roof similar range 140 to 180;
      field height_from_street_to_roof similar range 170 to 210;
      field height_from_street_to_roof similar range 200 to 240;
      field height_from_street_to_roof similar range 230 to 270;
      field height_from_street_to_roof similar range 260 to 300;
      field height_from_street_to_roof similar range 290 to 450;

end;
```

Figure 31 – M-RAM CASL modification definition block

thus guiding the matching process, rather than performing modification to the retrieved solution.

The pre-processing rule definition block defines rules, which are used to alter the case input by the user for matching against or to alter the weight values used for comparing cases. It is used to apply rules to the user case (instead of the repair-case) before the search is carried out. This block was not used in the M-RAM implementation.

The repair rule definition block contains the repair rules. The repair rules are used to modify the solution retrieved from the case-base, to make it more suitable for the current situation. In the conceptual design problem domain is very difficult to create those repair rules because it requires taking into consideration not just the representation of the design cases but the representation of the knowledge needed to adapt the design. This block was not used in the M-RAM implementation because it requires the incorporation of large amounts of domain knowledge where all possible adaptation scenarios must be foreseen and recognized in order for the adaptation result in feasible solutions. M-RAM relied in its adaptation agent to provide this functionality minimizing the need for extensive generalized knowledge for design case adaptation.

The last sets of blocks in the case file are the case instances. These are the cases that make up the case-base (Figure 32).

```
case instance sanwa_bank is

height_from_street_to_roof = 99.7;
number_of_stories = 25;
number_of_levels_below_ground = 4;
building_use = office;
frame_material = steel;
typical_floor_life_load = 3;
design_fundamental_period_transverse = 3;
design_fundamental_period_longitudinal = 3;
design_acceleration = 40;
design_dumping = 2;
earthquake_loading = 0.1;
typical_floor_story_height = 3.84;
typical_floor_beam_span = 24;
typical_floor_beam_depth = 850;
typical_floor_beam_spacing = 3.15;
columns_size_at_ground_floor = 1600;
columns_spacing = 3.15;
structural_system = braced_frame_and_moment_resisting_frame;



solution is
        building = ['http://m-ram.mit.edu/sanwa/'];


end;


case instance act_tower is

height_from_street_to_roof = 221.9;
number_of_stories = 47;
number_of_levels_below_ground = 2;
building_use = mixed;
frame_material = steel;
typical_floor_life_load = 5;
basic_wind_velocity = 30;
maximum_lateral_deflection = 1000;
design_fundamental_period_transverse = 4.52;
design_fundamental_period_longitudinal = 4.73;
design_acceleration = 52;
design_dumping = 1.5;
earthquake_loading = 0.06;
typical_floor_story_height = 3.58;
typical_floor_beam_span = 17.5;
typical_floor_beam_depth = 850;
typical_floor_beam_spacing = 6.4;
columns_size_at_ground_floor = 4500;
columns_spacing = 6.4;
structural_system = braced_frame_and_moment_resisting_frame;



solution is
        building = ['http://m-ram.mit.edu/act/'];


end;
```

Figure 32 - M-RAM CASL partial case definition block (2 of 54 cases)

4.5.4 Retrieving Cases

The case retrieval is carried out in three stages:

1. Pre-processing rules may be applied to find a set of appropriate weights for performing case matching, or for expanding the definition of user case.

2. The index values are used to perform an index search. This retrieves a subset of cases from the case-base, which match all the index values exactly.

3. Once the list of cases has been retrieved, the user can allow the program to automatically select a case, based on weight matching. For each case in the subset, the case matcher finds a weight that is obtained by totaling the weights of all fields that matched. Fields which do not match exactly but are defined to be similar by the modification rules return a value which is less than the field's normal weight. The case matcher selects the case with the highest total weight.

4.5.5 Weight Matching Algorithm

Caspian's weight matching mechanism is a simple form of "nearest-neighbor" search. There are a number of rules that are used to calculate how close a field value is to the corresponding value in the user case. This results in a closeness value for each field, and the total of these values is the match-weight of the case. The case with the highest match-weight is deemed to be the best match, and is the one retrieved from the case-base.

The rules are as follows:

1. If two fields match exactly the full weight of the field is returned.

2. If two fields are similar, ¾ of the weight is returned

3. If two numbers are similar (i.e. both falls within the same range as defined in the modification section) then a value is calculated which reflects how close they are in proportion to the range, and this is multiplied by the weight of the field.

4. Strings must match exactly

5. When matching lists, the following rules apply:

- When matching values at the top level, order is not important, but when matching lists within lists the order of elements is important. So for example when matching two simple lists of symbols, for each symbol in the list from the case being examined, Caspian scans through the elements of the user case until it finds a symbol which is the same or similar. If it finds one, this increases the returned weight. The match-weight of the list is calculated by totaling the match-weight of each element divided by the length of the list from the user case.
- It is possible for sub-lists to be similar. The full weight of a sub-list is one.
- If an element in a list is symbol which is also the name of a field, and the field contains a string which matches a string/symbol in the user case, then the two elements are defined to be similar.
- Symbol elements may be similar. Similar symbols have a match-weight of ¾.

4.6 The GA Agent

In the M-RAM implementation the partially matching retrieved design cases provided by the CBR agent are used as the initial population (generation 0) for the GA. The retrieved design cases that are represented using attribute-value formalism describe a specific design solution and correspond to the building's genotype. Here is important to consider that unlike in the traditional implementation of GA where the initial (seed) population have a few extraordinary individuals in a population of mediocre colleagues the M-RAM model deals with above average initial population provided by the CBR agent. In this GA implementation what is needed is just a fine-tuning.

GAlib, a C++ genetic Algorithm library developed at MIT and available for non-profit purposes, was used to implement M-RAM's GA agent [Wall, 1996].

4.6.1 The GAlib Library

GAlib is a C++ library of genetic algorithm objects. The library includes tools for using genetic algorithms to do optimization in any C++ program using any representation and any genetic operators.

The GAlib library is centered around two main classes: a genome and a genetic algorithm. Each genome instance represents a single solution to the problem. The genetic algorithm object defines how the evolution should take place. The genetic algorithm uses an objective function to determine how "fit" each genome is for survival. It uses genome operators (built into the genome) and selection/replacement strategies (built into the genetic algorithm) to generate new individuals.

The genetic algorithm object determines which individuals should survive, which should reproduce, and which should die. It also records statistics and decides how long the evolution should continue (Figure 33).

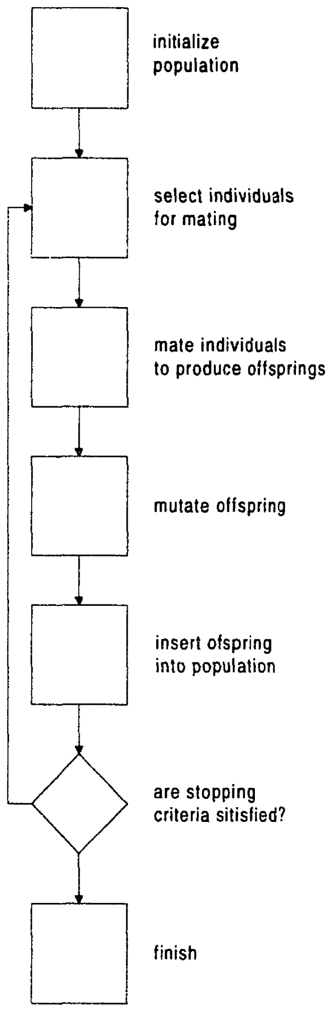


Figure 33 – GAlib GA [adapted from Wall, 1996]

The library contains four flavors of genetic algorithms. The first is the standard simple genetic algorithm, as described by Goldberg [1989]. This algorithm uses non-overlapping populations and optional elitism. The second is a steady state genetic algorithm that uses overlapping populations. In this variation, the user can specify how much of the population should be replaced in each generation. The third variation is the incremental genetic algorithm in which each generation consists of only one or two children. The fourth type is the deme genetic algorithm. This algorithm evolves multiple populations in parallel using a steady state algorithm. In each generation the algorithm migrates some of the individuals from each population to one of the other populations. M-RAM's GA agent uses the steady state algorithm because it had a better performance to solve a problem that just required fine-tuning. Unlike the simple genetic algorithm that generates an entirely new population of individuals in each generation, the steady state algorithm allows the replacement of some genes while keeping others intact.

The genetic algorithm contains the statistics, replacement strategy, and parameters for running the algorithm. The population object, a container for genomes, also contains some statistics as well as selection and scaling operations. A typical genetic algorithm will run forever. The library has built in functions for specifying when the algorithm should terminate. These include terminate-upon-generation, in which the user specifies a certain number of generations for which the algorithm should run, and terminate-upon-convergence, in which the user specifies a value to which the best-of-generation score should converge.

There are three requirement needed to solve a problem using GAlib:

1. Definition of a representation

2. Definition of the genetic operators

3. Definition the objective function

4.6.1.1 Defining a Representation

When using GAlib to solve an optimization problem there is the need to represent a single solution to the problem in a single data structure. In GAlib, this simple data structure is called GAGenome. The library contains four types of genomes: GAListGenome, GATreeGenome, GAArrayGenome, and GABinaryStringGenome. These classes are derived from the base GAGenome class and a data structure class as indicated by their names.

Defining an appropriate representation is part of the art of using genetic algorithms. There is the need to use a representation that is minimal (avoiding to increase the search space damaging the GA performance) but completely expressive. The representation should be able to represent any solution to the problem, but, if possible, should not represent infeasible solutions.

4.6.1.2 The Genetics Operators

Each genome has three primary operators: initialization, mutation, and crossover.

The initialization operator determines how the genome is initialized. It is called when the user initializes a population on the genetic algorithm. In the M-RAM implementation the genetic material for this initialization operator is obtained from the output of M-RAM's CBR agent.

The mutation operator defines the procedure for mutating each genome. Mutation means different things for different data types. For example, a typical mutation for a binary string genome flips the bits in the string with a given probability. A typical mutation for a tree, on the other hand, swaps subtrees with a given probability.

The crossover operator defines the procedure for generating a child from two parent genomes. Like the mutation operator, crossover is specific to the data type. Unlike mutation, however, crossover involves multiple genomes. In GAlib, each genome "knows" its preferred method of mating.

4.6.1.3 Objective Functions and Fitness Scaling

The genetic algorithm needs only a single measure of how good a single individual is compared to the others individuals. The objective function provides this measure. It is important to note the distinction between fitness and objective scores. The objective score is the value returned by the objective function, it is the raw performance evaluation of the genome. The fitness score, on the other hand, is a possible-transformed rating used by the genetic algorithm to determine the fitness of individuals for mating. The fitness score is typically obtained by a linear scaling of the raw objective scores. The genetic algorithm uses the fitness scores, not the objective scores, to do selection.

4.6.2 M-RAM's GA Agent GAlib Implementation

The objective of the M-RAM GA Agent is to combine bits and pieces from several past experiences retrieved from the CBR Agent in order to solve a new problem.

The attribute-value representation of cases retrieved by M-RAM's CBR Agent are reinterpreted and used in the context of the genetic algorithm by considering the attributes used in the description of a case to be equivalent to genes. Thus, the collection of attributes used to describe a specific building corresponds to the building's genome.

The values that are associated to each attribute in the description of a case make up the case description and can be defined as the case phenotype.

4.6.2.1 M-RAM's GA Agent Genome Representation

The structural conceptual design can be defined as a continuous variable engineering design domain where a design is represented by a number of continuous design parameters. The GARealGenome (Real Number Genome) class from the GAlib library was then chosen to develop the representation scheme for the genotypes. The GARealGenome is meant to be used for applications whose representation requires array of (possible bounded) real numbers parameters. The elements of the array can assume bounded values, discrete bounded values, or enumerated values, depending on the type of allele set that is used to create the genome. The allele set defines the possible values that each element in the genome may assume.

Figure 34 shows the genome of the ACT Tower in Hamamatsu City, Japan, one of the buildings in M-RAM's database.

Gene	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Value	211.9	47	2	5	1	5	30	1000	4.52	4.73	52	1.5	0.06	3.58	17.5	850	6.4	4500	6.4

Where:

Gene	Attribute
0	Height from the street to the roof
1	Number of stories
2	Number of levels below ground
3	Building use: 1=office; 2=hotel; 3=residential; 4=garage; 5=mixed
4	Frame material: 1=steel; 2=concrete; 3=mixed
5	Typical floor life load
6	Basic wind velocity
7	Maximum lateral deflection
8	Fundamental-period transverse
9	Fundamental-period longitudinal
10	Design acceleration
11	Design damping
12	Earthquake loading
13	Typical floor story height
14	Typical floor beam span
15	Typical floor beam depth
16	Typical floor beam spacing
17	columns size at ground floor
18	Columns spacing

Figure 34 – A building genome representation

4.6.2.2 M-RAM's GA Agent Genetic Operators

The type of genetic algorithm applied by M-RAM's GA Agent is the steady state algorithm implemented by GAlib's GASteadyStateGA Class. This genetic algorithm uses overlapping populations with a user-specified amount of overlap. The algorithm creates a population of individuals by cloning the population passed by M-RAM's CBR Agent. In each generation the algorithm creates a temporary population of individuals, adds these to previous population, then removes the worst individuals in order to return the population to its original size.

GAlib's GASteadyStateGA Class has an abstract superclass called GAGeneticAlgorithm. This base genetic algorithm class keeps track of evolution statistics such as number of mutations, number of crossovers, number of evaluations, best/mean/worst in each generation, and initial/final population statistics. It also defines the terminator, a member function that specifies the stopping criterion for the algorithm.

M-RAM GA Agent runs the steady state algorithm for 1,500 generations using GAGeneticAlgorithm default values of mutation and crossover.

4.6.2.3 M-RAM GA Agent Objective Function

The objective function is an adaptation of the one presented by Maher and da Silva Garza [1996]. Two types of considerations are used to measure a solution's fitness in M-RAM's GA Agent: the problem requirements and general structural design constraints (represented by attribute correlation). Both types of fitness are represented as constraints and the objective is measured by the number of constraints not violated. The solution is feasible if none of the constraints are violated, therefore the goal of the GA Agent is to maximize constraints non violations (Figure 35, 36 and 37).

Two constraint examples are:

1. **Problem requirement constraint:** If 300 meters is the required height of the building being designed the height is considered then to be one of the design constraints. To evaluate a genome proposed by the genetic algorithm the objective function adds 10 points to the objective value of all genomes that have height in the interval (285, 315m), 5 points to genomes that has height in the intervals (240, 285m) and (305, 360m) and nothing to all others genomes.

2. **General structural design constraint:** To retrieve feasible design solutions the objective function give points to solutions that keep the proportionality among number of floors, story height and building height. The objective function gives 10 points to all genomes that respects the following equation:

0.95*Story Height < (Height of the Building/Number of Floors) < 1.15* Story Height

```
// This objective tries to maximize each element in the genome.

float
Objective(GAGenome& g)
{

  GARealGenome& genome = (GARealGenome&)g;
  float value=0.0;
  if ((0.95< (height/genome.gene(0))) && ((height/genome.gene(0)) < 1.05))
    value = value+10;
  if((0.8< (height/genome.gene(0))) && ((height/genome.gene(0)) < 0.95))
    value = value+5;
  if((1.05< (height/genome.gene(0))) && ((height/genome.gene(0)) < 1.2))
    value = value+5;
  if ((0.95< (stories/genome.gene(1))) && ((stories/genome.gene(1)) < 1.05))
    value = value+10;
  if((0.8< (stories/genome.gene(1))) && ((stories/genome.gene(1)) < 0.95))
    value = value+5;
  if((1.05< (stories/genome.gene(1))) && ((stories/genome.gene(1)) < 1.2))
    value = value+5;
  if ((0.95< (underground/genome.gene(2))) && ((underground/genome.gene(2))
< 1.05))
    value = value+10;
  if((0.8< (underground/genome.gene(2))) && ((underground/genome.gene(2)) <
0.95))
    value = value+5;
  if((1.05< (underground/genome.gene(2))) && ((underground/genome.gene(2)) <
1.2))
    value = value+5;
  if ((0.95< (use/genome.gene(3))) && ((use/genome.gene(3)) < 1.05))
    value = value+10;
  if((0.8< (use/genome.gene(3))) && ((use/genome.gene(3)) < 0.95))
    value = value+5;
  if((1.05< (use/genome.gene(3))) && ((use/genome.gene(3)) < 1.2))
    value = value+5;
  if ((0.95< (material/genome.gene(4))) && ((material/genome.gene(4)) <
1.05))
    value = value+10;
  if((0.8< (material/genome.gene(4))) && ((material/genome.gene(4)) < 0.95))
    value = value+5;
  if((1.05< (material/genome.gene(4))) && ((material/genome.gene(4)) < 1.2))
    value = value+5;
  if ((0.95< (lifeload/genome.gene(5))) && ((lifeload/genome.gene(5)) <
1.05))
    value = value+10;
  if((0.8< (lifeload/genome.gene(5))) && ((lifeload/genome.gene(5)) < 0.95))
    value = value+5;
  if((1.05< (lifeload/genome.gene(5))) && ((lifeload/genome.gene(5)) < 1.2))
    value = value+5;
  if ((0.95< (windvelocity/genome.gene(6))) &&
((windvelocity/genome.gene(6)) < 1.05))
    value = value+10;
  if((0.8< (windvelocity/genome.gene(6))) && ((windvelocity/genome.gene(6))
< 0.95))
    value = value+5;
  if((1.05< (windvelocity/genome.gene(6))) && ((windvelocity/genome.gene(6))
< 1.2))
  value = value+5;
```

Figure 35 – M-RAM's GA agent objective function – part I

```
if ((0.95< (latdeflection/genome.gene(7))) && ((latdeflection/genome.gene(7)) <
1.05))
    value = value+10;
if((0.8< (latdeflection/genome.gene(7))) && ((latdeflection/genome.gene(7)) <
0.95))
    value = value+5;
if((1.05< (latdeflection/genome.gene(7))) && ((latdeflection/genome.gene(7)) <
1.2))
    value = value+5;
if ((0.95< (periodtrans/genome.gene(8))) && ((periodtrans/genome.gene(8)) < 1.05))
    value = value+10;
if((0.8< (periodtrans/genome.gene(8))) && ((periodtrans/genome.gene(8)) < 0.95))
    value = value+5;
if((1.05< (periodtrans/genome.gene(8))) && ((periodtrans/genome.gene(8)) < 1.2))
    value = value+5;
if ((0.95< (periodlong/genome.gene(9))) && ((periodlong/genome.gene(9)) < 1.05))
    value = value+10;
if((0.8< (periodlong/genome.gene(9))) && ((periodlong/genome.gene(9)) < 0.95))
    value = value+5;
if((1.05< (periodlong/genome.gene(9))) && ((periodlong/genome.gene(9)) < 1.2))
    value = value+5;
if ((0.95< (acceleration/genome.gene(10))) && ((acceleration/genome.gene(10)) <
1.05))
    value = value+10;
if((0.8< (acceleration/genome.gene(10))) && ((acceleration/genome.gene(10)) <
0.95))
    value = value+5;
if((1.05< (acceleration/genome.gene(10))) && ((acceleration/genome.gene(10)) <
1.2))
    value = value+5;
if ((0.95< (dumping/genome.gene(11))) && ((dumping/genome.gene(11)) < 1.05))
    value = value+10;
if((0.8< (dumping/genome.gene(11))) && ((dumping/genome.gene(11)) < 0.95))
    value = value+5;
if((1.05< (dumping/genome.gene(11))) && ((dumping/genome.gene(11)) < 1.2))
    value = value+5;
if ((0.95< (earthloading/genome.gene(12))) && ((earthloading/genome.gene(12)) <
1.05))
    value = value+10;
if((0.8< (earthloading/genome.gene(12))) && ((earthloading/genome.gene(12)) <
0.95))
    value = value+5;
if((1.05< (earthloading/genome.gene(12))) && ((earthloading/genome.gene(12)) <
1.2))
    value = value+5;
```

Figure 36 - M-RAM's GA agent objective function – part II

```
if ((0.95< (storyheight/genome.gene(13))) && ((storyheight/genome.gene(13)) <
1.05))
    value = value+10;
if((0.8< (storyheight/genome.gene(13))) && ((storyheight/genome.gene(13)) < 0.95))
    value = value+5;
  if((1.05< (storyheight/genome.gene(13))) && ((storyheight/genome.gene(13)) <
1.2))
    value = value+5;
if ((0.95< (beamspan/genome.gene(14))) && ((beamspan/genome.gene(14)) < 1.05))
    value = value+10;
if((0.8< (beamspan/genome.gene(14))) && ((beamspan/genome.gene(14)) < 0.95))
    value = value+5;
if((1.05< (beamspan/genome.gene(14))) && ((beamspan/genome.gene(14)) < 1.2))
    value = value+5;
if ((0.95< (beamdepth/genome.gene(15))) && ((beamdepth/genome.gene(15)) < 1.05))
    value = value+10;
if((0.8< (beamdepth/genome.gene(15))) && ((beamdepth/genome.gene(15)) < 0.95))
    value = value+5;
if((1.05< (beamdepth/genome.gene(15))) && ((beamdepth/genome.gene(15)) < 1.2))
    value = value+5;
if ((0.95< (beamspacing/genome.gene(16))) && ((beamspacing/genome.gene(16)) <
1.05))
    value = value+10;
if((0.8< (beamspacing/genome.gene(16))) && ((beamspacing/genome.gene(16)) < 0.95))
    value = value+5;
if((1.05< (beamspacing/genome.gene(16))) && ((beamspacing/genome.gene(16)) < 1.2))
    value = value+5;
if ((0.95< (columnssize/genome.gene(17))) && ((columnssize/genome.gene(17)) <
1.05))
    value = value+10;
if((0.8< (columnssize/genome.gene(17))) && ((columnssize/genome.gene(17)) < 0.95))
    value = value+5;
if((1.05< (columnssize/genome.gene(17))) && ((columnssize/genome.gene(17)) < 1.2))
    value = value+5;
if ((0.95< (columnsspacing/genome.gene(18))) && ((columnsspacing/genome.gene(18)) <
1.05))
    value = value+10;
if((0.8< (columnsspacing/genome.gene(18))) && ((columnsspacing/genome.gene(18)) <
0.95))
    value = value+5;
if((1.05< (columnsspacing/genome.gene(18))) && ((columnsspacing/genome.gene(18)) <
1.2))
    value = value+5;
if((0.85 < ((genome.gene(0)/genome.gene(1))/genome.gene(13)) &&
((genome.gene(0)/genome.gene(1))/genome.gene(13)) < 1.15))
    value=value+10;
if((0.75 < ((genome.gene(0)/0.5)/genome.gene(7)) &&
((genome.gene(0)/0.5)/genome.gene(7)) < 1.25))
     value=value+10;
  return value;
}
```

Figure 37 - M-RAM's GA agent objective function – part III

4.7 Chapter Conclusions

In this chapter, M-RAM's components were presented together with the architecture that support those components to collaborate solving a conceptual structural design problem.

It was demonstrated how the WWW interface receives the expected data as input, how it launches the manager program that decides which server will perform which operation on the data. Was demonstrated too how each agent work and how they solve each part of the problem. Finally was presented how the manager coordinates the partial solutions assuring a consistent solution for the problem as a whole and how the manager program send the solution or possible solutions of the structural design problem back to a WWW interface to displays it to the user.

Chapter 5

5 Illustrative Example

5.1 Introduction

Section 1.1 described M-RAM as a model that aim to assist engineers in the initial stage of tall buildings structural design. During this initial stage, defined as conceptual design phase, the designer makes an overall guess of a possible solution for the design problem based on information provided in the problem definition. After an analysis step the designer decides if this initial guess is feasible or not. If the proposed solution is feasible he/she can move to the next design phase but if in the other hand, the proposed solution is not feasible he/she needs to develop another initial guess reinitiating the design process once again. This chapter presents a tall building structural design example demonstrating how M-RAM would help a designer to obtain a good initial guess improving his/her productivity by reducing the number of these design-analyses cycles.

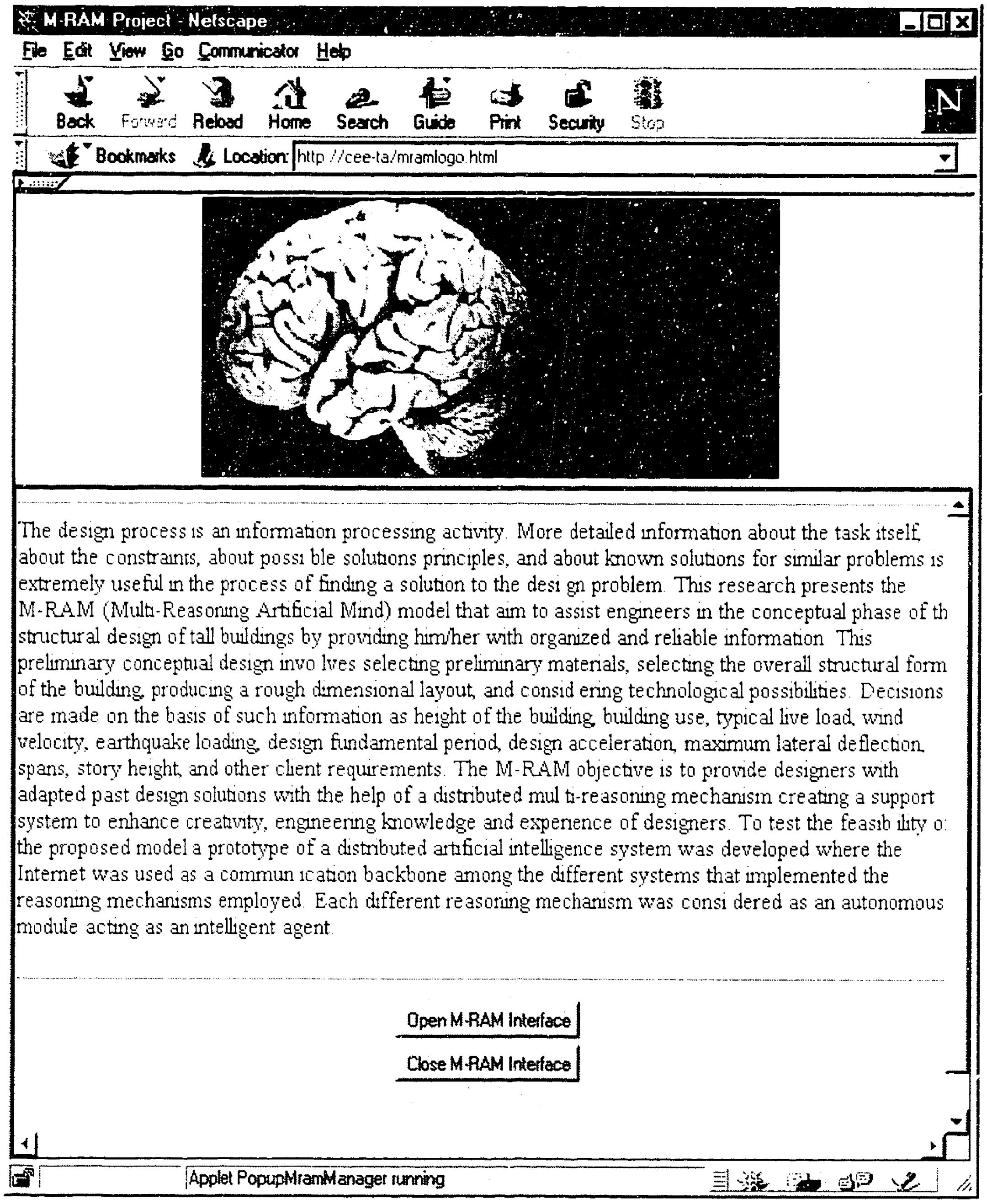
5.2 Building Design Example

M-RAM provides designers with adapted design solutions first by classifying the design problem with the objective of reducing the search space, then by retrieving past experience and finally by adapting the retrieved cases. This is done with the help of a distributed multi-reasoning mechanism as described in Chapter 4.

Lets consider, for example, the design of an office building in Boston, Massachusetts. The specifications of the new design are for 85 floors but, because the building is going to be built in Boston's financial area that is located near Logan Airport, the architect wants to keep the building total height as low as possible. To do so he defined the total height to be around 280 meters. The specifications also require the structural material to be steel because of New England's cold winter and labor characteristics making difficult and expensive the execution of massive concrete structures. Finally, because of space requirements the architect is also specifying 12 meters of space between columns. The attribute-value pairs representing the design requirements are the following:

▪ **Height from the street to the roof:**	**280 meters**
▪ **Number of stories:**	**85**
▪ **Building Use:**	**Office**
▪ **Structural Material:**	**Steel**
▪ **Columns Spacing:**	**12.00 m**

To be able to run M-RAM with these information as requirements the designer first needs to open the M-RAM project home page in the URL: http://cee-ta.mit.edu. In this home page as shown in figure 38 there is the open M-RAM interface button. Clicking in this button causes the M-RAM interface (Figure 39) to pop in the computer screen.


M-RAM Project - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://cee-ta/mramlogo.html
Open M-RAM Interface
Close M-RAM Interface
Applet PopupMramManager running

The design process is an information processing activity. More detailed information about the task itself, about the constraints, about possi ble solutions principles, and about known solutions for similar problems is extremely useful in the process of finding a solution to the desi gn problem. This research presents the M-RAM (Multi-Reasoning Artificial Mind) model that aim to assist engineers in the conceptual phase of th structural design of tall buildings by providing him/her with organized and reliable information. This preliminary conceptual design invo lves selecting preliminary materials, selecting the overall structural form of the building, producing a rough dimensional layout, and consid ering technological possibilities. Decisions are made on the basis of such information as height of the building, building use, typical live load, wind velocity, earthquake loading, design fundamental period, design acceleration, maximum lateral deflection, spans, story height, and other client requirements. The M-RAM objective is to provide designers with adapted past design solutions with the help of a distributed mul ti-reasoning mechanism creating a support system to enhance creativity, engineering knowledge and experience of designers. To test the feasib ility o the proposed model a prototype of a distributed artificial intelligence system was developed where the Internet was used as a commun ication backbone among the different systems that implemented the reasoning mechanisms employed. Each different reasoning mechanism was consi dered as an autonomous module acting as an intelligent agent.

Figure 38 – M-RAM project home page

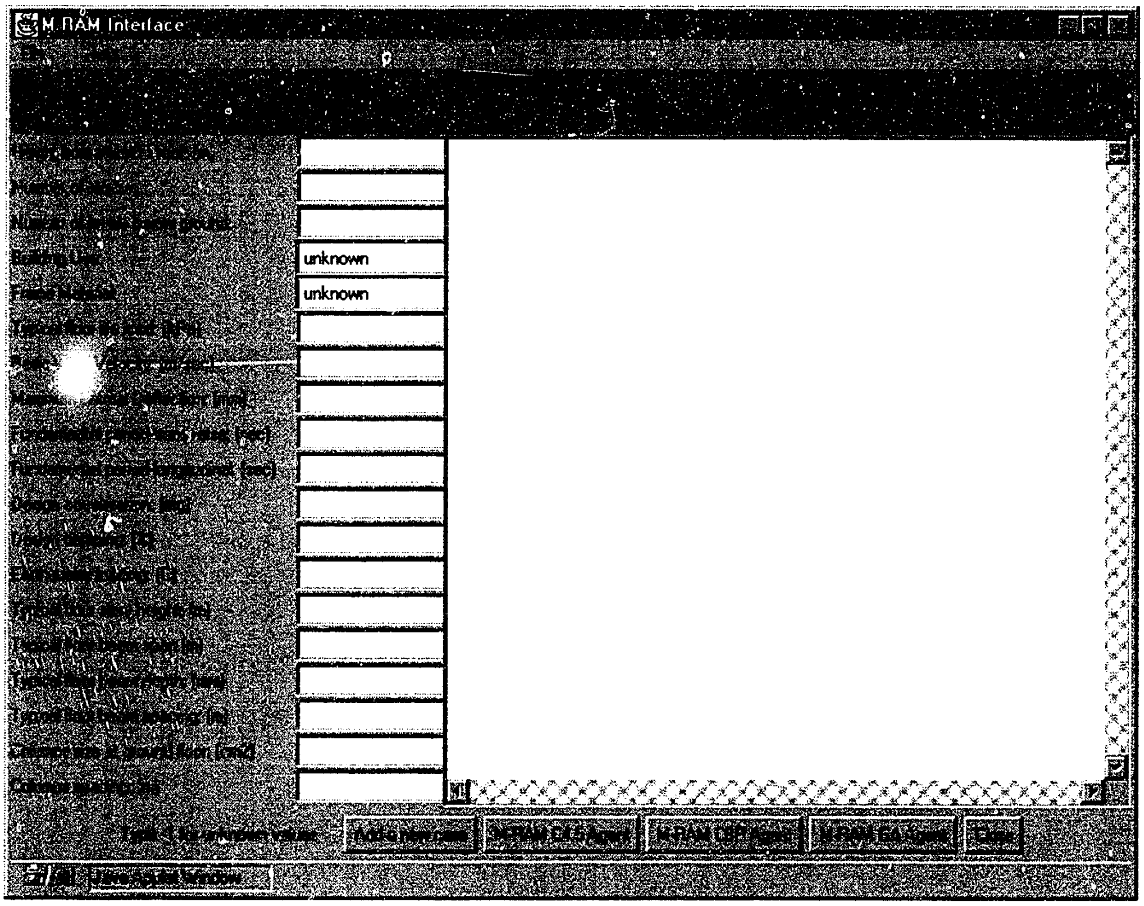
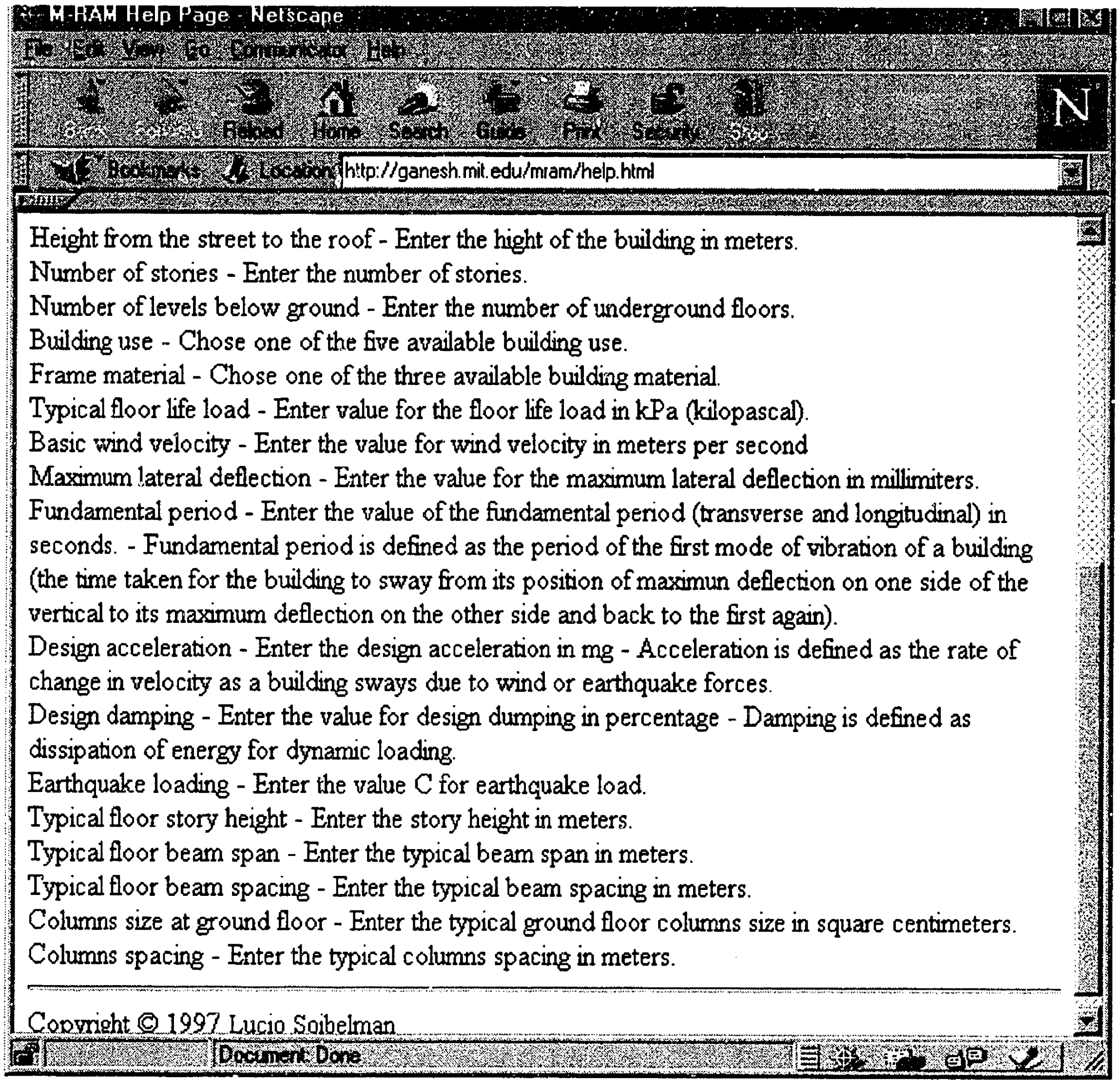


Figure 39 – Initial M-RAM interface

This easy to use interface allows the designer to interact with the three available agents. If the designer has any doubt about the input data he/she can use the available interface help. Clicking the interface help menu automatically opens a browser with information about M-RAM attributes (Figure 40).

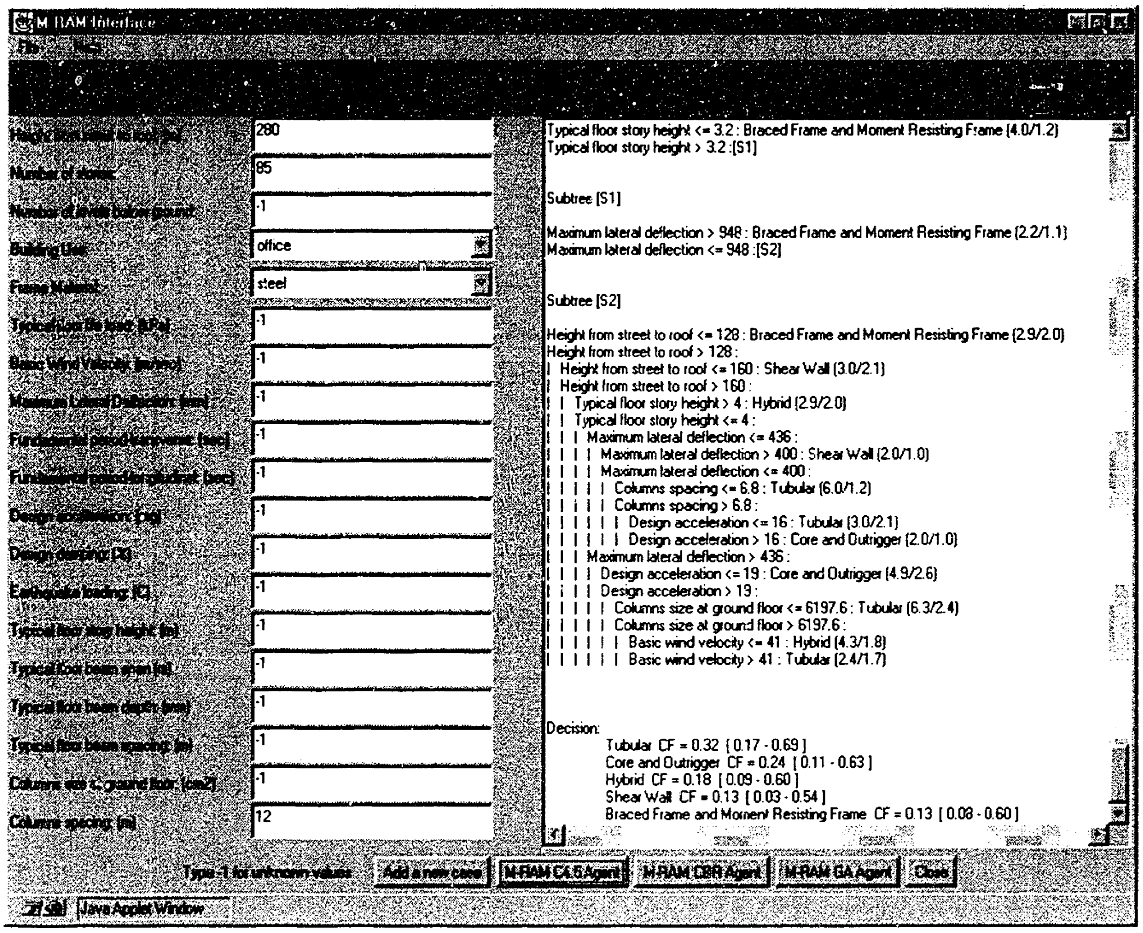

M-RAM Help Page - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://ganesh.mit.edu/mram/help.html
Document: Done

Height from the street to the roof - Enter the hight of the building in meters.
Number of stories - Enter the number of stories.
Number of levels below ground - Enter the number of underground floors.
Building use - Chose one of the five available building use.
Frame material - Chose one of the three available building material.
Typical floor life load - Enter value for the floor life load in kPa (kilopascal).
Basic wind velocity - Enter the value for wind velocity in meters per second
Maximum lateral deflection - Enter the value for the maximum lateral deflection in millimiters.
Fundamental period - Enter the value of the fundamental period (transverse and longitudinal) in seconds. - Fundamental period is defined as the period of the first mode of vibration of a building (the time taken for the building to sway from its position of maximun deflection on one side of the vertical to its maximum deflection on the other side and back to the first again).
Design acceleration - Enter the design acceleration in mg - Acceleration is defined as the rate of change in velocity as a building sways due to wind or earthquake forces.
Design damping - Enter the value for design dumping in percentage - Damping is defined as dissipation of energy for dynamic loading.
Earthquake loading - Enter the value C for earthquake load.
Typical floor story height - Enter the story height in meters.
Typical floor beam span - Enter the typical beam span in meters.
Typical floor beam spacing - Enter the typical beam spacing in meters.
Columns size at ground floor - Enter the typical ground floor columns size in square centimeters.
Columns spacing - Enter the typical columns spacing in meters.

Copyright © 1997 Lucio Soibelman

Figure 40 – M-RAM help page

To run the classification agent the designer needs to click in the M-RAM C4.5 agent button after typing in the M-RAM interface the values for the design problem requirements (typing –1 for unknown values). When the designer clicks the M-RAM C4.5 Agent button the attribute values are sent to the server where the C4.5 engine is located. The C4.5 engine classifies the design problem sending the classification solution back to M-RAM manager that displays it in the text area of the M-RAM interface as shown in figure 41.



[illegible]	280
Number of stories:	85
Number of levels below ground:	-1
Building Use:	office
Frame Material:	steel
Typical floor live load: [kPa]	-1
Basic Wind Velocity: [m/sec]	-1
Maximum Lateral Deflection: [mm]	-1
Fundamental period [illegible]: [sec]	-1
Fundamental period [illegible]: [sec]	-1
Design acceleration: [mg]	-1
Design damping: [%]	-1
Earthquake loading: [C]	-1
Typical floor story height: [m]	-1
Typical floor beam span: [m]	-1
Typical floor beam depth: [mm]	-1
Typical floor beam spacing: [m]	-1
Columns size @ ground floor: [cm2]	-1
Columns spacing: [m]	12

```
Typical floor story height <= 3.2 : Braced Frame and Moment Resisting Frame (4.0/1.2)
Typical floor story height > 3.2 :[S1]


Subtree [S1]

Maximum lateral deflection > 948 : Braced Frame and Moment Resisting Frame (2.2/1.1)
Maximum lateral deflection <= 948 :[S2]


Subtree [S2]

Height from street to roof <= 128 : Braced Frame and Moment Resisting Frame (2.9/2.0)
Height from street to roof > 128 :
|   Height from street to roof <= 160 : Shear Wall (3.0/2.1)
|   Height from street to roof > 160 :
|   |   Typical floor story height > 4 : Hybrid (2.9/2.0)
|   |   Typical floor story height <= 4 :
|   |   |   Maximum lateral deflection <= 436 :
|   |   |   |   Maximum lateral deflection > 400 : Shear Wall (2.0/1.0)
|   |   |   |   Maximum lateral deflection <= 400 :
|   |   |   |   |   Columns spacing <= 6.8 : Tubular (6.0/1.2)
|   |   |   |   |   Columns spacing > 6.8 :
|   |   |   |   |   |   Design acceleration <= 16 : Tubular (3.0/2.1)
|   |   |   |   |   |   Design acceleration > 16 : Core and Outrigger (2.0/1.0)
|   |   |   Maximum lateral deflection > 436 :
|   |   |   |   Design acceleration <= 19 : Core and Outrigger (4.9/2.6)
|   |   |   |   Design acceleration > 19 :
|   |   |   |   |   Columns size at ground floor <= 6197.6 : Tubular (6.3/2.4)
|   |   |   |   |   Columns size at ground floor > 6197.6 :
|   |   |   |   |   |   Basic wind velocity <= 41 : Hybrid (4.3/1.8)
|   |   |   |   |   |   Basic wind velocity > 41 : Tubular (2.4/1.7)



Decision:
        Tubular  CF = 0.32  [ 0.17 - 0.69 ]
        Core and Outrigger  CF = 0.24  [ 0.11 - 0.63 ]
        Hybrid  CF = 0.18  [ 0.09 - 0.60 ]
        Shear Wall  CF = 0.13  [ 0.03 - 0.54 ]
        Braced Frame and Moment Resisting Frame  CF = 0.13  [ 0.08 - 0.60 ]
```

Figure 41 – M-RAM interface with C4.5 agent output

The complete output from the Classification agent that is printed in the text area of the M-RAM interface can be seen in figures 42 and 43.

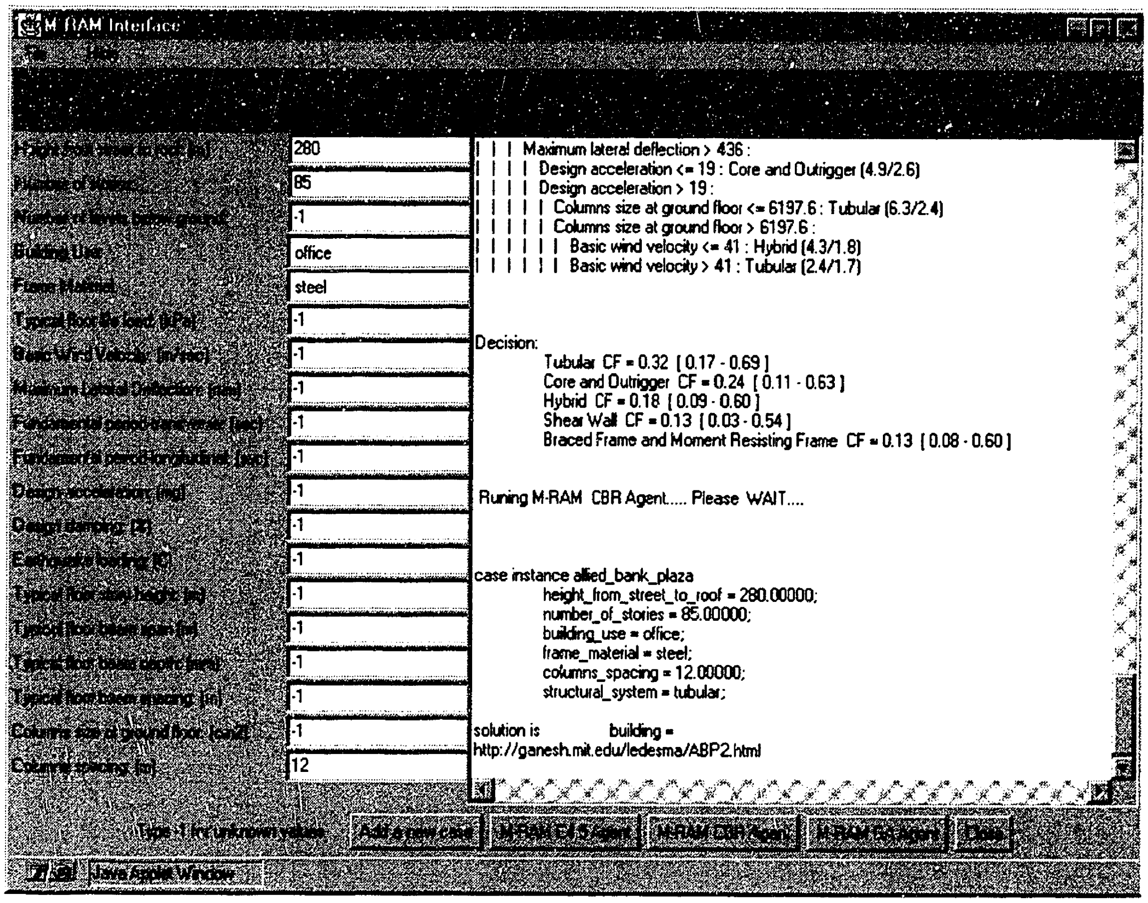
```
Runing M-RAM Classification Agent..... Please WAIT....


C4.5 [release 5] M-RAM Classifier - Lucio Soibelman          Sun Apr  5 23:11:52 1998
-------------------------------------------------

Height from street to roof = 280.000000
Number of stories = 85.000000
Number of levels below ground = -1.000000
Building Use = office
Frame material = steel
Typical floor live load = -1.000000
Basic wind velocity = -1.000000
Maximum lateral deflection = -1.000000
Design fundamental period-transverse = -1.000000
Design fundamental period-longitudinal = -1.000000
Design acceleration = -1.000000
Design Dumping = -1.000000
Earthquake loading = -1.000000
Typical floor story height = -1.000000
Typical floor beam span = -1.000000
Typical floor beam depth = -1.000000
Typical floor beam spacing = -1.000000
Columns size at ground floor = -1.000000
Columns spacing = 12.000000Decision Tree:

Typical floor story height <= 3.2 : Braced Frame and Moment Resisting Frame (4.0/1.2)
Typical floor story height > 3.2 :[S1]


Subtree [S1]

Maximum lateral deflection > 948 : Braced Frame and Moment Resisting Frame (2.2/1.1)
Maximum lateral deflection <= 948 :[S2]
```

Figure 42 – M-RAM classification agent output part I

```
Subtree [S2]

Height from street to roof <= 128 : Braced Frame and Moment Resisting Frame (2.9/2.0)
Height from street to roof > 128 :
|  Height from street to roof <= 160 : Shear Wall (3.0/2.1)
|  Height from street to roof > 160 :
|  |  Typical floor story height > 4 : Hybrid (2.9/2.0)
|  |  Typical floor story height <= 4 :
|  |  |  Maximum lateral deflection <= 436 :
|  |  |  |  Maximum lateral deflection > 400 : Shear Wall (2.0/1.0)
|  |  |  |  Maximum lateral deflection <= 400 :
|  |  |  |  |  Columns spacing <= 6.8 : Tubular (6.0/1.2)
|  |  |  |  |  Columns spacing > 6.8 :
|  |  |  |  |  |  Design acceleration <= 16 : Tubular (3.0/2.1)
|  |  |  |  |  |  Design acceleration > 16 : Core and Outrigger (2.0/1.0)
|  |  |  Maximum lateral deflection > 436 :
|  |  |  |  Design acceleration <= 19 : Core and Outrigger (4.9/2.6)
|  |  |  |  Design acceleration > 19 :
|  |  |  |  |  Columns size at ground floor <= 6197.6 : Tubular (6.3/2.4)
|  |  |  |  |  Columns size at ground floor > 6197.6 :
|  |  |  |  |  |  Basic wind velocity <= 41 : Hybrid (4.3/1.8)
|  |  |  |  |  |  Basic wind velocity > 41 : Tubular (2.4/1.7)


Decision:
        Tubular  CF = 0.32  [ 0.17 - 0.69 ]
        Core and Outrigger  CF = 0.24  [ 0.11 - 0.63 ]
        Hybrid  CF = 0.18  [ 0.09 - 0.60 ]
        Shear Wall  CF = 0.13  [ 0.03 - 0.54 ]
        Braced Frame and Moment Resisting Frame  CF = 0.13  [ 0.08 - 0.60 ]
```

Figure 43 – M-RAM classification agent output part II

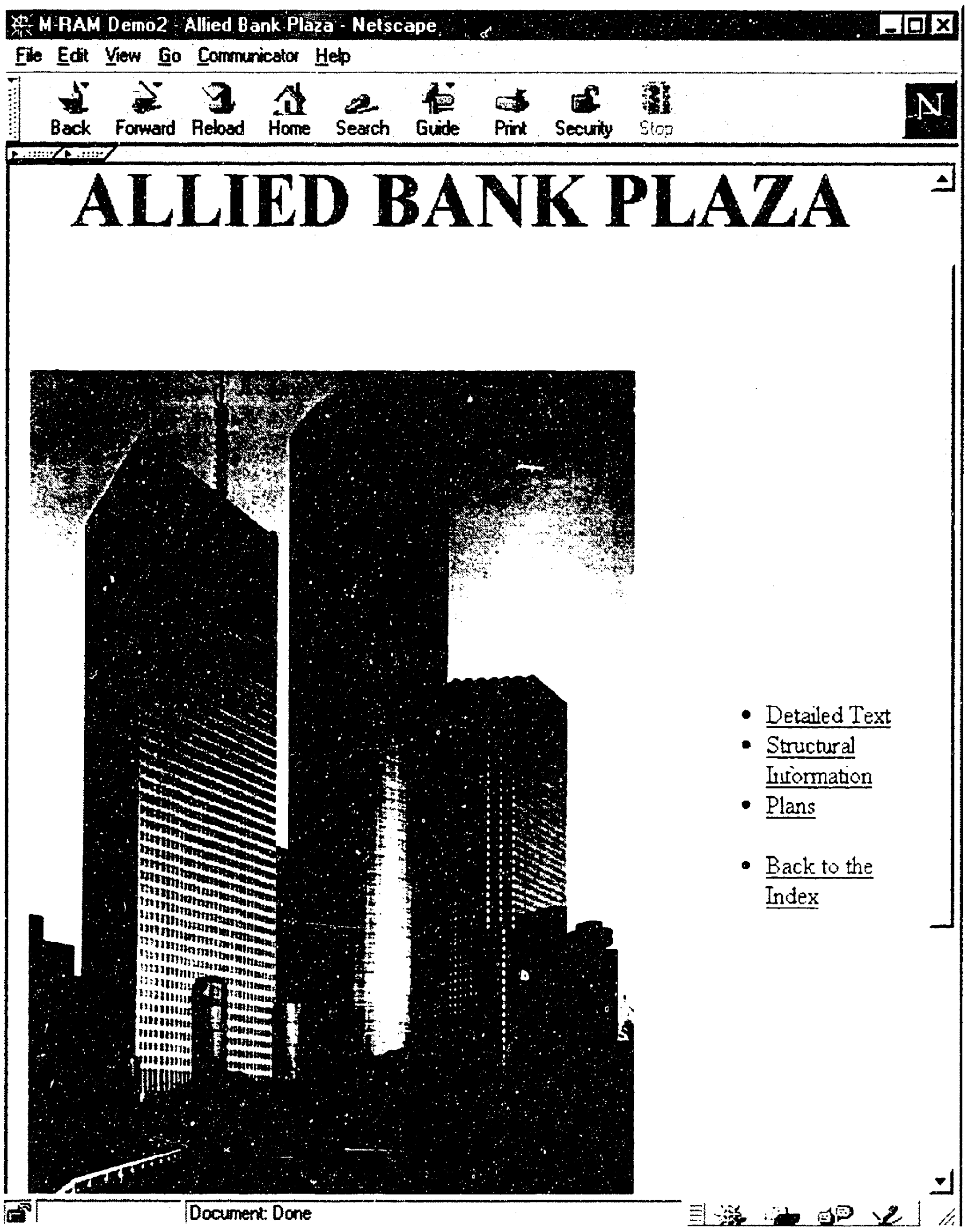
As can be seen in figures 42 and 43 the output of the classification engine can be divided in three parts. The first part prints the information inputted by the designer, the second part presents the actual decision tree and the third part presents the classification results. The example design was classified as a tubular system with a probability somewhere between 0.17 to 0.69, the best guess being 0.32 (certainty factor –CF- equal to 0.32). M-RAM is thus informing the designer that the majority of buildings with similar characteristics of the one being designed were designed with tubular structural systems to resist lateral loads. With this information M-RAM is ready to run its CBR agent to retrieve from its database of buildings design the one that is the best fit if compared with the example design. To do so the designer just need to press the M-RAM CBR Agent button. The manager then sends the design attributes added to the class defined by the C4.5 agent to the server were the CBR agent is located. The CBR agent retrieves from its memory the URL of the best matching case and send this information back to the manager that display it to the user in the M-RAM interface text area (Figure 44). As can be seen in figure 44 the CBR agent retrieved as a matching case for the design example a case instance called Allied Bank Plaza linking the design solution to the URL: http://ganesh.mit.edu/ledesma/ABP2.html. With this URL the M-RAM manager launches a browser window that pops up in the designer computer with the home page of the retrieved case (Figure 45). From this home page the user can go to a text description of the retrieved case (Figure 46), to a page with structural information of the case in tabular format (Figure 47), to a page with graphical structural details like the one shown in figure 48 and to a page with an index of all cases of the same class as the one being designed (Figure 49). From this index the designer is able to browse among available cases obtaining more information if compared to the one provided by the best matching case.



Height from street to roof [m]	280
Number of stories	85
Number of levels below ground	-1
Building Use	office
Frame Material	steel
Typical floor live load [kPa]	-1
Basic Wind Velocity [m/sec]	-1
Maximum Lateral Deflection [mm]	-1
Fundamental period-transversal [sec]	-1
Fundamental period-longitudinal [sec]	-1
Design acceleration [mg]	-1
Design damping [%]	-1
Earthquake loading [G]	-1
Typical floor story height [m]	-1
Typical floor beam span [m]	-1
Typical floor beam depth [mm]	-1
Typical floor beam spacing [m]	-1
Columns size at ground floor [cm2]	-1
Column spacing [m]	12

```
| | | Maximum lateral deflection > 436 :
| | | | Design acceleration <= 19 : Core and Outrigger (4.9/2.6)
| | | | Design acceleration > 19 :
| | | | | Columns size at ground floor <= 6197.6 : Tubular (6.3/2.4)
| | | | | Columns size at ground floor > 6197.6 :
| | | | | | Basic wind velocity <= 41 : Hybrid (4.3/1.8)
| | | | | | Basic wind velocity > 41 : Tubular (2.4/1.7)



Decision:
          Tubular CF = 0.32 [ 0.17 - 0.69 ]
          Core and Outrigger CF = 0.24 [ 0.11 - 0.63 ]
          Hybrid CF = 0.18 [ 0.09 - 0.60 ]
          Shear Wall CF = 0.13 [ 0.03 - 0.54 ]
          Braced Frame and Moment Resisting Frame CF = 0.13 [ 0.08 - 0.60 ]


 Runing M-RAM CBR Agent..... Please WAIT....



case instance allied_bank_plaza
          height_from_street_to_roof = 280.00000;
          number_of_stories = 85.00000;
          building_use = office;
          frame_material = steel;
          columns_spacing = 12.00000;
          structural_system = tubular;

solution is            building =
http://ganesh.mit.edu/ledesma/ABP2.html
```

Figure 44 – M-RAM interface with CBR agent output

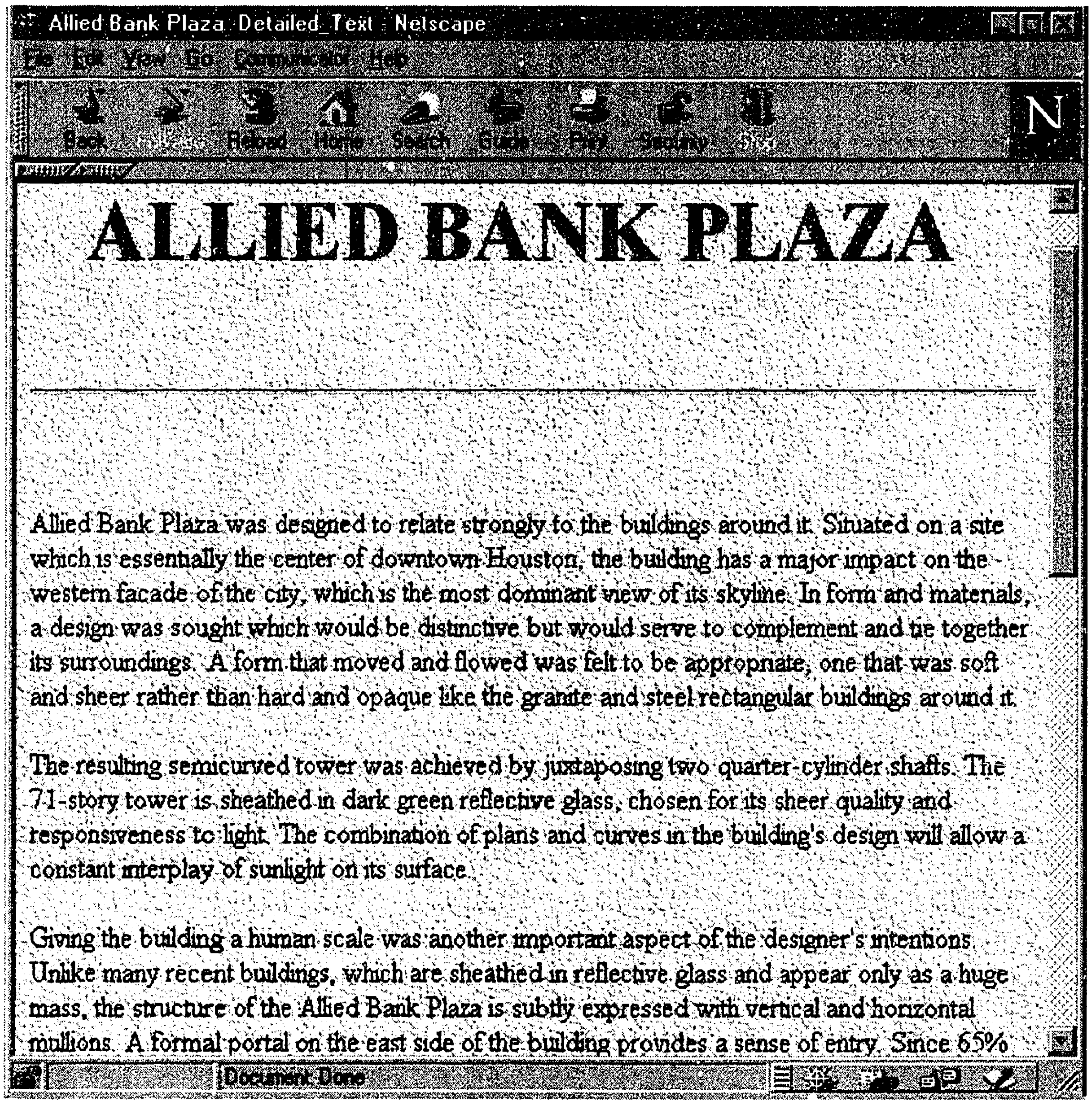

M-RAM Demo2 - Allied Bank Plaza - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Document: Done

ALLIED BANK PLAZA

- Detailed Text
- Structural Information
- Plans

- Back to the Index

Figure 45 – Home page of the example's matching case


Allied Bank Plaza: Detailed_Text - Netscape
File Edit View Go Communicator Help
Back Reload Home Search Guide Print Security
Document: Done

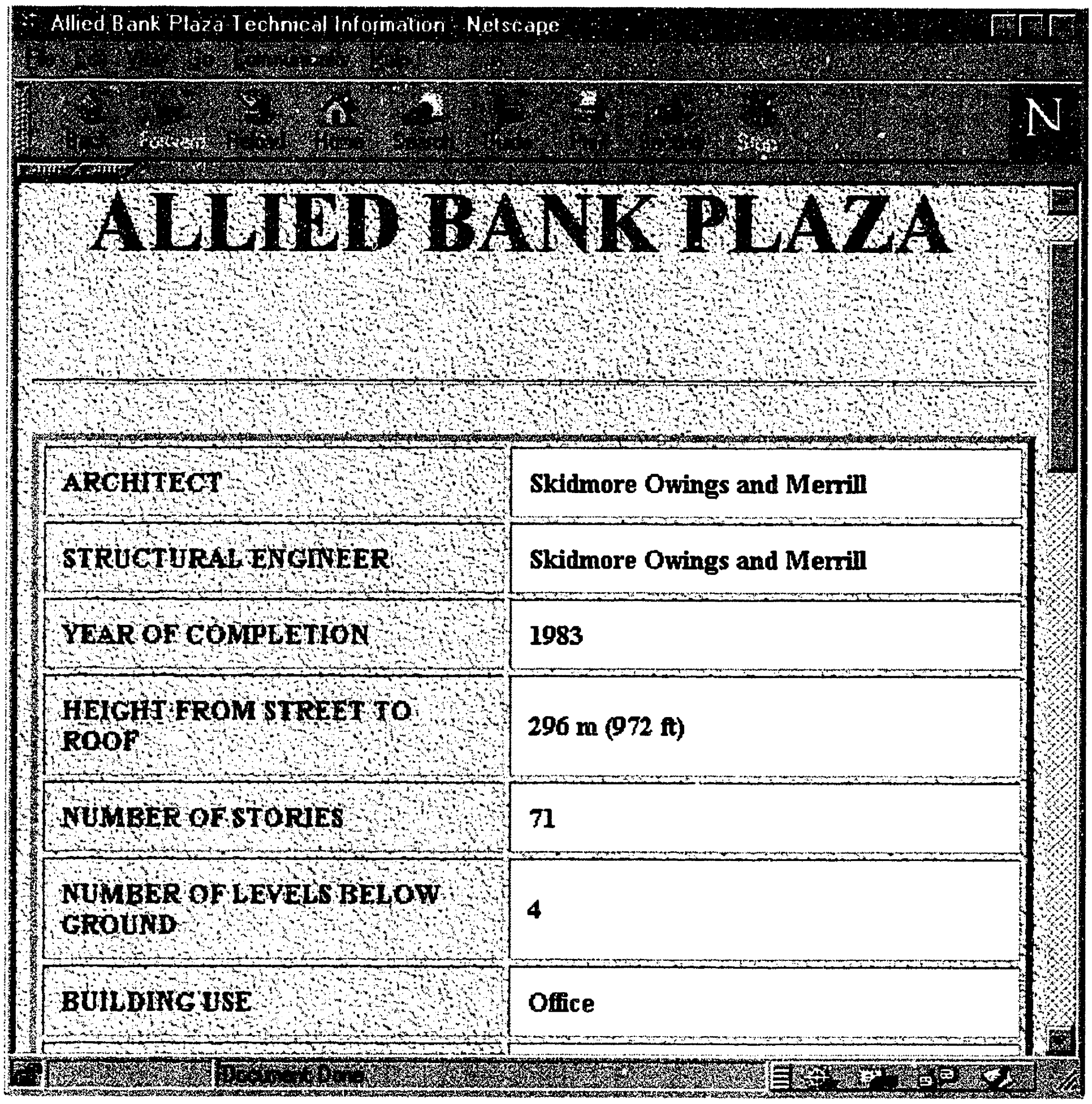
ALLIED BANK PLAZA

Allied Bank Plaza was designed to relate strongly to the buildings around it. Situated on a site which is essentially the center of downtown Houston, the building has a major impact on the western facade of the city, which is the most dominant view of its skyline. In form and materials, a design was sought which would be distinctive but would serve to complement and tie together its surroundings. A form that moved and flowed was felt to be appropriate, one that was soft and sheer rather than hard and opaque like the granite and steel rectangular buildings around it.

The resulting semicurved tower was achieved by juxtaposing two quarter-cylinder shafts. The 71-story tower is sheathed in dark green reflective glass, chosen for its sheer quality and responsiveness to light. The combination of plans and curves in the building's design will allow a constant interplay of sunlight on its surface.

Giving the building a human scale was another important aspect of the designer's intentions. Unlike many recent buildings, which are sheathed in reflective glass and appear only as a huge mass, the structure of the Allied Bank Plaza is subtly expressed with vertical and horizontal mullions. A formal portal on the east side of the building provides a sense of entry. Since 65%

Figure 46 – Text description of the example's matching case



ALLIED BANK PLAZA

ARCHITECT	Skidmore Owings and Merrill
STRUCTURAL ENGINEER	Skidmore Owings and Merrill
YEAR OF COMPLETION	1983
HEIGHT FROM STREET TO ROOF	296 m (972 ft)
NUMBER OF STORIES	71
NUMBER OF LEVELS BELOW GROUND	4
BUILDING USE	Office

Figure 47 – Structural information of the example's matching case

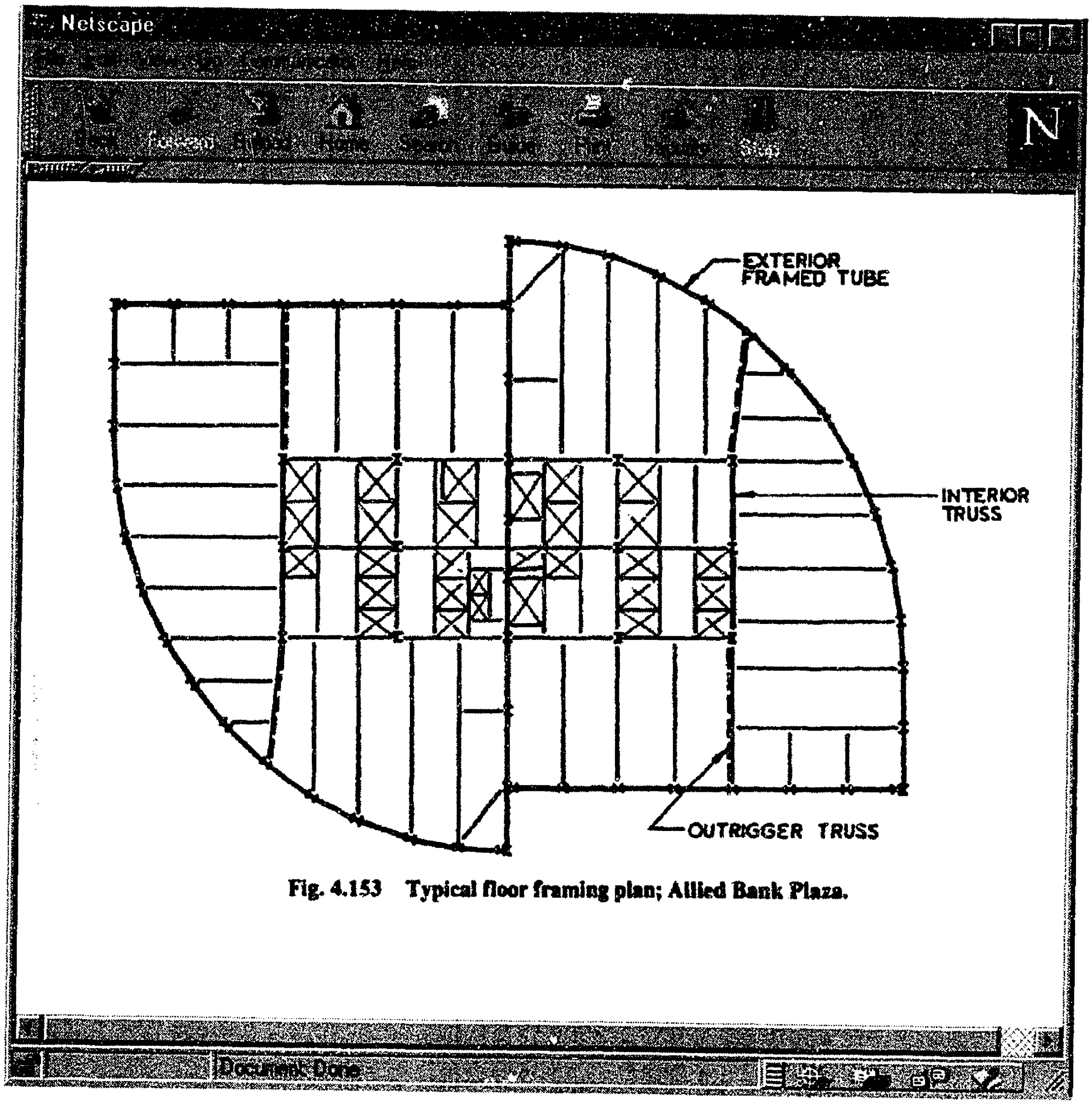


Fig. 4.153 Typical floor framing plan; Allied Bank Plaza.

Figure 48 – Structural detail of the example's matching case

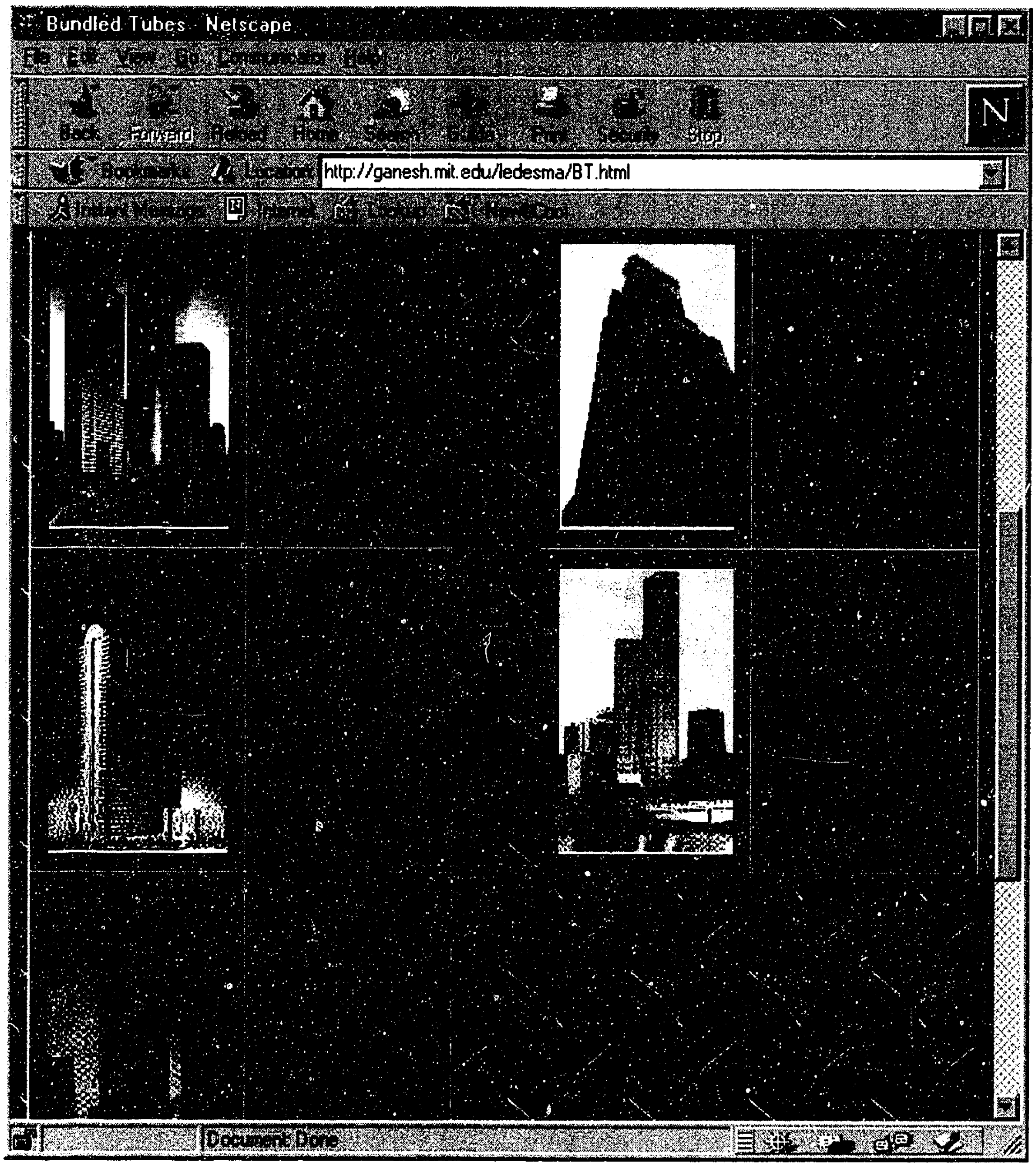

Bundled Tubes - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://ganesh.mit.edu/ledesma/BT.html
Instant Message Internet Lookup New&Cool
Document: Done

Figure 49 – Home page of indexed cases of the example's matching class

The attributes and corresponding values for the best matching retrieved case are shown here:

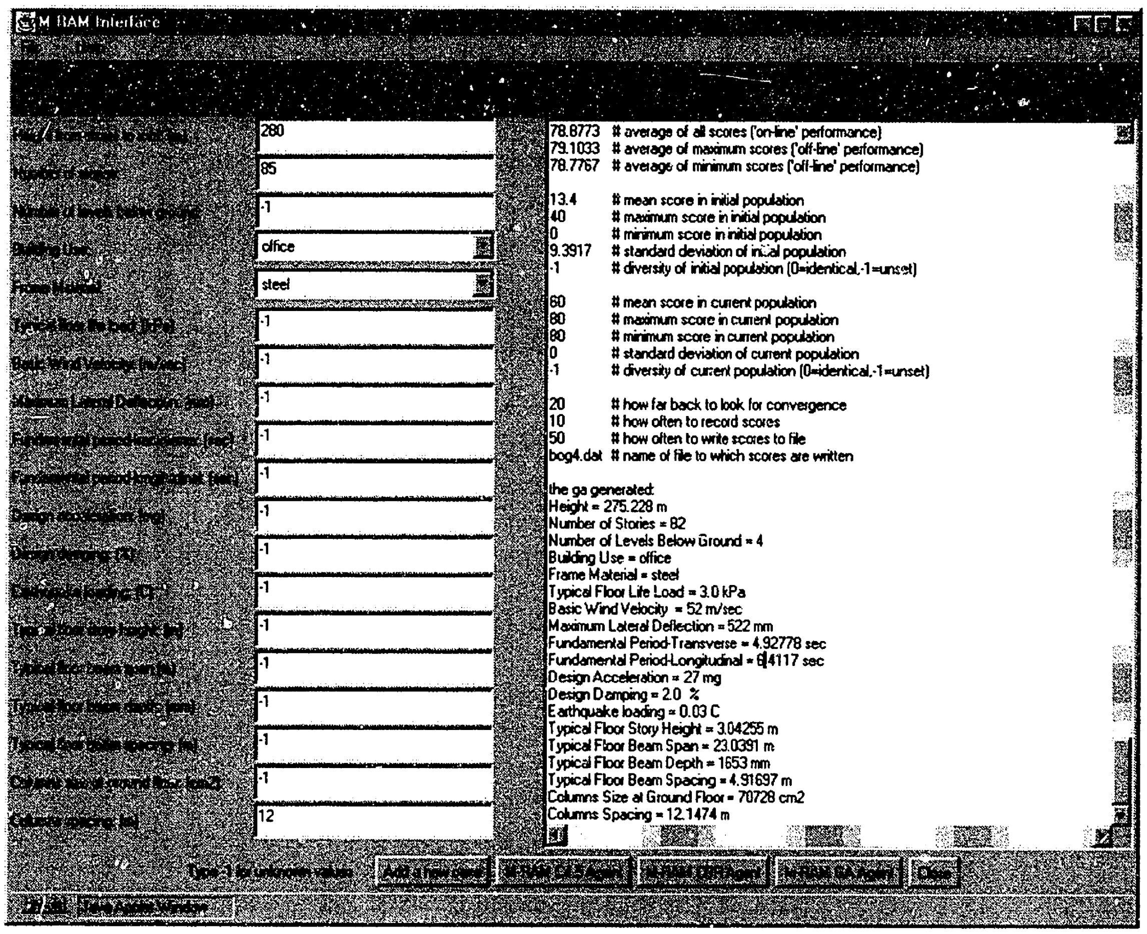
• Case:	Allied Bank Plaza, Houston, Texas, USA
• ***Height from the street to the roof:***	***296 m***
• ***Number of Stories:***	***71***
• Number of stories below ground:	4
• ***Building use:***	***Office***
• ***Frame material:***	***Steel***
• Typical life load:	2.5 kPa
• Basic wind velocity:	Unavailable [force = 196 kN/m]
• Maximum lateral deflection:	H/500
• Design fundamental period:	Not available
• Design acceleration:	Not available
• Design damping:	1% serviceability
• Earthquake loading:	Not applicable
• Type of structure:	Perimeter framed tube; diagonally braced core with outrigger trusses
• Typical floor:	
• Story height:	4.0 m
• Beam span:	15.2 m
• Beam depth:	530 mm
• Beam spacing:	5.6 m
• Material:	Steel, grade 250 MPa
• Slab:	83 mm concrete on 76 mm metal deck
• ***Columns:***	Build-up, 1016 by 610 mm in perimeter; 610 by 610 mm interior

• ***Spacing:***	***4.6 m perimeter; 9.15 by 6.1 m interior***
• Material:	Steel, grade 250 and 350 MPa
• Core:	Braced steel frame, grade 350 MPa

When comparing the problem requirements with the retrieved case attributes it is possible to see that the retrieved case matches the specifications on the attributes height from the street to the roof, building use and structural material. The retrieved case is a building approximately as tall as the one being designed built with the same structural material and for the same use. Unfortunately the retrieved case doesn't completely match the specifications on the other two requirements: number of floors and column spacing.

The building being designed has almost the same total height if compared with the retrieved case but because of the fact that it has a larger number of floors it is possible to conclude that it is going to have shorter floor-to-floor heights. The retrieved solution is not able to teach the designer how to deal with shorter floor-to-floor heights and with larger distance among columns. The third M-RAM agent is then used to transform the retrieved solution by mixing parts of partial matches available in M-RAM's database and retrieved by the CBR agent.

To do so the designer needs to press the M-RAM GA Agent button. The manager then sends the design attributes that will be used as input to the objective function as demonstrated in Section 4.6.2.3 added to the attributes of different cases retrieved by the CBR agent that will be used as genetic seed for the initial genetic algorithm population to the server were M-RAM's genetic algorithm agent is located. The genetic algorithm agent runs for 1,500 generations mixing bit and parts of the design cases provided by the CBR agent sending its solution back to the manager that display it to the user in the M-RAM interface text area (Figure 50).



Height from ground to roof [m]	280
Number of stories	85
Number of levels below ground	-1
Building Use	office
Frame Material	steel
Typical floor live load [kPa]	-1
Basic Wind Velocity [m/sec]	-1
Maximum Lateral Deflection [mm]	-1
Fundamental period-transverse [sec]	-1
Fundamental period-longitudinal [sec]	-1
Design acceleration [mg]	-1
Design damping [%]	-1
Earthquake loading [C]	-1
Typical floor story height [m]	-1
Typical floor beam span [m]	-1
Typical floor beam depth [mm]	-1
Typical floor beam spacing [m]	-1
Column size at ground floor [cm2]	-1
Columns spacing [m]	12

```
78.8773  # average of all scores ('on-line' performance)
79.1033  # average of maximum scores ('off-line' performance)
78.7767  # average of minimum scores ('off-line' performance)

13.4     # mean score in initial population
40       # maximum score in initial population
0        # minimum score in initial population
9.3917   # standard deviation of initial population
-1       # diversity of initial population (0=identical,-1=unset)

60       # mean score in current population
80       # maximum score in current population
80       # minimum score in current population
0        # standard deviation of current population
-1       # diversity of current population (0=identical,-1=unset)

20       # how far back to look for convergence
10       # how often to record scores
50       # how often to write scores to file
bog4.dat # name of file to which scores are written

the ga generated:
Height = 275.228 m
Number of Stories = 82
Number of Levels Below Ground = 4
Building Use = office
Frame Material = steel
Typical Floor Life Load = 3.0 kPa
Basic Wind Velocity  = 52 m/sec
Maximum Lateral Deflection = 522 mm
Fundamental Period-Transverse = 4.92778 sec
Fundamental Period-Longitudinal = 6.4117 sec
Design Acceleration = 27 mg
Design Damping = 2.0  %
Earthquake loading = 0.03 C
Typical Floor Story Height = 3.04255 m
Typical Floor Beam Span = 23.0391 m
Typical Floor Beam Depth = 1653 mm
Typical Floor Beam Spacing = 4.91697 m
Columns Size at Ground Floor = 70728 cm2
Columns Spacing = 12.1474 m
```

Figure 50 - M-RAM interface with GA agent output

The complete output from the genetic algorithm agent that is printed in the text area of the M-RAM interface can be seen in figures 51 and 52.

```
M-RAM GA Agent
by Lucio Soibelman - March, 1998.


running M-RAM ga ...Please WAIT....
te ga statistics:
1500    # current generation
1       # current convergence
39000   # number of selections since initialization
33792   # number of crossovers since initialization
6971    # number of mutations since initialization
37500   # number of replacements since initialization
34483   # number of genome evaluations since initialization
1501    # number of population evaluations since initialization
80      # maximum score since initialization
0       # minimum score since initialization
78.8773 # average of all scores ('on-line' performance)
79.1033 # average of maximum scores ('off-line' performance)
78.7767 # average of minimum scores ('off-line' performance)

13.4    # mean score in initial population
40      # maximum score in initial population
0       # minimum score in initial population
9.3917  # standard deviation of initial population
-1      # diversity of initial population (0=identical,-1=unset)

80      # mean score in current population
80      # maximum score in current population
80      # minimum score in current population
0       # standard deviation of current population
-1      # diversity of current population (0=identical,-1=unset)

20      # how far back to look for convergence
10      # how often to record scores
50      # how often to write scores to file
bog4.dat# name of file to which scores are written
```

Figure 51 - M-RAM GA agent output part I

```
the ga generated:
Height = 275.228 m
Number of Stories = 82
Number of Levels Below Ground = 4
Building Use = office
Frame Material = steel
Typical Floor Life Load = 3.0 kPa
Basic Wind Velocity  = 52 m/sec
Maximum Lateral Deflection = 522 mm
Fundamental Period-Transverse = 4.92778 sec
Fundamental Period-Longitudinal = 6.4117 sec
Design Acceleration = 27 mg
Design Damping = 2.0  %
Earthquake loading = 0.03 C
Typical Floor Story Height = 3.04255 m
Typical Floor Beam Span = 23.0391 m
Typical Floor Beam Depth = 1653 mm
Typical Floor Beam Spacing = 4.91697 m
Columns Size at Ground Floor = 70728 cm2
Columns Spacing = 12.1474 m
```

Figure 52 - M-RAM GA agent output part II

The output of the genetic algorithm agent can be divided in two parts. The first part prints statistics information about the genetic algorithm and the second part presents the genetic algorithm proposed solution to the design problem. Analyzing the potential solution generated above at a glance it is apparent that it presents a better solution for the design problem. It proposed a tall building with the same use, structural material and nearly the same height, number of floors, and columns spacing as that indicated in the problem specification. As expected, it has shorter floor-to-floor height if compared with the case retrieved by the CBR agent.

5.3 Chapter Conclusions

This example shows how M-RAM supports structural designers during the conceptual phase of structural design of tall buildings. It shows how M-RAM can be used by the structural designer to first classify the design problem, then to retrieve past experience and finally to adapt retrieved examples.

Usually the information at the conceptual design phase is not sufficient for an engineering analysis to determine the feasibility of a solution being difficult to generalize the knowledge for proposing feasible modifications to existing design cases. The example demonstrated how M-RAM hide all this from the designer providing him with knowledge of similar past design cases and with a good adapted solution allowing him to increase his productivity by cutting the number of design-analysis cycles.

Chapter 6

6 Conclusions

6.1 Introduction

The use of computer-aided design, as well as the development of a computer supported design environment can produce fundamental changes not only in the speed of the design, but also in the design process itself.

To demonstrate the power of a computer supported design environment this dissertation has developed a model - M-RAM – that represents the conceptual phase of structural design. The M-RAM model proposes a design solution through the use of the following constructs: proposal, recommendation, intent, artifact, justification, and a computer designer divided in a manager, a user interface, a classification engine, a past experience engine, and an adaptation engine. This model provides relationships that ensure a structure is followed during the design process.

The model and the system prototype proposed in this work are implemented following an object-oriented methodology. The constructs are independent objects which contain methods that specify their role and behavior in any context.

M-RAM performs three basic functions: classification, retrieval of past experience, and adaptation of past experience. The classification function uses a decision three builder to be able to classify the design problem. The retrieval of past experience function uses a case based reasoning engine to retrieve matching past experience. Finally the case adaptation function uses a genetic algorithm engine to mix bits and parts of past examples creating adapted solution to the design problem being solved.

The use of the model and prototype presented in this dissertation improves the design process both in terms of time and quality. The required design-analysis cycle do not need to be repeated several times during the design process simply because M-RAM provides the designer with a good initial input for the design process. In addition, the ability of the system to retrieve lessons learned in previous similar designs allow the designer to correct problems avoiding repeating past mistakes in new designs.

6.2 Benefits of the Model

Every design involves first of all a confrontation of the problem with what is known or not known. The intensity of this confrontation depends on the knowledge, ability and experiences of the designer. In all cases, however, more detailed information about the task itself, about constraints, about possible solution principles, and about known solutions for similar problems is extremely useful as it clarifies the precise nature of the requirements. Usually the information at the conceptual design stage is not sufficient for an engineering analysis to determine the feasibility of a proposed solution. M-RAM model supports designers providing the needed information in an organized and reliable structure.

This model is an improvement over the existing design process. Designers depend solely in their knowledge and experience to provide the assumptions they make during the initial stages of structural design, as well as the specifications of the design artifact. These initial assumptions have a strong impact in the quality of the final design and consequently in the quality of the final artifact. This model improves the conceptual design by augmenting the designer knowledge and experience. The well-organized database added by different functions provided by the model helps the designer to classify the design problem, retrieve useful past experience, and to adapt past experience providing the needed information to improve the quality and speed of the design process. In addition, the model provides a structured representation for providing information to the designer. Hence, the user will not be exposed to radical changes in their work technologies neither will be replaced by an automated computer designing system. This model just supports the actual design process by providing the required information that helps the designer to elaborate the design.

Another benefit of using the model is that it provides justification for the design decisions. Past experience obtained by retrieved matching cases provides a good justification for design decisions. Past success and failures with the provided relationship between design and problems are recognized as the most powerful form of design justification.

M-RAM can help not just the professional designer but can be a very powerful tool to support structural engineering education. Professional structural designers will benefit because through M-RAM the structural design knowledge will be easily available. This will help designers not only to verify their decisions but will help them to speed the design process reducing the number of design-analysis cycles. M-RAM will also enhance the experienced designer's ability to explore innovative structural concepts.

The structural engineering education has even more to benefit since M-RAM would be able to help students to quickly assimilate knowledge learning with past design examples. Special tutors can be easily developed as an add on to the proposed model.

6.3. Contributions

In summarizing this work, the contributions can be divided in three parts. First a model for representing the conceptual phase of structural design was developed. This model provides primitives for representing design knowledge in terms of the reasoning process used by designers in generating an artifact satisfying their design intents. This model also takes into consideration the different reasoning mechanisms that collaborate to solve the design problem and is used to provide active computer support for designers in the reasoning process.

The second contribution was the development of a distributed open system architecture that integrates different reasoning mechanisms. A novel system architecture was applied where different reasoning mechanisms can be located in different servers dispersed in diverse locations around the world. Unlike previous work in Distributed Artificial Intelligence this research had the objective of integrating agents that were not intended to collaborate to reach a common solution.

Finally, the third contribution is related to the fact that this work provided a pioneer understanding of how future intelligent agents will interact in an automated collaborative design system.

6.4 Future Research

As can be seen in Chapter 5, M-RAM is able to support designers in the conceptual phase of structural design. However, many other issues need to be resolved for the effective implementation of M-RAM in a real world-working environment. These issues are:

1. To ensure a robust model, the database, collected from Kowalczyk et al. [1995] needs to be improved, by including new and future building being built presently.

2. Testing of both the model and the prototype have been limited to small-scale examples. Further testing is necessary to validate the extensibility of the model. The system has to be tested by comparing its results to designs done by human designers.

3. There is the question of how fast can the model evolve to support new design trends. How fast can the database be updated to be able to provide designers with state-of-the-art solutions? How are the innovative and evolving systems placed within the classification scheme such that cataloging and data collection of structural systems can be continuously updated and of use to the practicing engineer?

4. The proposed system architecture using CORBA standards allows easily exchange of available agents for more powerful ones. However there is the need to continue the search for more suitable agents that allows:

   - New classification engine that allows incremental induction. When adding new data to the training set the actual C4.5 implementation discards the previous classifier building a new one from scratch with height computational cost.

   - New CBR agent algorithm to allow the retrieval of cases ordered according to their suitability. The actual Caspian implementation retrieves the best match or a list of all the cases that match the indexed attribute. We still need tools for making case collection and indexing easier. Who can update a case library? When does it get done? Those are very important issues confronting those whose systems have multiple users in multiple locations.

   - New GA agent developed for a continuous design space optimization that uses GA operators and strategies tailored to the structure and properties of engineering design domains. The fitness in the M-RAM GA agent implementation is obtained with the use of design specifications denoted as constraints. These constraints only consider the information available with the case description making it difficult to provide comprehensive evaluation of feasibility. One possible approach is to use a neural network with emphasis on pattern matching rather than in domain formalization.

5. Finally, the biggest technological issue is scale-up. How can we make retrieval algorithms that work for hundreds of cases work efficiently enough for thousands of cases? M-RAM with its classification agent that reduces the search space represents

an improvement to the scale-up problem so common in the majority of CBR systems but still its behavior is untested in very large databases.

REFERENCES

[Barr amd Feigenbaum, 1981] Barr, A. and Feigenbaum, E.A. (1981) "*The Handbook of Artificial Intelligence,*" Vol 1, Morgan Kaufmann, Los Altos, CA.

[Booch, 1994] Booch, G. (1994) "*Object-Oriented Analysis and Design With Applications,*" The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA.

[Bond and Gasser 1988] Bond, A. H. and Gasser, L. (1988) "An Analysis of Problems and Research in Distributed Artificial Intelligence," in *Readings in Distributed Artificial Intelligence,* Morgan Kaufmann San Mateo, CA

[Brachman and Anand 1996] Brachman, R.J. and Anand, T. (1996) "The Process of Knowledge Discovery in Databases in Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. editors "*Advances in Knowledge Discovery and Data Mining,*" AAAI Press/The MIT Press, Cambridge, MA

[Chaib-draa 1995] Chaib-draa, B. (1995) "Industrial Applications of Distributed AI," *Communications of the ACM*, Vol. 38, num. 11, pages 49-55.

[Cook 1986] Cook, J. (1986) "The Base Selection Task in Analogical Planning," *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*

[Cox, 1994] Cox. E. (1994) *"The Fuzzy Systems Handbook – A Practitioners Guide to Building, Using, and Maintaining Fuzzy Systems,"* AP Professional, Cambridge, MA

[Dertouzos, 1997] Dertouzos, M. (1997*) "What Will Be - How the New World of Information Will Change our Lives,*" HarperCollins, New York, NY

[Dixon, 1966] Dixon, J.R. (1996) *"Design Engineering Inventiveness, Analysis and Decision,"* McGraw-Hill, New York, NY.

[Dixon and de Kleer, 1988] Dixon, M and de Kleer, J. (1988) "Massively Parallel Assumption-based Truth Maintenance", *Proceedings of the National Conference on Artificial Intelligence (AAAI-88)*

[Durfee and Lesser, 1987] Durfee, E.H. and Lesser, V.R. (1987) "Using Partial Global Plans to Coordinate Distributed Problem Solvers," *Proceedings of the 1987 International Joint Conference on Artificial Intelligence,* pages 875-883.

[Drumheller, 1986] Drumheller, M. (1986) "Connection Machine Stereomatching," *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*

[Fayyad et al., 1996] Fayyad, U.M., Piatetsky-Shapiro, G. and Smyth, P. (1996) "From Data Mining to Knowledge Discovery: An overview in Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. editors *"Advances in Knowledge Discovery and Data Mining,"* AAAI Press/The MIT Press, Cambridge, MA.

[Fenves et al., 1995] Fenves, S.J., Rivard, H., Gomez, N. and Chiou, S.C. (1995) "Conceptual Structural Design in Seed," *Journal of Architectural Engineering*, Dex. 1995, Vol.1No.4

[Finin, et al., 1992] Finin, T., Fritzson, R. And McKay, D. (1992) "A Language and Protocol to Support Intelligent Agent Interoperability," *Proceedings of the CE&CALS Washington 92 Conference.*

[Finin, et al., 1992] Finin, T., Fritzson, R., McKay, D. and McEntire, R. (1994) "KQML as an Agent Communication Language," *Proceedings of the Third International Conference on Information and Knowledge Management*, ACM Press.

[French, 1985] French, M. (1985) *"Conceptual Design for Engineers,"* second edition, Design Council, London, UK.

[Fowler, 1997] Fowler, M. (1997) *"UML Distilled – Applying The Standard Object Modeling Language,"* Addison-Wesley, Reading, MA

[Gardam, 1994] Gardam, A. (1994) "The Impact of Design Tools on the Engineering Design Process in Practice," in Sharpe, J. and Oh, V. editors Computer Aided Conceptual Design, LooseLeaf Co., Lancaster, UK.

[Gasser, 1992] Gasser, L. (1992) "An Overview of DAI" in Les Gasser and Nicholas M.Avouris editors, *Distributed Artificial Intelligence: Theory and Praxis,* Kluwer, Dordrecht, Netherlands

[Geller, 1991] Geller, J. (1991) *"Advanced Update Operations in Massively Parallel Knowledge Representation,"* New Jersey Institute of Technology, CIS-91-28

[Gero and Quian, 1992] Gero, J.S. and Quian,L (1992) "A Design System Using Analogy," *Artificial Intelligence in Design* '92, Kluwer, Dordrecht, The Netherlands

[Goldberg, 1989] Goldberg, D.E. (1989) *"Genetic Algorithms in Search, Optimization, and Machine Learning,"* Addison Wesley, Reading, MA.

[Haykin, 1994] Haykin, S. (1994) *"Neural Networks – A Comprehensive Foundation,"* Macmillan College Publishing Company, New York, NY.

[Hubka, 1982] Hubka, V. (1982) *"Principles of Engineering Design,"* Butterworth Scientific, London, UK.

[Huhns and Bridgeleand, 1991] Huhns, M.N. and Bridgeleand, D.M. (1991) "Multiagent Truth Maintenance," *IEEE Transactions on Systems Man. and Cybernetics*, Vol.21, No.6. November-December, 1991

[Jackson, 1990] Jackson, P. (1990) *"Introduction to Expert Systems,"* Addison Wesley, Reading, MA

[Kettler et al.] Kettler, B., Andersen, W., Hendler, J. and Evett, M, (1993) "Massive Parallel Support for Case Based Planning System," *Proceedings of the Ninth IEEE Conference on AI Applications,* Orlando, Florida

[Kolodner, 1988] Kolodner, J. (1988) "Retrieving Events from a Case Memory: A Parallel Implementation", *Proceedings of Case Based Reasoning Workshop*

[Kolodner, 1993] Kolodner, J. (1993) *"Case-Based Reasoning,"* Morgan Kaufmann, San Mateo, CA.

[Koza, 1996] Koza, J.R. (1996) *"Genetic Programming - On The Programming of Computers by Means of Natural Selection,"* MIT Press, Cambridge, MA.

[Kowalczyk et al., 1995] Kowalczyk, R.M., Sinn, R. and Kilmister, M.B. (1995) *"Structural Systems for Tall Buildings,"* Council on Tall Buildings and Urban Habitat - McGraw-Hill, New York, NY

[Kuipers, 1994] Kuipers, B (1994) *"Qualitative Reasoning – Modeling and Simulation with Incomplete Knowledge,"* The MIT Press, Cambridge, MA

[Lander and Lesser, 1989] Lander, S. and Lesser, V.R. (1989) "A Framework for the Integration of Co-operative Knowledge-Based Systems", *Proceedings of IEEE International Symposium on Intelligent Control,* pp. 472-477.

[Leake, 1996] Leake, D.B. (1996) "CBR in Context: The Present and the Future," in David B. Leake editor, *Case-Based Reasoning -Experience, Lessons, and Future Directions,* AAAI Press/The MIT Press, Cambridge, MA.

[Lenat, 1975] Lenat, D.B. (1975) "Beings:Knowledge as Interacting Experts," *Proceedings of the 1975 International Joint Conference on Artificial Intelligence,* pp. 126-133.

[Lesser and Durfee, 1989] Lesser, V.R. and Durfee, E.H. (1989) "Negotiating Task Decomposition and Allocation Using Partial Global Planning," in Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence, Vol II,* Morgan Kaufmann, San Mateo, CA.

[Lichter et al., 1994] Lichter, H., Schneider, M.H. and Zullighoven, H. (1994) Prototyping in Industrial Software Projects – Bridging the Gap Between Theory and Practice," *IEEE Transactions on Software Engineering*, vol.20, no.11, pp. 825-832, November

[Maes, 1991] Maes, P. (1991) *"Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, "* The MIT Press, Cambridge, MA.

[Maes, 1993] Maes, P. (1993) "Behavior-Based Artificial Intelligence," *Proceedings of the Second Conference on Adaptive Behavior,* MIT Press, Cambridge, MA.

[Maher, 1987] Maher, M.L. (1987) "Expert Systems for Structural Design," *Journal of Computing in Civil Engineering*, ASCE, 1(4), 270-283.

[Maher and da Silva Garza, 1996] Maher, M.L. and de Silva Garza, A.G. (1996) "The Adaptation of Structural Systems Designs Using Genetic Algorithms," in Kumar B. editor, *Information Processing in Civil and Structural Engineering Design*, Civil-Comp Press

[Maher and Fenves, 1985] Maher, M.L. and Fenves, S.J. (1985) "HI-RISE: An Expert System for the Preliminary Structure Design of High Rise Buildings," in Gero, J.S. editor, *Knowledge Engineering in Computer-aided Design*, North-Holland, Amsterdan.

[Maher and Zhao, 1992] Maher, M.L. and Zhao, F. (1992) "Using Network-Based Prototyped to Support Creative Design by Analogy and Mutation," *Artificial Intelligence in Design '92*, Kluwer, Dordrecht, The Netherlands

[Malone et al., 1988] Malone, T.W., Fikes, R.E., Grant, K.R and Howard, M.T. (1988) "Enterprise: A Market-like Task Scheduler for Distributed Computer Environments," in B.A. Huberman editor *The Ecology of Computation*, pp 177-205, North-Holland

[Manfaat et al., 1997] Manfaat, D., Duffy, A.H.B. and Lee, B.S. (1997) "*SPIDA: for the Utilization of Layout Design Experience*," University of Strathclyde.

[Minsky, 1988] Minsky, M. (1988) *"The Society of Mind,"* Touchstone/Simon and Schuster, New York, NY

[Mitchell, 1977] Mitchell, M. (1997) *"An Introduction to Genetic Algorithms,"* The MIT Press, Cambridge, MA.

[Mowbray and Ruh, 1997] Mowbray, T.J. and Ruh, W.A. (1997) *"Inside Corba – Distributed Object Standards and Applications,"* Addison Wesley Longman, Inc, Reading, MA

[Mukherjee and Despande, 1995] Mukherjee, A. and Deshpande, J.M. (1995) "Modeling Initial Design Process Using Artificial Neural Networks," *Journal of Computing in Civil Engineering*, ASCE, 9(3), 194-200.

[OMG, 1997] Object Management Group (1997) *"The Common Object Request Broker: Architecture and Specification,"* Revision 2.1., Object Management Group Inc.

[Orfali and Harkey, 1997] Orfali, R. and Harkey, D. (1997) *"Client/Server Programming with Java and Corba,"* John Wiley and Sons, New York, NY.

[Pahl and Beitz, 1996] Pahl, G. and Beitz, W. (1996) *"Engineering Design: A Systematic Approach,"* Springer-Verlag, New York, NY.

[Peña-Mora, 1994] Peña-Mora, F. (1994) "Design Rationale for Computer Supported Conflict Mitigation During the Design-Construction Process of Large-Scale Civil Engineering Systems" Ph.D. Thesis, Massachusetts Institute of Technology

[Peña-Mora et al., 1995] Peña-Mora, F., Sriram, D. and Logcher, R. (1995) "Design Rationale for Computer-Supported Conflict Mitigation," *Journal of Computing in Civil Engineering*, ASCE, 1(9), 57-74.

[Pope et al., 1992] Pope, R.P., Conry, S.E. and Mayer, R.A. (1992) "Distributing the Planning Processing a Dynamic Environment," *Proceedings of the Eleventh International Workshop on Distributed Artificial intelligence, pp.* 317-331, February.

[Pouangare, 1986] Pounangare C.C. (1986) *"Strategies for the Conceptual Design of Structures,"* M.S. Thesis, Massachusetts Institute of Technology.

[Quinlan, 1993] Quinlan, J.R., (1993) *"C.4.5 Programs for Machine Learning,"* Morgan Kaufmann, San Mateo, CA.

[Rao and Rao, 1995] Rao, V.B. and Rao, H.V. (1995) "C++, Neural Networks and Fuzzy Logic", Second Edition, MIS Press, New York, NY.

[Rasheed et al., 1997] Rasheed, K., Hirsh, H. and Gelsey, A. (1997) "A Genetic Algorithm for Continuous Design Space Search," To appear in *AI in Engineering*.

[Rational Software Corporation, 1997a] Rational Software Corporation (1997) "*Unified Modeling Language - Notation Guide - Version 1.0,*" http://www.rational.com.

[Rational Software Corporation, 1997b) Rational Software Corporation (1997) "*Unified Modeling Language - Process-Specific Extensions - Version 1.0,*" http://www.rational.com

[Rational Software Corporation, 1997c] Rational Software Corporation (1997) "*Unified Modeling Language - UML Summary - Version 1.0.1,*" http://www.rational.com

[Rational Software Corporation, 1997d] Rational Software Corporation (1997) "*Unified Modeling Language - UML Semantics - Version 1.0,*" http://www.rational.com

[Robertson, 1987] Robertson, G. (1987) "Parallel Implementation of Genetic Algorithms in a Classifier System," in Davis, L. editor, *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, San Mateo, CA

[Rosenchein and Zlotkin, 1994] Rosenchein, J.S. and Zlotkin, G. (1994) *"Rules of Encounter - Designing Conventions for Automated Negotiation among Computers," The* MIT Press, Cambridge, MA

[Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991) *"Object-Oriented Modeling and Design,"* Prentice Hall Inc., Englewood Cliffs, NJ.

[Serrano, 1987] Serrano, D. (1987) *"Constraint Management in Conceptual Design,"* Ph.d. Thesis, Massachusetts Institute of Technology

[Sharples et al., 1989] Sharples, M., Hogg, D., Hutchinson C., Torrance, S. and Young, D. (1989*) "Computers and Thought: A Practical Introduction to Artificial Intelligence"* The MIT Press, Cambridge, MA.

[Shoham and Tennenholtz, 1993] Shoham, Y. and Tennenholtz, M. (1993) "On Social Laws for Artificial Agent Societies: Of-line Design," *Artificial Intelligence.*

[Smith, 1980] Smith, R.G. (1980) "The Contract Net Protocol: Hight-level Communication and Control in a Distributed Problem Solver," *IEEE Transactions on Computers,* pp 1104-1113, December

[Stanfill et al., 1989] Stanfill, C., Thau, T. and Waltz, D. (1989) "A Parallel Indexed Algorithm for Information Retrieval," *Proceedings of the SIG-IR-89*

[Sycara, 1989] Sycara, K. (1989) "Co-operative Negotiation in Concurrent Engineering Design," *Proceedings of MIT-JSME Workshop in Computer-Aided Co-operative Product Development,* pp. 269-297.

[UW, 1996a] UW Aberystwyth (1996) *"An Introductory Guide to Caspian"*, University of Wales – Aberrystwith, Aberrystwith, UK.

[UW, 1996b] UW Aberystwyth (1996) *"Creating a Case-Base Using CASL"*, University of Wales – Aberrystwith, Aberrystwith, UK.

[Wall, 1996] Wall, M. (1996) *"GAlib: A C++ Library of Genetic Algorithm Components – Version 2.4 – Revision B"*, Mechanical Engineering Department-Massachusetts Institute of Technology, Cambridge, MA.

[Wilkenfeld and Kraus, 1993] Wilkenfeld, J. and Kraus, S. (1993) "A Strategic Negotiation Model with Applications to an International Crisis," *IEEE Transactions on Systems, Man. And Cybernetics*, Vol 23, No.01, January-February

[Woodson, 1966] Woodson, P.H. (1966) *"Introduction to Engineering Design,"* McGraw-Hill, New York, NY.

[Zlotkin and Rosenchein, 1991], Zlotkin, G. and Rosenchein, J.S. (1991) "Incomplete Information and Deception in Multi-Agent Negotiation," *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 225-231, August

THESIS PROCESSING SLIP

FIXED FIELD: ill. ______ name ______

index ______ biblio ______

► COPIES: [✓ Archives] Aero Dewey [✓ Eng] Hum

Lindgren Music Rotch Science

TITLE VARIES: ► ☐ ______

NAME VARIES: ► ☐ ______

IMPRINT: (COPYRIGHT) ______

► COLLATION: 174 l

► ADD. DEGREE: ______ ► DEPT.: ______

SUPERVISORS: ______

NOTES:

cat'r: date:

page: N180

► DEPT: C. E.

► YEAR: 1998 ► DEGREE: Ph. D.

► NAME: SOIBELMAN, Lucio