

INFORMATION TO USERS

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

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

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

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

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

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

RICE UNIVERSITY

A Computer-Based Tutor for Engineering Design

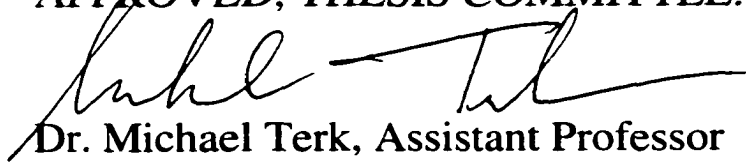
by

Prabhu Prakash Ganesh

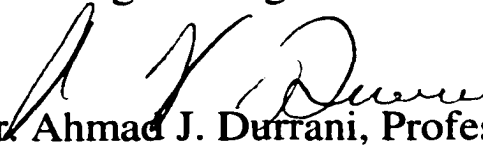
**A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE**

MASTER OF SCIENCE

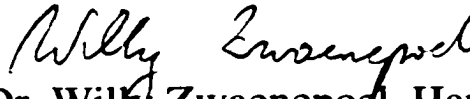
APPROVED, THESIS COMMITTEE:



**Dr. Michael Terk, Assistant Professor
Civil Engineering**



**Dr. Ahmad J. Durrani, Professor, Chair
Civil Engineering**



**Dr. Willy Zwaenepoel, Harding Professor
Computer Science**

**Houston, Texas
May 2000**

UMI Number: 1399297

UMI[®]

UMI Microform 1399297

Copyright 2000 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

A Computer-Based Tutor for Engineering Design

by

Prabhu PrakashGanesh

Modern computer technology facilitates development of rich learning environments that can enhance a student's ability to learn; however, none of the existing educational software systems can support the drill-and-practice mode of learning through the use of problem sets in routine engineering design domains. This thesis discusses the design of a computer-based system that benefits the instructor and the students in a design course by automating the creation of problem sets and their solution. The system allows the instructor to specify a set of design procedures as the design concept on which the generated problems should test the student. Each design procedure has a set of applicable conditions and these are formulated into a constraint satisfaction problem. Using the solution to this problem, the system then generates several problem descriptions along with their solution. The software, developed on a distributed component architecture model, is a general framework that could support multiple domains.

Acknowledgements

First, I would like to express my sincere thanks and gratitude to my advisor Dr. Michael Terk for his guidance, criticism and encouragement that were indispensable in the completion of my work. Without his help and encouragement, I wouldn't have been able to perform my research.

During my two years at Rice, I was fortunate enough to come in contact with many other wonderful people who had enriched my life and work at Rice. It is impossible to thank all of you in this page, but your contribution will stay with me for a lifetime. A special thanks to all my close friends with whom I had a wonderful time at Rice.

I want to thank my committee members, Dr. Willy Zwaenepoel and Dr. Ahmad J. Durrani. A special thanks to Dr. Panos Dakoulas for having been on my defense committee.

This research was supported by the Andrew W. Mellon foundation and my sincere thanks to the foundation for having supported this research work. I also want to thank all the people in the Civil Engineering Department who helped me in my study at Rice.

Finally, I would like to express my gratitude to my family, whose courage and sacrifice allowed me the opportunity to complete my Masters at Rice. Everything I have achieved is a testament to the values that they had instilled in me and to the support and encouragement they have provided to me.

Table of Contents

1. Introduction	1
1.1 Motivation	1
1.2 Requirements or Features of the Engineering Design Tutor	3
1.3 Organization	5
2. Background	7
2.1 Technology in Education and Learning	7
2.1.1 World Wide Web	7
2.1.2 Client-Server Systems	8
2.1.3 Object-Oriented Software	9
2.1.4 Client-Side User Interface	10
2.1.5 Technology Integration	10
2.2 Educational Software systems and their features	11
2.2.1 Authoring Systems	11
2.2.2 Collaborative Learning Systems	12
2.2.3 Virtual Reality Based Learning Systems	13
2.2.4 Assessment Systems	14
2.2.5 Intelligent Tutoring Systems	16
2.3 Comparison of the Features of Existing Educational Software Systems with Requirements of the Engineering Design Tutor (EDT)	18
3. Framework of EDT	22
3.1 Routine Engineering Design	22
3.1.1 Engineering Design Problems	23
3.1.2 Need for Automated Problem Generation	24
3.2 Methodology for Automated Problem Generation	26
3.3 The EDT Architecture	28
3.3.1 Domain Knowledge Component	30
3.3.2 Problem Generator Component	31
4. The Domain Knowledge Component	33
4.1 Domain Representation	33
4.2 SASE Methodology	34
4.2.1 Data Items	35
4.2.2 Decision Tables	36
4.3 Advantages of SASE Methodology	38
4.4 Domain Knowledge Management Interface	39
5. Problem Generator	43
5.1 Outline of the Problem Generator	43
5.2 Definitions	45
5.2.1 Decision Tree	45
5.2.2 Interaction Network	46
5.2.3 Problem Tree	48
5.2.4 Solution Path	49
5.2.5 Problem Path	50

5.3 Working of the Problem Generator	51
5.3.1 Problem Generator UI	51
5.3.1.1 Problem Domain	51
5.3.1.2 Problem Definition	52
5.3.2 Domain Knowledge Interface Component	53
5.3.3 Variable Identifier Component	54
5.3.4 Constraint Satisfaction Problem Formulator	55
5.3.5 Application Specific Post-Processor	57
5.3.6 Solution Generator	58
6. Constraint Satisfaction	59
6.1 Constraint Satisfaction Problem	59
6.2 Algorithms for Constraint Solving	61
6.2.1 Search Algorithms	62
6.2.2 Constraint Logic Algorithms	63
7. Software Architecture of the EDT	67
7.1 Distributed Component Architecture	67
7.2 DCA Models	68
7.3 Advantages of DCA	72
7.4 Distributed Component Architecture of the EDT	73
7.5 EDT Implementation	75
8. The EDT for Steel Member Design	77
8.1 Implementation Methodology	77
8.2 Example	80
9. Conclusion	87
9.1 Summary	87
9.2 Scope for Future Development	88
Bibliography	90
Appendix	93

List of Tables

Table 2.1 Comparison of features among different educational software systems	19
Table 4.1 Decision table for evaluating M_{nLTB}	37
Table 5.1 Decision table for evaluating M_{nLTB}	45

List of Figures

Fig 3.1 Outline of the EDT system	29
Fig 4.1 Decision table management user interface	41
Fig 4.2 The decision table viewer of the user interface	41
Fig 4.3 User interface to create a new decision table	42
Fig 5.1 Representation of the working of the problem generator	44
Fig 5.2 Decision tree for the decision table in Table 5.1	46
Fig 5.3 Interaction network in the domain steel beam design subject to bending	47
Fig 5.4 Problem tree for the interaction network in Fig 5.3	49
Fig 5.5 A Solution path from the problem tree in Fig 5.4	50
Fig 5.6 Another Solution path for the problem tree in Fig 5.4	50
Fig 5.7 The components of the problem generator	52
Fig 7.1 Software architecture of the EDT	74
Fig 8.1 UI for drawing a structural member and loads on it	81
Fig 8.2 UI for specifying testing rules	82
Fig 8.3 UI window for defining feasible values for variable	83
Fig 8.4 UI window that shows the generated problems	85
Fig 8.5 UI window that shows the solution procedure	86

Chapter 1

Introduction

This chapter describes the motivation and objectives of this research and outlines the remainder of the thesis.

1.1 Motivation

Incorporating technology into a student's learning greatly benefits students' understanding (Bazillion and Braun, 1998, Grossman, 1999). The use of technology has not only created new opportunities within the traditional classroom but has also served to expand learning experiences beyond the popular notion of "classroom." Indeed, "distance learning," with the utilization of the Internet, is becoming a widely used delivery alternative at universities nationwide. Educational software technology represents a new facet in learning because a software system can allow students to learn at their own pace and the students can use the software as often as they choose to. It also makes it possible to monitor a student's progress and tune the instruction to the needs of the student. In general, an educational software system provides one or more of the following characteristics:

- Serves as an effective pedagogical environment that has a positive impact on the students' learning
- Provides an easy means for instructors to present course material
- Enables easy accessibility for students
- Saves instructors' time and enables them to teach larger classes

This research work is targeted towards developing educational software systems in the field of engineering design.

In the domain of engineering, one of the main subjects that a student learns is design of components or systems. Practical engineering design is intuitive, and a successful engineer needs to have an intuition based on the understanding of the basic ideas of the behavior of the engineering system and the experience of having designed different systems. The only way a student can develop this intuition for various aspects of engineering design is by a drill-and-practice mode of learning, i.e., working on numerous similar problems. The instructor has to develop a set of problems that challenge and reinforce the concepts covered in the class and then gauge the level of understanding gained by the students based on their response.

The use of drill-and-practice approach to teaching engineering design is very costly in terms of instructor's time and has the shortcoming of failing to challenge and evaluate the learning process of majority of students. The instructor typically creates a single problem set that is targeted at the "average" student in class. However, the learning process and ability of each student is different. While some students might master a particular design concept by working out just one or two problems, other students may have to work on more problems before they understand a design concept fully. Hence, the number of assignment problems would vary from student to student and for the same student from one design concept to another. As a result the "average" problem set often fails to satisfy the needs of majority of the students in the class.

After an instructor has evaluated and returned a student's work, the student is expected to review the feedback and correct the mistakes. Because of the time constraints, it is impossible for an instructor in a medium to large size class to give personalized attention to each student to and ensure that they have relearned the concepts.

With computers having become an integral part of a student's life, it is now conceivable to think of an automated electronic system that would enhance the traditional drill-and-practice method of teaching engineering design.

1.2 Requirements or Features of the Engineering Design Tutor (EDT)

The need for a computer-based tutoring system that automates the drill-and-practice mode of learning for engineering design is established. Since this routine training method is common to several engineering domains, it would be very useful to have a general EDT framework that can be easily implemented for different domains. The central requirement or feature of the EDT is a problem generator capable of generating and evaluating a number of different problems testing the same design concept. By making the problem generation automatic, the system will enable the instructor to customize the homework to the needs of the individual students. Such a system has significant benefits for both the instructors and the learners.

Instructor's benefits:

- The system will reduce the burden of the instructors by transferring the problem set generation to the newly designed EDT architecture
- The system will greatly reduce the amount of time spent in grading students' work
- The previous two features will enable the instructor to customize a problem set for the ability of individual students. The instructor can test the students for competency by using "custom homework". This means that the system will continue to generate problems for students till they demonstrate their understanding by solving a problem correctly.

- The system would provide a significant reduction in the amount of lecture time that is currently devoted to in-class explanation of example problems. This would allow either a reduction of lecture time required to complete a course or will enable instructors to cover more material in the same amount of time.

Student's benefits:

- The tutoring system will help students to improve their understanding of the design concepts and to hone their design skills by exposing them to large sets of similar problems.
- The system will enable self-paced learning. If a student is not confident about a particular concept, the system can provide examples with solutions and generate additional problems that the student can use to test his/her understanding.
- The computer-based system will ensure that the students' performances are evaluated quickly thereby reducing the delay in providing feedback to them.

In addition to the key feature of automated problem generation, it would also be appropriate to incorporate the following features in the design of the EDT system.

1. **Easy accessibility and portability:** The software should be easily accessible so that the students can access the system at any time from anywhere, as best suits their needs. An easily accessible electronic system will also enable students to pace their study as required thereby making individualized attention possible.
2. **Flexibility:** Flexibility incorporated in the design of the software would enable it to be used for different applications and domains with minimum modifications. An

architecture that is transferable to other applications would greatly reduce the cost of building new tutoring systems for each domain. Also a flexible system would enable new features to be added easily without affecting the structure or design of the system.

3. Scalability: The computer-based system would be able to support a large number of users thereby allowing instructors to have more students their classes. The ability to scale the prototype for a larger user base would multiply the savings and benefits of the EDT system.

Recent computer and information technology developments eliminate technological barriers for building and incorporating the above features in the system. By using today's commonly accepted technologies the high custom design cost associated with older computer assisted training systems can be reduced.

1.3 Organization

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

Chapter 2, *Background*, provides a description of the technology used in developing learning environments for students and also discusses the features and capabilities of existing educational software systems

Chapter 3, *The Framework of EDT*, describes the general framework of the EDT system and how its requirements are incorporated into the design of the system

Chapter 4, *The Domain Knowledge Component*, discusses the organization and implementation of the domain knowledge component in the EDT

Chapter 5, *The Problem Generator*, describes the working of the problem generator component in the EDT

Chapter 6, *Constraint Satisfaction*, discusses the constraint satisfaction problem, which is an important step in problem generation. It also compares different solution techniques used for constraint solving and describes the features of the solver used in the problem generator

Chapter 7, *Software Architecture of the EDT*, describes the software architecture of the system, different models that can be used for implementing this architecture and how the choice of the model used in the EDT system was made

Chapter 8, *The EDT for Steel Member Design*, discusses how the general design of the EDT was implemented for the domain of steel member design and also illustrates the working of the system by providing examples

Chapter 9, *Conclusion*, provides a summary of the work done in this project and also provides a list of possible extensions

Chapter 2

Background

This chapter discusses the impact of technology on students learning and the features and capabilities of different educational software systems.

2.1 Technology in Education and Learning

The design and implementation of an electronic learning environment is technology dependent. Key technological developments are now in place to design learning environments, which could potentially change the way education is imparted in fundamental ways. Given the rate at which technology is being integrated into our lives, such environments are likely to be in use for a long time. With the appropriate technology and tools, it is possible to design electronic learning environments that either make use of existing pedagogical approaches or use new methods or paradigms for teaching and learning. It is important to understand that the objective of the EDT is not to implement a new pedagogical approach but to serve as a new means to teaching engineering design by automating the drill-and-practice mode of learning.

The following short discussion describes central technological elements that affect the quality of learning environments. These technologies have an impact not only on the users (instructors and learners) but also on the developers of the educational software systems.

2.1.1 World Wide Web (WWW)

The development of the World Wide Web (WWW) has allowed the Internet to be useful for all society sectors rather than just for scientists, engineers and professors. The Web is simply a way to access resources located anywhere on the Internet with the use of

the uniform resource locator (URL), hyperlinks and a user-friendly graphical user interface called a Web browser. The two major components of the WWW are hypertext transfer protocol (HTTP), the protocol used to access the resources, and hypertext markup language (HTML), a language to encode hyperlinks and other content. The WWW has made possible the large-scale distribution and publishing of online educational materials. Most of the existing educational learning environments use Internet as a communication means to deliver instructional content and also foster interaction. The WWW is significant for the design of Internet-based learning systems because HTML acts as the glue that holds the elements of the system together and links related information. More importantly, the Internet has become increasingly accessible to people all over the world. According to the Associated Press, there are 74 million Internet users in the United States (Gearan, 1999). The U.S. Department of Education (1999) reports that, in 1998, 89% of public schools have access to the Internet and 51% of all American classrooms are connected to the Internet. As a result, the WWW not only serves as an ideal environment to implement learning systems with the use of latest information technology, but also enables students to have round the clock access to educational systems from any location.

2.1.2 Client-Server Systems

A client-server system is the most used method of organizing software within computers connected to the Internet. A client-server system is basically a system where components called servers have resources and information that other components called clients wish to access. Clients connect to a server to obtain the desired resources or information. The significance of client-server systems is that they work with the Internet

in a synergistic fashion thus allowing clients and servers to be geographically dispersed. Many of the elements of Internet-based learning systems are best designed using the client-server paradigm. Client-server systems have become so pervasive that much of the current Internet technology is classified as client technology or server technology. Client technology tools are significant for learners because they are the tools that learners are expected to acquire, buy and use. Server technology tools are important for designers and institutions providing the learning environment because they support the infrastructure needed to effect the learning systems.

2.1.3 Object-Oriented Software

Object-oriented (OO) systems represent a shift from the procedural way of programming and they constitute one of the most powerful and recent paradigms of complex software design and implementation. With OO, code and data are organized into objects that conceptually represent the behavior and state of a phenomenon in the problem domain. The focus is on building programs that correspond to the models of the problem domain and hence the final program mirrors the domain. Educational software designers can get so immersed in details that it becomes extremely difficult to develop adequate larger modules of instruction. With OO methodology, the “details first” orientation in the design and implementation of a software can be reduced. Also the modules developed are easily reusable or modifiable. Several object-oriented environments and languages such as Smalltalk, C++, Visual Basic and Java simplify the design an implementation of all of the elements of Web-based-learning systems using the OO paradigm.

2.1.4 Client-Side User Interface

The user interface is important because it is the way learners interact with the virtual learning environment. Traditional graphical interface technologies are X-windows and Microsoft windows. Recent programming environments facilitate the design and creation of highly interactive user interfaces with multi-media capabilities. For example, Java provides a powerful and user-friendly graphical interface that can be incorporated into WWW sites. These interfaces, which translate into X-windows or Microsoft windows depending upon the platform, serve as a powerful medium for learners to access educational material and resources located anywhere in the world.

2.1.5 Technology Integration

In a progressive technological discipline, each innovation should accumulate as the structures and systems of a functioning whole. For example, an Internet-based learning system can be constructed as an interconnected system of learning tools developed using various technologies. Such an integration of the many innovations of educational research into a single unit can support large-scale reform in teaching practices and lead to changes in students' learning outcomes. As a result a critical challenge for educational technology is integration of tools using an appropriate software and development environment. Such a development environment would enable designers to use applications from different vendors, written in different languages and working under different platforms to be put together in a single package. Not only is an integration technology efficient in terms of code usability, but it also makes the system flexible, scalable and easily maintainable. Distributed component architecture (DCA) is a mechanism for integrating different software tools and it permits assembly of a compound system from stand-alone parts

developed independently by different programmers. Highly interactive and creative learning tools can be designed very easily by just putting together already developed components. DCA shows promise in eliminating barriers such as different computer platforms and different programming languages, and as a result provide a solid platform for designing a compound educational software system that integrates different learning tools.

2.2 Educational Software Systems and their Features

The existing educational software systems can be broadly classified into the following categories:

- Authoring systems
- Collaborative learning systems
- Virtual reality based learning systems
- Assessment systems
- Intelligent tutoring systems

2.2.1 Authoring Systems

Lectures are often complemented by course-support web sites in colleges and universities. Course-support sites are WWW environments offering a variety of features related to course information, communication and course management. The most common features of these web sites are that they

- Provide an environment for on-line course resources (i.e. syllabus, lecture notes, case studies, problems, readings)
- Permit 24-hour access to electronic course material

- Enable self-paced study
- Provide course management tools like electronic grade book and allow tracking of a student's performance.

Developing and maintaining such web sites can be too much work for an instructor. However, some commercially available software systems such as Lotus Learning Space or WebCT simplify the job of designing, developing and maintaining course sites. These systems integrate a database with a WWW server to provide a common environment for the instructor and all the students in the course. Hence authoring systems provide a cost-effective way for developing WWW-based environments that can be easily accessed by learners. The authoring tools in these systems also enable the instructors to easily present the course material online. But it is still the students' responsibility to make use of the resources to learn the concepts taught by the instructor.

2.2.2 Collaborative Learning Systems

Interactivity has a positive influence on a student's learning as it allows a learner to see, handle, verify and understand the work of a colleague (Burg et al., 1999). In classrooms, a student's interaction is limited to that with fellow classmates and the instructor. However, the technology of online communication provides easily accessible interaction tools that enable geographically dispersed people to communicate and exchange ideas. As a result, WWW-based technology has evolved to become the most popular and common way for implementing collaborative or interactive learning systems that allow learners to work with remote learning resources. By employing various

Internet-based technologies, information such as content materials can be easily exchanged between the instructor and students. In addition, an interactive environment can be created for learners to communicate, discuss and find solutions to common problems. The online communication tools that are typically incorporated in collaborative learning systems can be classified into two categories: synchronous and asynchronous. Synchronous tools such as videoconference constantly require learners and instructors to be available at the same time. These tools allow users to send and receive information in real time and are useful for brainstorming activities. But it can be quite costly to run and maintain such a system, as they require additional hardware accessories. Asynchronous tools such as E-mail lists or newsgroups do not require the students and instructors to be available at the same time but enable exchange of materials and ideas with ease and immediacy. However, student's can get overloaded with messages, especially in large class and they might not be motivated to check or post messages.

2.2.3 Virtual Reality Based Learning Systems

Software packages with multimedia presentations and graphical user interfaces serve as very good tutoring tools where visualization of the objects and processes is an intrinsic part of understanding a concept. Significant advances in image processing, graphics, and hardware technology make it possible to design media-rich virtual environments where learners can interact with a simulated view of a process, mechanism or device. The key features of a virtual reality based educational system are that it:

- Captures the learner's focus of attention in a virtual world, which is comprehensive and realistic enough to induce a willing suspension of disbelief.
- Promotes learning and recall through multi-sensory stimulation

- Induces learners to spend more time and concentration on a task by means of well-designed interactive environments (Pimentel & Teixeira, 1993).
- Multiple representations and three-dimensional frames of reference can enhance the meaningfulness of data and provide qualitative insights (Erickson, 1993).

NewtonWorlds, developed for teaching principles of Newtonian mechanics, is an example of a virtual reality based learning system (Dede, R. Loftin, J. and Regian, 1994). Virtual reality interfaces enable learners to gain experiential intuitions and hence they serve as effective learning environments for understanding scientific principles. But compared to other software systems, the cost of implementing and using a virtual reality learning environment is very high due to the need for multi-media technologies and tools.

While the web-based authoring tools and collaborative learning tools are basically course delivery and management tools, virtual reality interfaces use visualization as a pedagogical approach to learning. However, none of these three learning environments address the central objective of the EDT system, which is to automate the drill-and-practice mode of learning. Assessment systems and Intelligent tutoring systems have some traits that are related to the requirements of the EDT.

2.2.4 Assessment Systems

Assessment systems are web-based test authoring and delivery tools that promote the concept of testing as a learning tool. These online assessment systems have some advantages over the conventional system of paper and pencil based tests. Assessment tools reduce the burden of instructors by enabling them to easily generate and administer tests to students. Feedback can be provided immediately to the student after an answer

has been submitted and graded, allowing the students to check their understanding of the course content.

QuestionMark, CAT and CAPA are a few examples of the web-based testing and assessment tools. Traditional objective type tests with true or false or multiple choice questions can be created, administered and immediately graded by these web-based testing packages. These systems also have test analysis functions such as test item difficulty level and can generate statistical reports of the student's performance. The important features of the assessment systems are:

- Easy creation of computerized tests and surveys
- Easy administration of the tests ensuring security
- Fast and accurate correction and analysis of answers

Some of the features of CAPA (Kashy et al., 1995), an assessment tool widely used in science education, are discussed below.

CAPA

CAPA, a system for learning, teaching, assessment and administration, provides students with problem sets, quizzes, or exams. CAPA is a tool, not a curriculum, and as such does not dictate course design, content or goals. With CAPA, an instructor can create problem sets, which include pictures, graphics and tables, with variables that can be randomized and modified for each student. For instance, consider the equation $E = mc^2$, where m is the mass of a body and c is the velocity of light in air, a constant. CAPA can randomly pick a value for m for each student and since c is a constant, the system can calculate the answer to the problem by using the equation. This method of picking different values for variables can be used only for very simple cases involving a single

simple mathematical equation. This unique facility of creating problem sets in CAPA makes it a widely used software for administering tests and exams to a large number of students. Other features of CAPA include instant feedback and relevant hints given to the students via the Internet. The system can quickly grade students' work and provide information on their performance in a test. This would enable them to gauge their level of understanding and give them more time to review unfamiliar concepts. In addition, the system can provide access to relevant resources that would enable the students to identify their mistakes and rectify them. The system also records the students' participation and performance in assignments, quizzes and examinations and these are available on-line to both the instructor and the individual student.

Web-based testing tools are currently being used for administering objective type tests only. Some researchers (Imbean et al., 1990, Kearsley, 1987) have worked on artificial intelligence tools to evaluate subjective tests by comparing the student's answers with a number of model essays previously parsed and analyzed by the computer, but the reliability of the software has not yet been established.

2.2.5 Intelligent Tutoring Systems

An Intelligent Tutoring System (ITS) uses artificial intelligence tools to model and compare a student's knowledge and an instructor's knowledge. The system intervenes with tutorial advice when differences between the two become evident (Kearsley 1987). ITS can be broadly classified into two categories: ability training and teaching fundamental principles.

Most of the existing systems are designed for training certain abilities, e.g., debugging electronic circuits or solving algebraic equations (Brown and Sleeman, 1982). Every ITS has at least two kinds of knowledge that can be clearly separated. One is the domain knowledge that is to be taught to the user. The other is the tutoring knowledge that tells the system how to teach. The domain knowledge of these systems consists mainly of the methods for solving problems in the domain and they are represented in the form of rules. The systems try to match their own actions with the observed user's actions.

For teaching fundamental principles, IT systems must support the user in learning, clarifying and consolidating fundamental knowledge, which consist mainly of the explanation of the concepts and their interrelationships, the presentation of examples, and the definition and refinement of categories (Teege, 1991). The difficulty in designing these "explanation" systems is the representation mechanism for the domain knowledge. This is because the domain knowledge cannot be represented by a set of rules as in the ability training systems. Since the main idea of this system is explanation of a concept to a user novice in the domain, the system should also easily communicate the domain knowledge to the user. Description Logic (designed by Teege, 1994) is one representation method for explanation systems.

Most of the IT systems use one of two approaches to represent the student knowledge. In the first approach, called overlay model, the user's knowledge is simply a subset of the system's domain knowledge (e.g. in WEST (Burton and Brown), 1982). Using this method, the system can represent missing knowledge but not wrong knowledge. The second approach incorporates predetermined typical bugs (e.g. in

DEBUGGY (Burton, 1982)). With this method a good reaction to user misconceptions can be achieved because a special explanation can be associated with every predetermined bug. The drawback is that it is not possible to handle unanticipated user bugs.

2.3 Comparison of the Features of Existing Educational Software Systems with Requirements of the Engineering Design Tutor (EDT)

Table 2.1 provides a summary of the features of different educational software systems and also highlights the features relevant to the requirements of the EDT. The most important requirement of the EDT is an automatic problem generator capable of generating numerous problem sets depending on the instructor's needs. The authoring systems enable an organized presentation of course material and also efficiently manage the course and keep track of students. Collaborative systems are useful for exchange of ideas and information among different learners. Virtual reality learning environments facilitate students to visualize objects, mechanisms and processes. These learning environments are not appropriate for an EDT and the section of the table with highlighted features re-emphasizes this fact. This is because the only way an engineering student can master design concepts is by rote training, i.e. exposure to numerous different kinds of examples and problems. Following the instructor's lectures on design concepts or interaction with other students or 3D visualization of the structural behavior of an engineering system can be extremely helpful to a student, but they can at best act as accessories. Testing a student's knowledge by making him work out problems is the powerful tutoring tool that the instructor of an engineering design course requires.

	Authoring systems (e.g. WebCT)	Collaborative systems	Assessment systems (e.g. CAPA)	Intelligent Tutoring systems	Virtual reality learning systems
Course material or content	X	X	X	X	X
Theory readings, case studies	X				
Course administration and management	X				
Visualization such as animations or simulations					X
Synchronous communication (conferencing etc)		X			
Asynchronous communication (email groups etc)	X	X			
Creating exercises, homework, quizzes			X		
Administering exams or tests	X		X		
Step by step solution procedure				X	
Personalized feedback			X	X	
Keeping track of students performance	X		X		

Table 2.1 Comparison of features among different educational software systems



- Features that are relevant to the design of the EDT

Assessment systems and Intelligent Tutor Systems have some features that can be incorporated in the design of the EDT. Web-based testing systems enable creation of tests

and quizzes, this feature being similar to the requirement of the problem generator of EDT. Intelligent tutoring systems for ability training represent the solution procedure of the problem as a set of rules. This enables the system to identify exactly at what step the student has difficulties and to give guidance appropriately. Engineering design is also based on a set of rules and step by step procedure and hence a similar representation can be used for the solution and tutoring the student.

Existing assessment systems like CAPA or QuestionMark cannot be used in the EDT because of the following limitations. Most of the assessment systems test a student's understanding by making him work out a standard set of related exercises. These exercises or problems are either predefined by the instructor or accessed from a static library/ database. Unfortunately, this testing procedure is not appropriate for engineering design. The EDT requires the problem generator to be able to generate numerous problems, i.e. the problem generation has to be dynamic, based on the user's needs. In CAPA, the instructor can create problem sets by picking values for variables. But a typical engineering design involves a lot of variables and the solution procedure is not a simple equation but a lot more complex. More importantly, the instructor has no control over the problem constructed as the variable values are chosen randomly. In engineering design, the solution procedure varies from problem to problem and a student has to get used to applying the correct design concept depending on the problem. Consequently, the system should allow instructors to set specifications for the kind of problems generated so that they can test the knowledge and understanding of a particular concept. Such a system can make the tutoring adaptive, based on the student's answer. If the student's response is incorrect, the system can guide him towards the solution and then give more

problems to work on, else it can directly jump to problems that test other design concepts. It is with this idea and motivation that the engineering design tutor is designed: to allow instructors to setup problem sets with several problems but at the same time let them control the problems generated by the system by defining what design concept the problem should test the student on.

Chapter 3

Framework of EDT

This chapter provides a general overview of the EDT architecture.

3.1 Routine Engineering Design

As discussed in the background chapter, the existing educational software systems do not address the primary function of the EDT, which is automated problem generation for routine engineering design. In this section, we discuss engineering design in general, describe typical engineering design problems and define their solution procedures.

Engineering design is governed by a set of requirements that are applicable under different situations. These requirements and the conditions under which they are applicable are normally documented in a standard and any practical engineering design has to satisfy the specifications of this standard. For example, a structural design of a building must conform to the LRFD manual of steel construction while a design for a chemical plant boiler must conform to the AICE specifications. To perform a design, an engineer has to come up with a system that is appropriate for the problem situation and also ensure that it satisfies all the requirements prescribed in the standard.

Engineering design is usually carried out as trial and error procedure. Starting with a set of assumptions formed on the basis of design intuition, a trial system is chosen and checked to ensure that it satisfies all requirements of the standard. If none of the requirements are violated, the trial system is deemed an acceptable design. Otherwise some modifications are made to the trial system and the new system is evaluated for compliance with the standard. This procedure can be repeated until a system that is both efficient and economical is obtained. Because of the routine nature of engineering design,

the requirements of the standard can be re-defined into a set of design procedures or rules, which when followed will lead to the design of a system that is guaranteed to be compliant with the specifications of the standard. The design procedures are reformulated from the requirements of the standard and hence, these procedures also have a similar set of applicability conditions. It is these requirements, their applicability conditions, and corresponding design procedures that engineering undergraduate students must master in their design courses.

Because of the trial and error approach to routine engineering design, the design solution produced by two engineers may not necessarily be the same. When starting with different sets of assumptions, each person might develop designs that vary in behavior, yet both satisfy the requirements of the standard. To gain a thorough understanding of the different design procedures and to develop a good design intuition, each engineering student is expected to learn all the requirements of the governing design standard and their applicability. This mastery is then evaluated by the instructor in homework assignments and tests.

3.1.1 Engineering Design Problems

Engineering design problems can be broadly classified into three categories as follows:

- Given a design system and situation in which it is used, the engineer must evaluate whether the system will perform in a satisfactory manner. The solution procedure for this type of problem is to identify a set of applicable requirements in a design standard and check for system compliance with these requirements.

- Given a design, the engineer must determine the range of conditions or situations where the particular system will perform satisfactorily. In this case the solution is obtained by finding the set of requirements satisfied by the system which can then be used to determine the capacity of the system or the limiting conditions under which the system will perform satisfactorily.
- Given a situation, the engineer must design an efficient and economical system. The solution procedure for the design problems is to assume a trial system and then evaluate whether the system would perform satisfactorily. If not, the trial system is modified and evaluated again and this procedure is repeated till a system that performs satisfactorily for the given situation is obtained. This trial and error procedure would reduce to working iteratively on problems of the first type or the second type.

3.1.2 Need for Automated Problem Generation

The majority of practical engineering design involves working with problems of the third type. Because of the trial and error nature of the solution procedure, a number of solutions may be possible for these design problems and therefore every student's design may have to be evaluated independently. As a result, it is a highly time consuming task for the instructor to grade such design problems. Because of time constraints, instructors usually use problems of the first two types called analysis problems to test a student's understanding of specific design concept or a set of design procedures. These problems have a unique solution because the solution procedure is prescribed by the requirements of the standard. Therefore grading these problems is relatively easier compared to a typical design problem. Unfortunately, creating problems of analysis type is very time

consuming. This is because, to set up a problem, the instructor has to come up with a suitable system and an appropriate situation and also evaluate whether the system would perform satisfactorily. Only after evaluating the system will the instructor know what set of design rules are applicable for this problem and whether the system will perform satisfactorily in the problem situation. As a result, it is time consuming for the instructor to author problems that are guaranteed to test the students' understanding of a specific design concept.

Because of the time required to set up and grade engineering design problems, instructors usually use examples from textbooks or from notes prepared earlier to create problem sets. This approach has a number of shortcomings all based on the fact that each instructor will only have a small set of solved problems from which to formulate both homework and exam problems. Specifically, this approach prohibits the instructors from creating customized homework sets that would be tailored to the abilities of each individual student or even to the current class. The cost of creating new problems and their solutions acts as an impediment to the instructor's ability to thoroughly test the students' understanding of the subject matter.

The challenge in designing a computer-based routine engineering design tutor is to develop an automated system capable of on-demand generation of design problems that test a specific design concept and also creating the corresponding solution. If available, such a system would greatly reduce the instructor's time in creating problem sets and grading them. In addition, it would enable the instructor to provide customized homework to the students. Finally, the system can be used by students to augment the problems described in lectures and textbook and to provide an additional mechanism for

testing their understanding of design procedures prior to starting the solution of homework problems or exams.

3.2 Methodology for Automated Problem Generation

For a given situation, the set of design procedures that must be followed to obtain the solution for analysis types of problems has been well established over the years in various engineering domains. The objective of the EDT is to formulate a reverse of this procedure as follows:

Given a design concept that corresponds to a set of design procedures, to generate a problem situation that will test a student's understanding of the specified concept.

The solution procedure in engineering design has multiple steps. Existing software systems like CAPA cannot support engineering design domains because they can generate problems with single-step solution only. The EDT system will be the first computer-based system that can be used for automated problem generation in engineering design domains. In this context, it is important to understand that this thesis uses illustrations from the design of steel members. But as the general principles of any engineering design are the same, the ideas discussed in this thesis are applicable to other fields.

To understand the process of generating engineering design problems, consider the following example of a design problem.

Problem statement: Determine if a W 14 X 68 beam of A242 steel with an unbraced length (L_b) of 5 ft is capable of supporting the maximum moment of (M_{max}) of 500 Kips.

To solve this problem, we need to compute the maximum moment capacity of the beam M_n and check if $M_n > M_{max}$, which is the requirement of the standard. To compute M_n , we need to first check if the following conditions specified in the LRFD manual are true:

$$b_f/2 t_f < 65 / \sqrt{F_y} \text{ (Table B5.1), } h / t_w < 640 / \sqrt{F_y} \text{ (Table B5.1) and } L_b \leq L_p \text{ (section F1.2).}$$

$$b_f/2 t_f - \text{width to thickness ratio of flange of the beam} = 7.0$$

$$F_y - \text{yield stress of steel} = 50 \text{ Ksi}$$

$$h / t_w - \text{height to thickness ratio of web of the beam} = 27.5$$

$$L_p - \text{limiting unbraced length for plastic yielding} = 8.7 \text{ ft.}$$

Evaluating the three conditions gives $b_f/2 t_f = 7.0 < 65 / \sqrt{50}$, $h/t_w = 27.5 < 640 / \sqrt{50}$ and $L_b = 5 \text{ ft} < L_p$.

Since these three conditions are satisfied yielding governs the moment capacity of the beam and M_n is calculated from the formula

$$M_n = F_y * Z_x = 479.2 \text{ ft-Kips (} Z_x \text{ - Section modulus of the beam)}$$

Now the requirement $M_n > M_{max}$ evaluates to false as $M_n = 479.2 \text{ ft-Kips} < M_{max} = 500 \text{ ft-Kips}$. The beam fails due to yielding because the moment capacity of the beam corresponding to yielding is less than the maximum moment due to the load on the beam.

The EDT must be able to solve the reverse of this problem: given that the instructor wants to generate problems that test a student's understanding of the concept of yielding of beams generate a set of problems where the yielding of beams is the controlling failure mode. The process of generating these problems begins with determining that the design rule to calculate the moment capacity of the beam corresponding to yielding is applicable only if the set of conditions $b_f/2 t_f < 65 / \sqrt{F_y}$, $h/t_w < 640 / \sqrt{F_y}$ and $L_b \leq L_p$ are satisfied. This set of constraints on the variables in the domain forms a constraint satisfaction problem (CSP). In this example, the CSP is to find

all the beams which have b_f , t_f , h , t_w and L_p such that $b_f/t_f < 65 / \sqrt{F_y}$, $h/t_w < 640 / \sqrt{F_y}$ and $L_b \leq L_p$ are true. The variables in the domain have a range of feasible values and the solution to the CSP gives a range of values for the variables that satisfy all the applicability constraints. The solution range of the variable values can then be formulated into a problem description.

The main contribution of this thesis is the development of a computer-based architecture that implements the CSP based methodology to automate the generation of routine engineering design problems and their solution. Such a system will be able to generate multiple problems that test a student's understanding of a particular design concept. The working of the system will involve the following steps:

- The EDT system will enable the instructor to specify a section of the domain knowledge as the design concept on which the generated problems should test the students.
- The system will then identify all applicable requirements in the domain knowledge, identify the applicability conditions that must hold for these requirements to be considered and formulate a CSP that would determine that values for the variables in the problem statement.
- The solution for the CSP will produce values of variables in the problem statement that ensure that the solution for the problem must follow the design procedure that includes the concepts specified by the instructor.

3.3 The EDT Architecture

Fig 3.1 provides a general overview of the EDT system. The EDT system supports two types of users, instructor and students. Instructors use the EDT system to

generate customized homework that tests the students on routine engineering design concepts. The students use the system as a problem server to solve as many problems as they want and improve their understanding of the design concepts. Because the drill-and-practice mode of learning is common to several engineering design fields, the EDT is designed as a general framework that can support multiple domains.

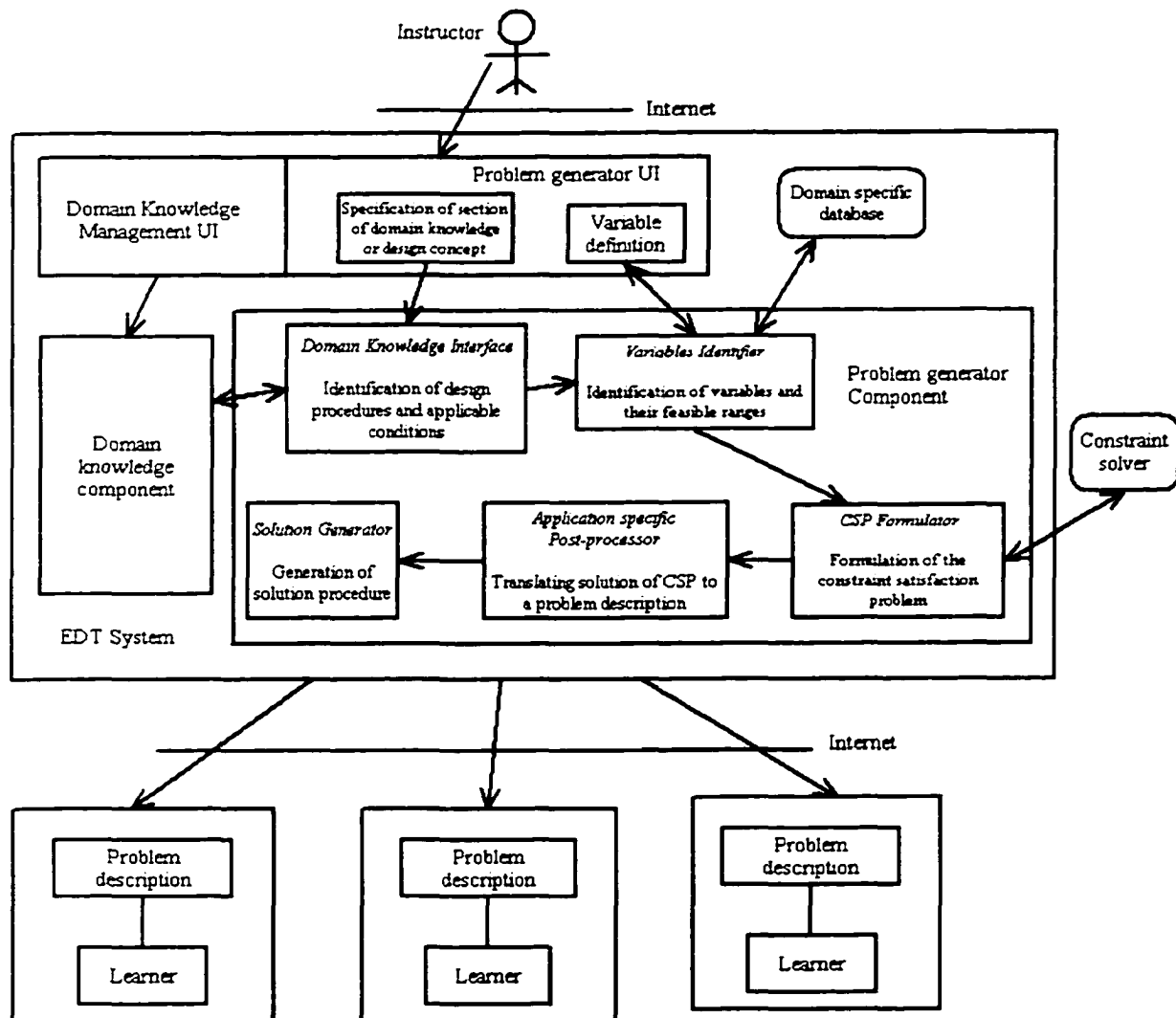


Fig 3.1 Outline of the EDT system

The system uses the Internet as the communication protocol for instructors and learners to access the system. The near ubiquitous availability of the Internet in college

campuses makes it possible to provide unlimited access to a tutoring system from a wide variety of locations. As a result, computer access is no longer an impediment to the student. By using the WWW browsers as the user interface, the tutoring system can be run on a wide range of computing platforms. The Internet also allows the system to be used by students at geographically distributed locations. Hence, the user base for these systems can be extended to encompass a large set of students from various institutions, thus enhancing the ability of the tutoring system developers to recoup the development costs by promoting wider dissemination.

The EDT architecture has two main components: a domain knowledge component and a problem generator component.

3.3.1 Domain Knowledge Component

The domain knowledge component is a module that represents the logic imbedded in the design procedures for a specific domain. The domain knowledge in engineering design is the set of requirements prescribed in a standard and their applicability conditions. The domain knowledge component must provide a representation for this knowledge that is rich enough to capture all aspects of this knowledge yet is easy to author, modify and maintain. Because we intend for the EDT to be applicable to multiple domains the domain knowledge component must support representations that are general enough to be applicable across domains. The domain knowledge component will also have to provide a management interface that would allow the instructor to manage the domain knowledge of a particular field by providing facilities to enter, modify or delete information.

3.3.1 Problem Generator Component

The problem generator component is the module that generates problems based on the instructor's specifications. The problem generator user interface of the EDT serves as a means for the instructor to communicate with the system. The instructor can communicate with the EDT through the user interface to define the specific provision or section of the code as the design concept that the students should be tested on. The generation of problems is done by the following sub components:

- **Domain knowledge interface:** The problem generator component analyses the domain knowledge of the system through the domain knowledge interface and identifies the set of design requirements corresponding to the specified design concept. It also identifies all the applicable conditions that need to be met for the set of design requirements to be applicable. Using the set of requirements and the applicable conditions, this component then generates a set of design procedures, which can be used to determine if a system is compliant with the requirements of the standard. This set of design procedures forms the basis on which any problem (analysis or design type) on the specified design concept can be solved. Since the domain knowledge component uses a generic representation the interface to this component can be implemented in a domain-independent manner.
- **Variable identifier:** The variable identifier component identifies all the variables that participate in the set of design procedures. It also can obtain the feasible range of values for those variables from the instructor or from a domain specific database.

- **CSP formulator:** The set of applicable conditions and the feasible range of values are formulated into a constraint satisfaction problem by the CSP formulator. The solution to the CSP is obtained from a constraint solver.
- **Application specific post-processor:** The problem generator uses an application specific post-processor to translate the CSP solution to a problem description. The solution to the CSP would provide a range of variable values that satisfy the constraints and hence multiple problems can be generated by the system. Formulating a problem description would involve using a standard problem template and plugging in values for different variables from the CSP solution.
- **Solution generator:** The solution generator module generates the step-by-step procedure that a student would have follow to obtain the correct solution for the problem.

The following two chapters describe the representation of the domain knowledge and the working of the problem generator in more detail.

Chapter 4

The Domain Knowledge Component

This chapter discusses the organization and implementation of the Domain Knowledge component of EDT

4.1 Domain Representation

The architecture for EDT separates the knowledge needed for problem generation into two main categories:

- The domain knowledge that captures the design requirements in a domain and
- The problem generation knowledge that operates on the domain knowledge to create design problems and to produce the corresponding solution.

This separation of the overall knowledge into two categories closely mirrors the approach taken in the ITS systems that manage two separate representations for knowledge: domain knowledge and tutoring knowledge. Unlike ITS, the domain knowledge in the EDT is also used by the instructor to control the type of problems generated by the system. This approach simplifies the process of extending and modifying the knowledge in the system and allows the creation of problem generation knowledge that would be independent of the domain to which it is applied.

One of the key issues in the implementation of EDT is the choice of the representation suitable for capturing the design requirements in a number of engineering domains. The domain representation must be simple enough to allow for clear and concise mechanism for authoring and verifying the logic of the domain representation. The domain representation must be also be rich enough to both represent the design

requirements and their applicability in a manner that would allow the problem generator to formulate the CSP.

4.2 SASE Methodology

The domain knowledge in routine engineering design often consists of the requirements and their applicability conditions as documented in a design standard. The Standards, Analysis, Synthesis and Expression (SASE) methodology (Fenves, 1987) has been widely used to represent design standards in Civil Engineering design domains and has the required characteristics of clarity, ease of use, rigor and richness. As a result, it was chosen as the representation for the domain knowledge in the EDT.

The SASE methodology, developed in 1987, provides an objective and rigorous representation of the meaning of a standard. It has been used extensively for evaluation and revision of building code requirements. The methodology embodied in SASE deals exclusively with the logic, format, and organization of standards, that is, their syntax. Because of this emphasis on syntax, SASE is applicable to a wide range of engineering standards and can be used to represent any solution procedure that is based on a set of conditional rules.

The two main elements of the SASE representation are data items and decision tables. The next two sections explain these elements in more detail.

4.2.1 Data Items

A data item or datum is any information element occurring in a standard. Each result or variable generated by evaluating a condition and applying a design rule is a datum. For example, the moment capacity of a beam, calculated by identifying the governing failure mode of the beam, is a datum. In addition, the status of each

requirement of a standard is also a datum. For instance, there is a requirement that the moment capacity of a beam is greater than the applied external moment. The status of this requirement (whether satisfied or violated) is also represented by a variable.

The data items in a standard can be divided into two categories. All variables referred to in a standard but not explicitly assigned a value by a requirement are called input data or basic variables. For example, the Young's modulus of elasticity (E) used in the steel design manual is a known constant for steel and is therefore a basic variable. Derived data items are those that are assigned values or a range of values by provisions or requirements in the code. The logic expressed by a design requirement and its applicable conditions to assign a value to a derived data item is termed as a design rule. Because each requirement has a set of applicability conditions, different design rules may exist for assigning values to a derived data item. The set of design rules with the expression for evaluating data items and the applicable conditions is the information needed to evaluate the compliance of a design with a standard.

4.2.2 Decision Tables

A decision table is a structure that is used to represent the rules for calculating or evaluating a derived data item. It defines a set of rules by specifying certain actions to be executed based on a specific set of conditions and hence are an orderly representation of the reasoning leading to a decision. A decision table is composed of conditions, actions and rules. A condition is a logical statement that may have one of the two values: true or false. An action in a general sense is any operation that assigns a value to a variable, usually by means of a formula.

In the SASE methodology, there are two fundamental principles for the use of decision tables to represent design rules. The first principle is that each decision table establishes the value for only one data item. In other words the only allowable actions are those which establish a value for the data item associated with a decision table. All of the other data items excluding basic variables used in the conditions and actions of the decision table are the *ingredients*. While this principle restricts somewhat the great flexibility of decision tables, it does lead to a desirable consequence: the decision tables thus formed tend to be small and therefore easy to formulate, understand and analyze. The second principle is that, constants, operators and basic variables aside, each condition and action contains only derived data items defined in the standard. This is necessary so that a network can be formed that links the decision tables of all ingredients that are derived data items.

In the graphical representation of a decision table, there are four parts. The top left portion is the condition stub where all the logical conditions that affect the data item are defined. The bottom left portion or the action stub has all the possible actions, each one assigning a different value to the data item. The top right section is the condition entry and the bottom right section is the action entry. Each column in the condition entry defines a rule. The action entry indicates which actions are to be executed for each rule. The rule is a logical AND function, i.e., the rule is not satisfied unless each of the condition entries it contains is matched. It should be ensured that the reasoning represented in the decision table leads to a unique result for the data item in each case and that no possibility exists for encountering an unanticipated situation.

A simple nomenclature is used to define the logic in a decision table. The condition entries can be a “T” or “F” or “I” indicating that the corresponding condition should be true, false or immaterial (can be either true or false). All the conditional entries in a column cannot be immaterial; otherwise, the data item would be a basic variable. For a rule to be applicable all the conditions should be satisfied as per the column entries. An “X” in an action entry indicates which action is to be taken for a given rule. As each rule has to lead to a unique value for the data item, there cannot be more than one “X” in a column. But more than one set of rules can lead to the same action.

		R1	R2	R3
Conditions		D1	D2	D3
C1	$L_b \leq L_p$	F	I	T
C2	$L_b \leq L_r$	T	F	I
Actions				
A1	$M_{nLTB} = C_b * (M_p - ((M_p - M_r) * ((L_b - L_p) / (L_r - L_p))))$	X		
A2	$M_{nLTB} = M_{cr}$		X	
A3	$M_{nLTB} = M_p$			X

Table 4.1 Decision table for evaluating M_{nLTB}

The above decision table is used to determine M_{nLTB} , the moment capacity of the beam corresponding to lateral torsional buckling mode. The table has two conditions, three possible actions and three rules. It is also seen that the table has the variables L_b , L_p , L_r , C_b , M_p , M_r and M_{cr} . Some of these are basic variables while others are the ingredients, i.e., to be derived from other decision tables. The rules in this table are that if

$L_p \leq L_b \leq L_r$, then M_{nLTB} is calculated from the formula in the first action; if $L_b \leq L_p$, then $M_{nLTB} = M_p$; and if $L_b > L_r$, then $M_{nLTB} = M_{cr}$.

4.3 Advantages of SASE Methodology

Decision tables form the core of the representation mechanism. The entire standard and its specifications that form the basis for any engineering design can be represented in this format by using decision tables. Solving design problems involves evaluating the appropriate decision tables in the correct sequence. The decision tables give a clear and elegant representation of the different decision rules to be applied for different situations. Consequently, by representing the rules in the form of decision tables, they can serve as an easy means for the problem generator to generate problems depending on the instructor's needs.

The use of SASE methodology to represent design by decision tables rules ensures the following qualities:

- Complete and correct representation of the documented standard
- The relations among the different provisions of the standards are connected explicitly. Cross-references will show the sequence in which the provisions have to be applied to evaluate data
- The relations are acyclic, i.e., without repetitions or loops in logic.

The EDT system using the SASE methodology for the representation of the design rules will have the following features:

- Maintenance of all information in a database, thus providing facilities to store, analyze, modify and combine standards.

- Convenient user interaction in an elegant and clear format for entry, analysis, modification and display geared to users with varying levels of proficiency.
- Facilities for processing and combining of large standards subdivided into several units.
- Facilities for interfacing with additional capabilities, including text generation.

4.4 Domain Knowledge Management Interface

In the domain knowledge component of the EDT system, relational databases are used to represent decision tables. Relational databases use tables with rows of data entries to store information and are hence appropriate for storage of the decision table structure of the domain knowledge. Each decision table in the domain knowledge is represented by two tables in the database, the condition table and the action table. These two tables together are an equivalent of the graphical representation of the decision table. The condition table contains the list of conditions and the corresponding set of condition entries while the action table has the list of actions and the corresponding action entries. Relational databases enable storage of data in an elegant and easily retrievable format. Hence the problem generator component can interact with the relational database of the domain knowledge for identifying the set of design procedures or rules corresponding to a specified design concept.

The domain knowledge management interface is a user interface that enables instructors to create decision tables and modify them. It serves as the content authoring module of the EDT. The user interface was developed as a set of active server pages (ASP). An ASP is a program written using a scripting language that is mixed in with

HTML. These programs enable creation of an online interface to a database and can be used to tie web pages with the data stored in databases dynamically. As a result, instructors can access the domain knowledge component via the Internet to add or modify data.

To create a new decision table, an instructor first specifies the number of conditions, decisions and actions in the table and then all the conditions, actions and their entries on a web page. The instructor can also specify a domain name to which a decision table belongs. This domain name is used to group all the decision tables in a particular section of the standard together. Then the ASP program performs some consistency checks to ensure that the data entered corresponds to a valid decision table before storing the data in the form of tables in a relational database. The consistency checks that are performed are

- At least one action is specified for each decision or rule
- No two columns of condition entries are the same in a decision or rule
- At least one of the condition entries in a rule is not immaterial

The user interface also allows the instructor to view a decision table or delete it or modify the data in a decision table. Consistency checks are performed every time the *information in a decision table is modified.*

A few screen shots of the user interface are shown below.

Welcome to the EDT Decision Table Viewer

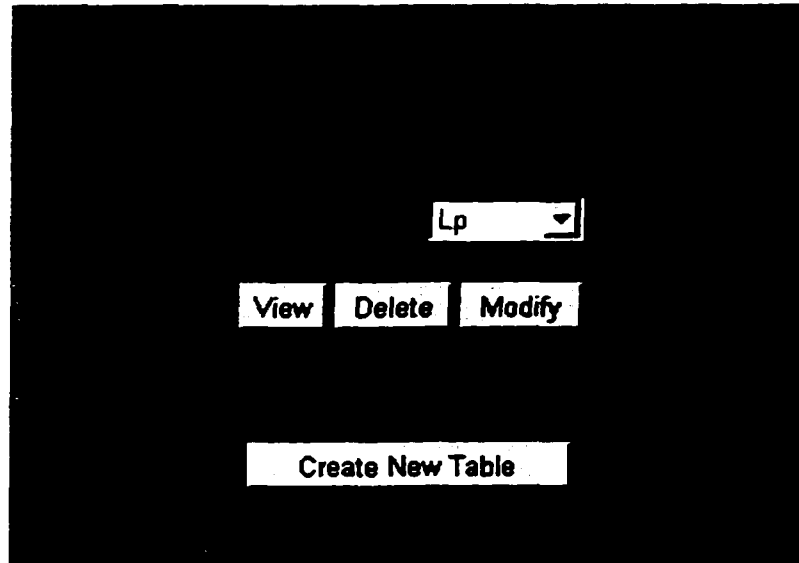


Fig 4.1 Decision table management user interface

Fig 4.1 shows the main user interface that enables instructors to choose an option either to create a new table or to view, delete or modify an existing table.

Viewing table Lp from domain Beam

Ishaped = 1	T	F	I shaped; Built up
$L_p = 300 \cdot r_y / \text{pow}(F_y f, 0.5)$	X		eqn. F1-4
$L_p = 3750 \cdot r_y \cdot \text{pow}((J \cdot A), 0.5) / M_p$	X		eqn. F1-5

Fig 4.2 The decision table viewer of the user interface

Fig 4.2 shows the decision table for L_p in the decision table viewer of the user interface

Creating table Mp in domain Beam

$Fy*Z \leq 1.5*My$	T	F	
$Mp = Fy*Z$	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
$Mp = 1.5*My$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Submit

Fig 4.3 User interface to create a new decision table

Fig 4.3 shows the user interface that can be used to define a new decision table. The conditions, actions and the corresponding entries can be specified.

Chapter 5

Problem Generator

This chapter describes the operations of the problem generator component of the EDT.

5.1 Outline of the Problem Generator

Fig 5.1 provides a graphical representation of the structure of the problem generator component. The task of generating a design problem using EDT involves the following steps:

- First, the instructor specifies the domain in which a student's understanding is to be tested by selecting a specific section of the standard. The instructor then defines the type of problem that will be generated by the EDT by specifying a set of design rules or "testing rules" which must be evaluated as part of a successful solution to a problem
- The domain knowledge interface component of the problem generator will then interact with the domain knowledge component to identify the set of code requirements and their corresponding preconditions that must be met in-order for these requirements to be included in the solution.
- The variable identifier parses the representation of the domain as it defined in the domain knowledge component and identifies all basic and derived variables that are included in the set of identified preconditions. The range of feasible values for all the basic variables can then either be obtained from the instructor or from an external, domain-specific component.

- The set of preconditions and the defined ranges of values for the identified basic variables form a basis of a classic constraint satisfaction problem (CSP). The CSP formulator component formulates the constraint satisfaction problem, formats it and feeds it into an external constraint solver. The external constraint solver is expected to produce the values or range of values for the basic variables that would ensure that the set of preconditions identified by the domain knowledge interface component is satisfied.
- The solution from the constraint solver is then translated into a readable problem description by the application specific post-processor component.
- Finally, the solution generator provides the step-by-step solution to the problem in a readable form.

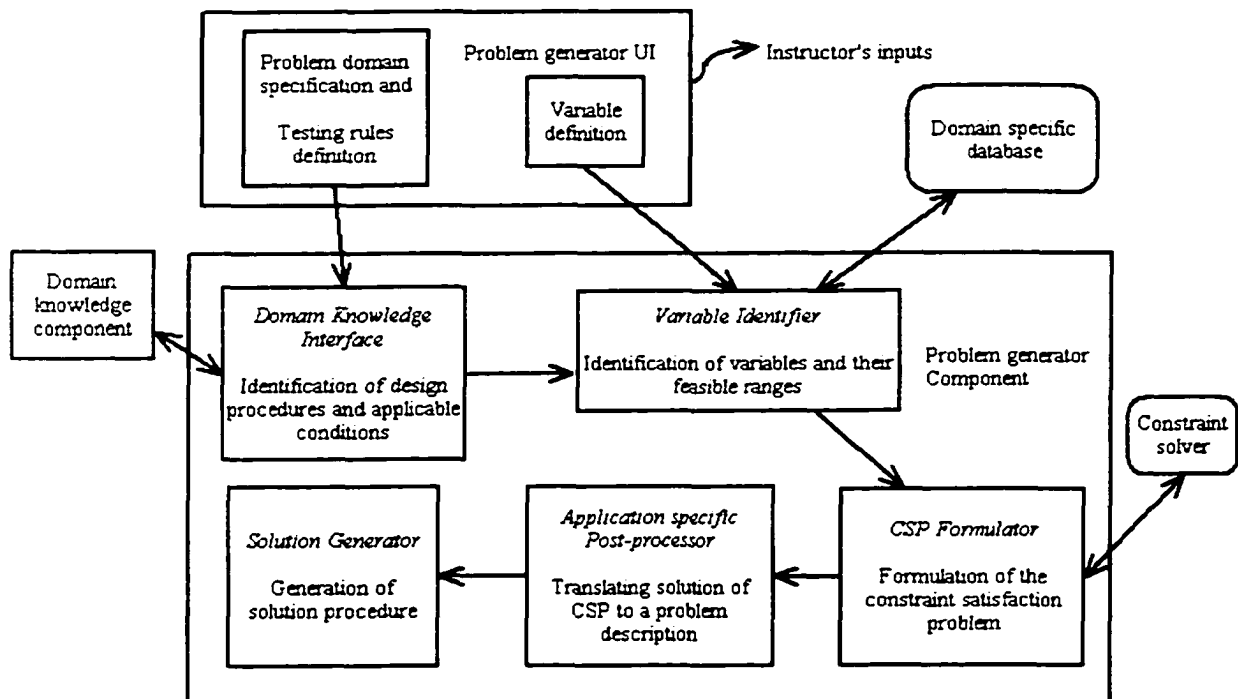


Fig 5.1 Representation of the working of the problem generator

5.2 Definitions

A few terms, related to the use of decision tables to represent design requirements, are defined in the following section.

Table 3.1 shows a decision table to determine M_{nLTB} , the moment capacity of the beam corresponding to lateral torsional buckling (LTB) failure. This decision table is used illustrate the terms defined in this section.

		R1	R2	R3
Conditions		D1	D2	D3
C1	$L_b \leq L_p$	F	I	T
C2	$L_b \leq L_r$	T	F	I
Actions				
A1	$M_{nLTB} = C_b * (M_p - ((M_p - M_r) * ((L_b - L_p) / (L_r - L_p))))$	X		
A2	$M_{nLTB} = M_{cr}$		X	
A3	$M_{nLTB} = M_p$			X

Table 5.1 Decision table for evaluating M_{nLTB}

5.2.1 Decision tree

The logic contained in a decision table can also be expressed as a decision tree, which is a network with one condition at each node. The branches from each node represent the possible condition entries, and the termination of each path, or limb, is a rule. Fig 5.2 gives the example of a decision tree generated for the decision table in Table 5.1. The expression of logic in a decision tree strongly resembles a conventional a binary

tree. In order to evaluate a decision table, a path in the decision tree is traversed until the applicable action is identified and evaluated. Decision trees provide additional clarity and can be of great help for students in trying to understand the logic of the provisions of the standard.

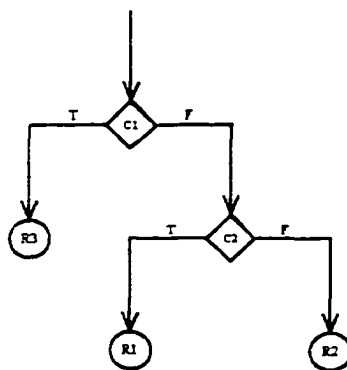


Fig 5.2 Decision tree for the decision table in Table 5.1

5.2.2 Interaction network

While decision trees represent the reasoning in a single decision table, interaction networks are used to represent inter-links and relations among all decision tables used to describe a specific domain. An interaction network is basically a network defined by a set of points called nodes connected by lines called branches. The requirements of an interaction network are that a branch may be connected to only two nodes, one at each end, and that branches may be connected only at nodes. In an interaction network, each node represents a derived data item. Since each decision table establishes the value for data single derived data item item, each node in the network also represents a decision table in the domain. Each node or decision table is in turn a root of a sub-tree consisting of nodes representing all derived data items found in its conditions and actions, i.e., the ingredients. A complete network for a domain can be assembled by interconnecting these sub trees.

Fig 5.3 shows an interaction network that represents a portion of the requirements of the standard in the domain of design of steel beams subject to bending. The global dependence of a node, i.e., all data items that use the derived variable affiliated with that node can be traced out by examining the network. Also all data items needed to evaluate a variable (global ingredients) of a node can be traced out from the network. The structure of the interaction network for a domain has several features that are important to the task of problem generation. Each network will have at least one terminal node. A terminal node is a root node of a network and as such it represents a derived variable that does not participate in any other decision table. In the decision table representation of design standards, these nodes represent the sub divided segments of the standard, where each segment pertains to a particular type of problem. For example, in structural steel design, the terminal nodes may represent beam or column design.

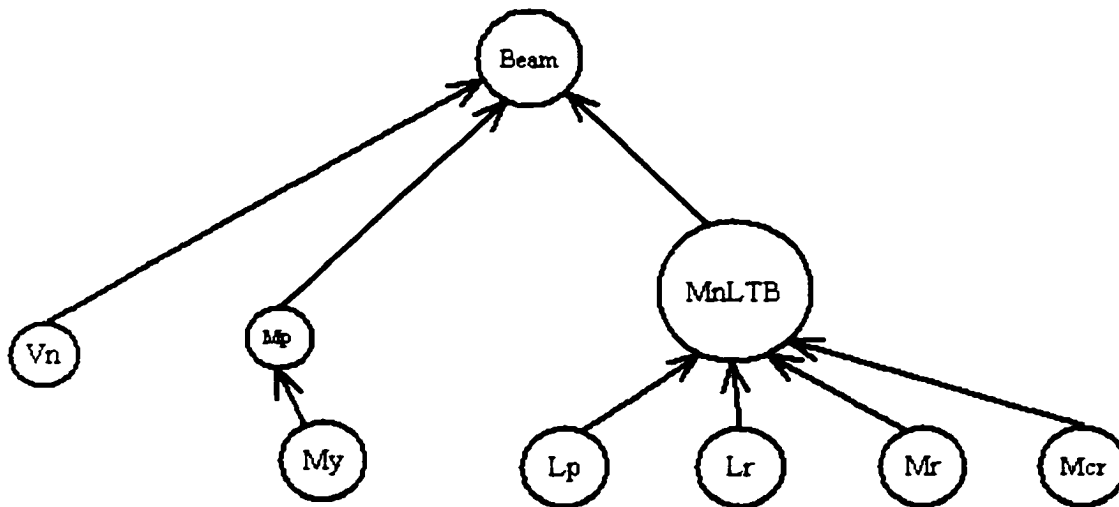


Fig 5.3 Interaction network in the domain steel beam design subject to bending

5.2.3 Problem tree

While an interaction network provides a concise representation of the relations among the decision tables, a problem tree provides a more detailed representation of the inter-relationships between elements of the decision table.

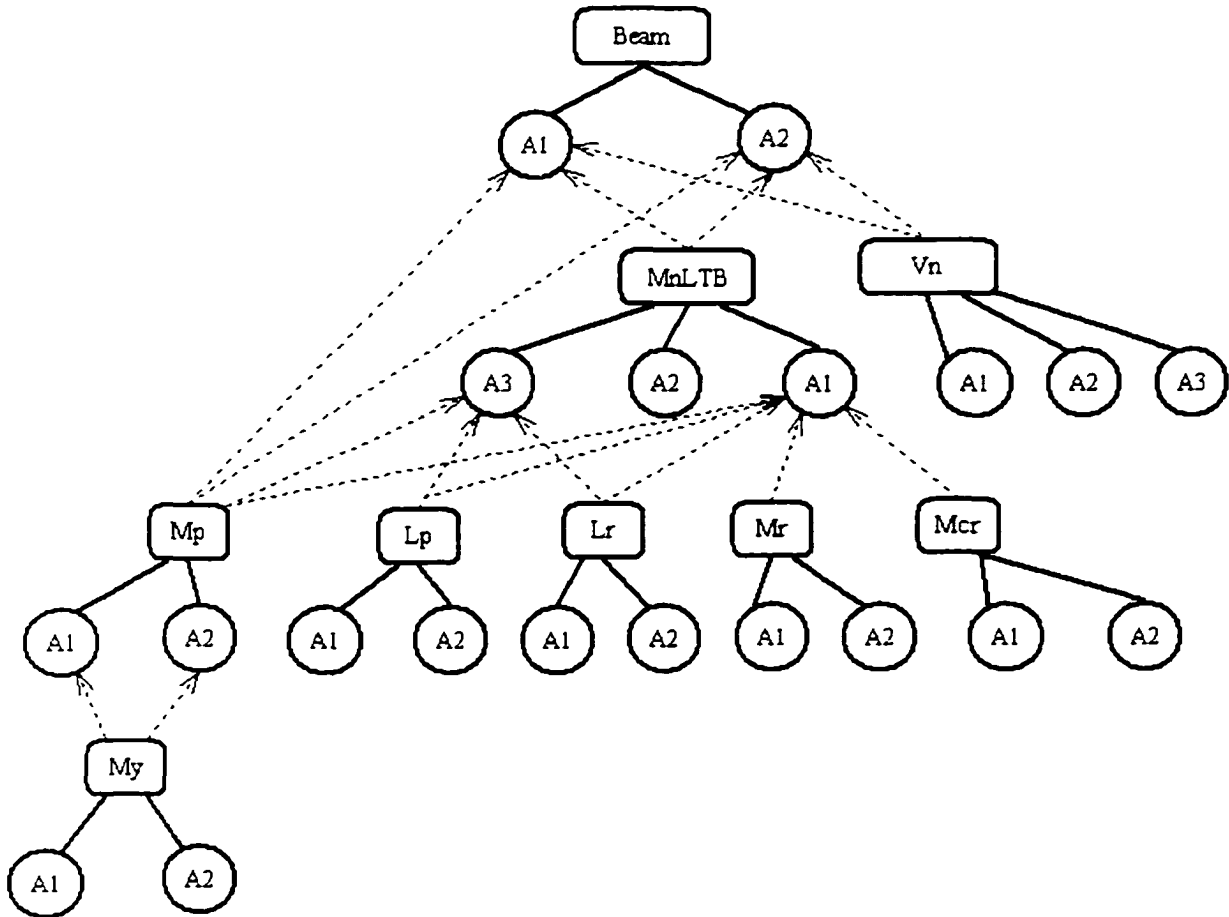


Fig 5.4 Problem tree for the interaction network in Fig 5.3

The problem tree consists of two types of nodes: a derived item node and an action node. Each derived item node has at least one action node associated with it. The total number of action nodes associated with a derived item corresponds to the number of possible actions defined in each table and represents the various ways a value can be

assigned to a derived data item. This relationship between derived items and action nodes is represented with solid lines in Figure 5.4. Since the action nodes represent expressions that may contain other derived items as ingredients the problem tree also shows the dependency between the action nodes and derived data items. These relationships are shown as dashed lines in Figure 5.4. The problem tree expands on the logic in the interaction network by defining the various ways a value can be assigned to a derived data item.

5.2.4 Solution path

A solution path is any path that provides all information needed to assign a value to the root node in a problem tree. A solution path is obtained by starting at one of the action entries of the root node and traversing the branches of that action entry. It is important to understand that while the branches from an action entry node to its ingredients indicate that all ingredients must be evaluated, the branches from the derived data item to the action entry mean that there are multiple ways of assigning a value to the derived data item. Also, derived item nodes, once traversed in a path, do not need to be traversed again. A path is complete when an action node has no ingredients or, more commonly, when all relevant ingredients have been evaluated. Fig 5.4 and 5.5 show two of the possible solution paths from the problem tree in Fig 5.3. Because of the multiple actions associated with most derived data items, a typical problem tree will have many possible solution paths, all of which represent possible design procedures of a design problem.

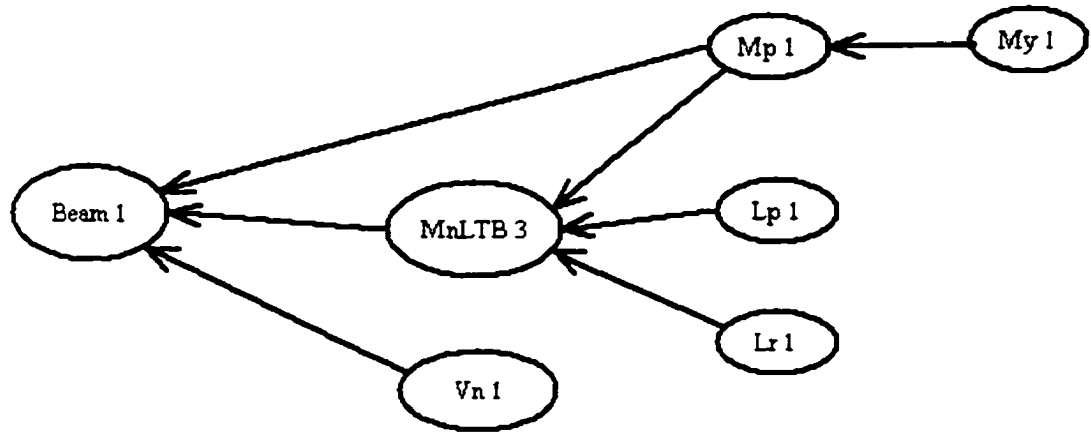


Fig 5.5 A Solution path from the problem tree in Fig 5.4

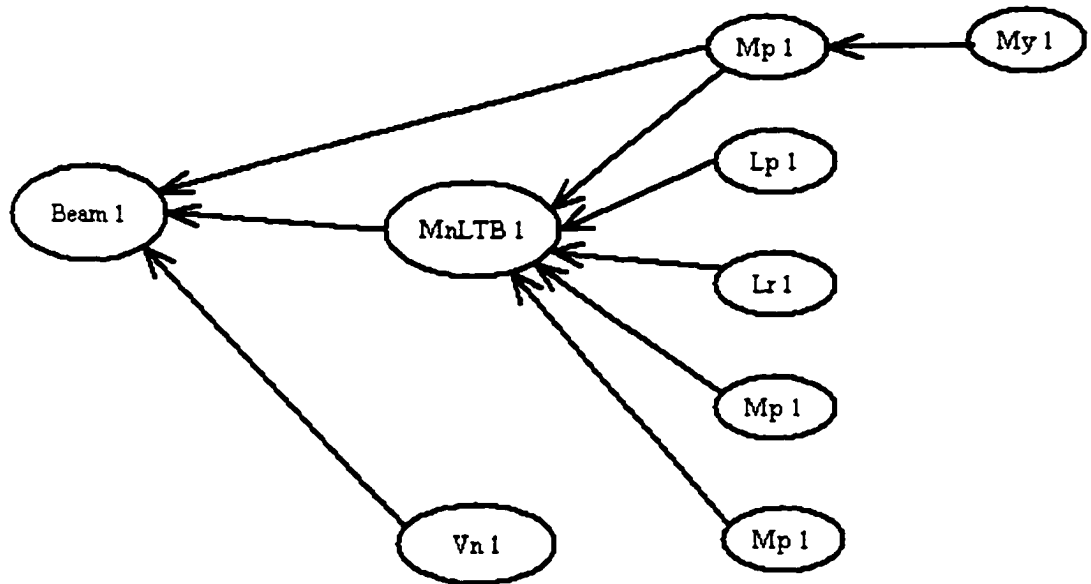


Fig 5.6 Another Solution path for the problem tree in Fig 5.4

5.2.5 Problem path

Multiple solution paths that pass through the nodes of the problem tree exist, each of which represents a different solution procedure and hence corresponds to a different type of problem. To generate a problem, the system has to choose a solution path and

translate that path into an equivalent problem situation. This chosen path is called problem path and is used for the purpose of problem generation.

The SASE methodology not only serves as an ideal representation mechanism for the design rules but also greatly helps the problem generator by facilitating an easy way to identify the problem path which can be translated into problem descriptions that conform to the instructor's requirements or specifications.

5.3 Working of the Problem Generator

A detailed explanation of the instructor's inputs, the formulation of the constraint satisfaction problem and the generation of the problem description are provided in this section.

5.3.1 Problem Generator UI

The instructor's input consists of two parts: the domain definition and the problem definition. The instructor specifies these inputs through the problem generator user interface component of the EDT system.

5.3.1.1 Problem Domain

A problem domain is a computer representation designed to capture and represent all possible variables and decisions in a topic covered by problems in a problem set. We expect each problem set to cover a limited set of topics, as for example, determining the strength of a steel beam subject to bending. In the domain knowledge component, each problem domain is represented by a corresponding set of decision tables. The user interface allows the instructor to specify a problem domain as input, and hence the instructor can choose the section of the standard to test the students.

5.3.1.2 Problem Definition

In preparing homework and exam problems, an instructor will generally begin by identifying concepts that will be tested by these problems. In the case of the routine design problems this can be accomplished by identifying the sections of the code that are critical to the understanding of these concepts. In the case of EDT, the instructor can define a set of problems to be generated by identifying a set of critical decision tables and the action that should be used to assign the value for the derived data items. The user interface to the problem generator allows instructors to select decision tables and actions from the table. Since each table represents a portion of the design standard, each action represents a simple requirement of a standard and the condition stubs represent the applicability conditions for that requirement. Hence, an identified action in a decision table corresponds to a design rule in that table and all the specified actions in the problem definition form the set of testing rules.

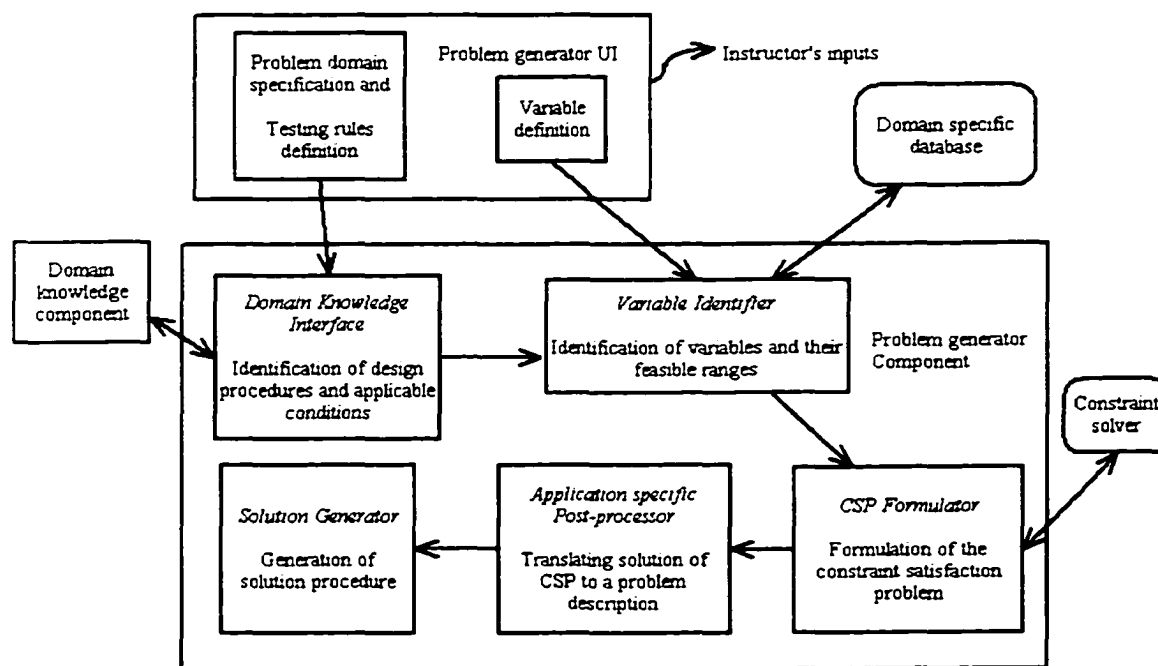


Fig 5.7 The components of the problem generator

5.3.2 Domain Knowledge Interface Component

The Domain Knowledge Interface (DKI) component receives instructor's input as a set of actions and their corresponding design tables. The DKI begins processing by parsing all decision tables in the domain representation and building a corresponding problem tree or trees.

Once the problem tree(s) are created the DKI will try to identify a solution path through the problem tree that includes all actions that were specified by the instructor. If a problem path can not be found, the problem statement is in error and a lack of a valid path would indicate that the set of actions includes two requirements that can not be applicable at the same time. In most cases one or more problems paths would be identified by the DKI. Each of the paths represents a feasible design procedure that would require the student to evaluate the actions defined in the problem statement.

In case where more than one path is identified by the DKI, a decision must be made about selecting the most "appropriate" path. In the current implementation of EDT, the DKI will select the shortest path since it represents the simplest problem that can be generated that still includes all the actions defined in the problem statement. This is accomplished by performing the breadth-first search of the problem tree to find the first path that includes all required actions. The shortest path also will produce the minimum number of constraints to be satisfied and will result in a very general problem. As a result, the instructor can control the types of problems by adding and removing the actions from the input set. A different approach would allow the instructor to choose among the identified solution paths. At this moment we decided to not implement this

approach since it would require the instructor to become familiar with the internal representations and methods used by EDT.

For example, in the design of a beam, there are four possible failure modes: plastic yielding, lateral torsional buckling, flange buckling and web buckling. The beam has a load carrying capacity corresponding to each of the failure modes and for the beam to be safe, each of those moment capacities has to be greater than the moment due to the applied external load. Calculation of the different moment capacities in turn depends on a number of other rules. A beam that fails by plastic yielding is the simplest one to design, as the solution path corresponding to yielding is the shortest compared to other failure modes. However, if an instructor wants to check a student's understanding of the lateral torsion buckling (LTB) failure mode, the problem generator can be used to generate problems in which the beam fails by LTB. In the decision table to calculate the moment capacity of the beam, there will be an action stub that assigns the value of the beam moment capacity to be the one corresponding to LTB. The instructor can specify the corresponding design rule as an input and the problem generator will ensure that all the preconditions for that action are satisfied in the problem generated.

5.3.3 Variable Identifier Component

The Variable Identification Component (VIC) receives its input in the form of the problem path identified by the DKI. The VIC will parse all of the decision tables and action stubs defined in the path to identify a set of variables. The VIC will then allow the instructor to specify values or range of values for the identified basic variables. As we discussed in the previous chapters, the design procedures described by design standards separate all design variables into two categories: basic and derived. The basic variables

have no design logic and must be specified as the input to EDT. Some of basic variables are predefined domain constants and hence can be specified by the instructor for the entire domain. For instance, material constants such as Young's modulus of elasticity are standard values associated with all problems in the domain of steel design and can be specified to the problem generator as part of the domain definition. But not all basic variables are domain constants in which case the variable can be assigned any value from its domain or range of all possible values. For instance, the instructor can define the length of the beam to be l , where l is any practically feasible value for a beam length. It is also possible that the instructor might not want to assign a specific value for a variable, but specify a range. The problem generator will then chose an appropriate value for the variable from the range that is consistent with other requirements of the problem. In this case, the limits of the particular variable are a part of the variable definition and the problem generator will ensure that the variable will have values within the limit. For instance, the instructor can define that the length of a beam to be in the interval $[x,y]$, where x and y are lower and upper limits on the length of the beam. The instructor can also leave some of the variables unspecified in which case, the VIC interacts with a domain specific database to identify values or feasible ranges for those basic variables. For example, basic variables like such as area of cross section of a beam or moment of Inertia can be obtained from a database with properties of different beam sections.

5.3.4 Constraint Satisfaction Problem Formulator

After the instructor has specified the problem definition and possible values for the basic variables, the problem generator will come up with a set of problems in which

the specified testing rules are followed. This step of problem generation based on the testing rules is done by constraint satisfaction. Constraint satisfaction problem is a set of constraints or conditions on different variables. In this case, the selected actions in the problem definition correspond to constraints on the variables or data items in the standard. And by solving for the values of the variables satisfying the constraints, the system can generate a problem that meets the instructors needs. The CSP formulator modifies the set of selected actions and the variable definitions as suitable inputs for the constraint satisfaction problem. All possible solution paths from the problem tree that include the specified testing rules are obtained and the shortest path is identified and used for problem generation. Since, the rules of the standard are connected and have to be applied in a sequence to check a design, the actions specified by the instructor for some data items will lead to constraints on different variables. The decision table representation of the standard is very useful in identifying the constraints on the data items. Each node of the problem path represents action entries in a decision table, which in turn correspond to a column or a decision rule. The rule is defined by the condition entries of the column. The condition entries of the different conditions in a decision table represent conditions or constraints that are to be satisfied for a particular rule to be applicable. For instance, in table 5.1, if A1 is a selected action, then the conditions to be satisfied are $L_b \geq L_p$ and $L_b \leq L_r$.

As a result each node in the problem path has an equivalent list of conditions or constraints to be satisfied all of which put together form a constraint satisfaction problem. In engineering design, these conditions are usually mathematical inequalities on variables. For constraint solving, the system also needs the physically meaningful ranges

or domains for the basic variables. These are either obtained from a standard database or can be given as inputs by the instructor. This set of constraints and the limits for the basic variables are all that are necessary for the constraint solver.

5.3.5 Application Specific Post-Processor

A constraint solver is used to find out the solution of a constraint satisfaction problem. The solution is a set of values of the basic variables that satisfy the constraints on the variables. Having obtained the feasible values or range of values of the variables, the system has to translate those into equivalent problem descriptions. This translation step from the constraint solver solution to the problem description is done by an application specific post-processor. Standard problem templates can be used to formulate a problem statement and the CSP solution can provide values for the variables in the problem. Since the solution consists of range of values for each variable, multiple problem descriptions in which the same set of design rules are applicable can be formulated. A problem description depends on the notation and representation used in the respective engineering fields. For example, in structural beam design problem, if the breadth of the flange, width of the flange, depth of the beam and dimensions of the web are the basic variables used in the constraint satisfaction problem representation, the problem description is stated in terms of the section designations. Each beam designation corresponds to particular values of the depth of beam, the flange and the web dimensions. Hence, the translation of the output of the constraint solver into a meaningful problem description is done depending on the specific field for which the problem generator is implemented. Thus, the problem generator can translate the instructor's input into a set of problems that test the students on specific concepts of engineering design taught in class.

5.3.6 Solution Generator

For each problem generated by the problem generator of the EDT, the instructor would also like to have the correct solution to this problem. This solution procedure is provided by the solution generator component of the problem generator. Checking for an engineering design would mean evaluating a set of relevant decision tables in the correct sequence and verifying that the design complies with requirements of the standard. As a result, the solution representation must consist of all feasible problem-solving steps, the logic describing how to evaluate whether a step is appropriate for a given situation, and all legal sequences of problem solving steps from the problem statement to the final solution. The values for all the variables in a problem description will correspond to a specific solution path in the problem tree. The step-by-step solution procedure can be obtained from this solution path of the problem tree.

Evaluation of the decision table corresponding to the root node of the problem tree provides the solution to the problem. However, this decision table will have derived variables that require other decision tables to be evaluated first. By a recursive backtracking procedure all necessary decision tables can be evaluated with the root node decision table finally being evaluated. This is exactly what students are expected to do when they solve the problems. Knowing which action to be used under which condition is all that the student needs to understand in engineering design. When all decision tables are evaluated, a path that passes through all relevant action entries of decision tables emerges from the problem tree which represent the solution procedure for the particular design problem.

Chapter 6

Constraint Satisfaction

Constraint solving is an important computational step in the problem generation. This chapter discusses constraint satisfaction in general, describes various techniques of constraint solving, and identifies the type of solver used in by EDT

6.1 Constraint Satisfaction Problem

Constraint Satisfaction problems or CSPs provide a general representation and solution model to a large class of non-linear problems. A CSP is formulated as set of requirements or constraints on a set of variables and constraint solving involves finding values for the variable such that the constraints are not violated. Constraint programming is an emergent software technology for declarative description and effective solving of large constraint satisfaction problems especially in areas of planning and scheduling.

Constraint programming has been successfully applied in numerous domains. Recent applications include

- Computer graphics (to express geometric coherence in the case of scene analysis)
- Natural language processing (construction of efficient parsers)
- Database systems (to ensure and/or restore consistency of the data)
- Operations research problems (like optimization problems)
- Molecular biology (DNA sequencing)

The constituent parts of a CSP definition are:

1. A finite set of variables
2. A domain, for each variable
3. A finite set of constraints

Each variable can be assigned any of the values in its associated domain. However, each constraint in the problem puts a restriction on the values that a group of variables can take in combination. An assignment of a value to a variable is known as a binding or instantiation. The instantiation of a variable v to a value a is denoted a/v . If the variable $X1$ has a domain $\{a,b,d\}$ then possible instantiations are $a/X1$, $b/X1$ and $d/X1$. A set of variable instantiations is known as a labeling.

A labeling is often represented as a tuple: $\langle a/X1, b/X2, f/X3 \rangle$

Constraints arise naturally in most areas of human endeavor and are a medium for people to express problem in many fields. They are typically mathematical expressions or inequalities. For instance, the three angles of a triangle sum to 180 degrees, the sum of the currents floating into a node must equal zero. Constraints enjoy several interesting properties:

- Constraints may specify *partial* information, i.e., constraint need not uniquely specify the values of its variables
- Constraints are *non-directional*, typically a constraint on (say) two variables X , Y can be used to infer a constraint on X given a constraint on Y and vice versa
- Constraints are *declarative*, i.e., they specify what relationship must hold without specifying a computational procedure to enforce that relationship

- Constraints are *additive*, i.e., the order of imposition of constraints does not matter, all that matters at the end is that the conjunction of constraints is in effect
- Constraints are *rarely independent*, typically constraints in the constraint store share variables.

A constraint that can be explicitly enumerated is commonly represented by a set of possible legal substitutions and hence a substitution S will meet a constraint C only if S is a subset of C . For example a constraint C for variables $X1$ and $X2$ may be denoted as $C(X1,X2) = \{ \{a,b\} \{d,f\} \}$

For $X1$ and $X2$ to have legal bindings according to this constraint either $a/X1$ and $b/X2$ or $d/X1$ and $f/X2$. A substitution that includes all the variables in the problem is in the set of solutions if it doesn't violate any constraints.

When considering solving CSPs, two variations must be considered. These variations are

1. Finding *a* solution to the CSP and
2. Finding *all* solutions to the CSP

Many of the algorithms used to solve CSPs are more suited to one or other of the variations.

6.2 Algorithms for Constraint Solving

The algorithms for constraint solving can be classified into two types: search algorithms and constraint logic algorithms (Torrens, 1997). The features and the applicability of these algorithm types to the task of problem generation for routine engineering design are discussed in this section.

6.2.1 Search Algorithms

One common algorithm used to solve a CSP is the generic Backtracking (BT) algorithm. This simple algorithm tries to instantiate each variable, and consistency checks are done for each instantiation. If all checks succeed, the next variable is instantiated, otherwise another instantiation with the next value is done. If all possible instantiations for one variable fail then a backtracking action is done to the most recently instantiated variable. When the last variable is instantiated with success, then a solution has been found. Many algorithms derived from Backtracking exist for solving CSPs. For example, Backjumping, Conflict-Directed Backjumping (CBJ), Graph-Based Backjumping (GBJ) and Backmarking algorithm (BM) have more sophisticated algorithms compared to the BT algorithm. However, it is necessary to manage additional data structures in these algorithms and hence the overhead costs of are greater. FC algorithm differs from all algorithms described before because it performs the consistency checks forward. FC is very efficient because of its ability to discover inconsistencies early. However, FC sometimes performs more consistency checks than backward algorithms.

The backtracking algorithm and its variation have several drawbacks that make them a poor choice to solve the CSP generated by the EDT. BT algorithm and all others based on it are designed to finding *a* solution to the problem while the EDT is interested in finding *all* possible solutions. This is important distinction since one goal of the EDT is to allow the instructor to generate a number of problems for each set of inputs. Another important assumption of BT is that the domain of each variable is a finite set of all possible values. In the case of design problems, the data items are usually real valued and the domain of these variable is an infinite set of continuous values. While it possible to

transform the normally real valued to a finite set by discretizing the interval of the domain and assigning specific values within the interval for the data items. Unfortunately, this transformation would make it very expensive to use the backtracking method or its variations to find all the solutions to a constraint satisfaction problem because of the increase in the number of instantiations and consistency checks would make the algorithm inefficient. Because of the requirement that the problem generator identify all solutions to the CPS and the nature of the domain of the variables used in design problems the backtracking algorithm is ill suited for use in EDT.

6.2.2 Constraint Logic Algorithms

Analytic Constraint Logic Programming (ACLP) is an alternative to search algorithm that is more suitable to the requirements of the EDT. This procedure uses an interval based constraint language to define scientifically interesting constraints on real numbers. An interval arithmetic constraint solver is an algorithm which takes an arithmetic constraint $C(X)$ on a tuple X of variables and returns an interval I of intervals such that every solution X to C is contained in I . This algorithm uses algebraic and numeric methods instead of combinations and search. Also the individual constraints can be more complicated, for e.g., nonlinear inequalities. This algorithm is implemented by using interval arithmetic operations iteratively for each constraint C to contract the intervals of the variables. Thus, the solver narrows the real intervals associated with the variables without removing any solution. Consequently, this algorithm has two features that are important to the task of problem generation. It ensures that all the possible

solutions for the constraint problem are obtained. In addition, the algorithm works with an infinite set for the domain of the variables.

Constraint logic algorithm compute solutions to constraint problems by contracting the domain or interval of the variables. Each of the variables is regarded as an unknown real and hence associated with an interval containing all values of this real that might occur in a solution. The initial intervals of all the variables are large enough to contain all solutions of interest. These intervals are then reduced to sub intervals such that values that are inconsistent with the constraints are removed.

Consider for example, the constraints of the form $u + v = w$. Assume that the intervals for u , v and w are $[0,2]$, $[0,2]$ and $[3,5]$ respectively. Then all three intervals contain inconsistent values. For instance, $v \leq 2$ and $w \geq 3$ implies that $u = w - v \geq 1$. Hence the values less than 1 for u are inconsistent (can no way be part of a solution). Similar considerations rule out values less than 1 for v and values greater than 4 for w . Removing all inconsistent values from the a-priori given intervals leaves the intervals $[1,2]$ for u and v and $[3,4]$ for w . This process of obtaining new intervals from old ones is referred to as applying a *constraint contraction operator*. The new intervals are computed by using the rules of interval arithmetic.

Constraint contraction can be done for primitive constraints only. Typical primitive constraints are:

$$x + y = z, x * y = z, x^n = y \text{ (} n \text{ is an integer), } x = y, x \leq y.$$

Complex constraints in a constraint problem should first be translated into equivalent primitive constraints. Only then narrowing of intervals by contraction can be done to

obtain the solution. For example the constraint $x^2 + y^2 = 1$ can be translated by introducing auxiliary variables x_2 and y_2 :

$$x^2 = x_2, y^2 = y_2, x_2 + y_2 = 1.$$

Typically constraints share variables and hence contraction has to be performed multiple times on any given constraint: every time another constraint causes the interval for a variable to contract, all constraints containing that variable have to have their contraction operators applied again. Because changes are always contractions and interval bounds are floating-point numbers, a finite number of contractions suffices to reach a state where all constraints yield a null contraction. A constraint logic algorithm terminates when this is found to be the case. Also contraction operators in interval constraints only remove inconsistent values. Hence, results in interval constraints have the following meaning: if a solution exists, then it is in the intervals found by the solver.

Several researchers have developed programs for constraint solvers that use the principle of interval constraints. The way the contraction operators are defined for primitive constraints is what differentiates these programs. Research work in interval constraints shows that the contraction method is widely applicable for solving practical constraint problems (E. Davis, 1987). The CHIP programming language (M. Dincbas et al, 1988) and BNR prolog (BNR, 1988) are a few front-end programming languages that use the principle of contraction of intervals for the purpose of constraint solving.

Given the advantages of using ACLP, an interval arithmetic based solver was found to be the most suitable algorithm for problem generation task of EDT. However, this algorithm has a limitation in that the narrowed interval of the variables is obtained by contracting the limits or intervals of the variables. The ACLP algorithm operates by

identifying the intervals, which do not contain any feasible values. As a result, given the initial interval for a variable of $[a,b]$, the ACLP will produce a solution set is $[c,d]$, that guarantees that the constraints can never be satisfied if the variable takes any value in the intervals $[a,c]$ or $[d,b]$. One of the characteristics of this process is that the solution interval may also contain values where one or more constraints may be violated.

In the case of the EDT, this issue may result in a system assigning the basic variables values that may violate a constraint generated from the problem path. To guard for this situation, the problem generator requires the instructor to discretize the solution intervals for all continuous variables and then will check the constraints for each of the possible values. If a set of values is found to violate one or more constraints it will not be used in creating a problem description. This approach adds additional cost to the ACLP algorithm but in most situations this cost will be significantly less than that of the backtracking algorithms.

Chapter 7

Software Architecture of the EDT

This chapter describes the software architecture used in the implementation of the EDT.

7.1 Distributed Component Architecture

As described in the previous chapters, the EDT architecture is organized as multiple components that work together as an automated problem and solution generator for routine engineering design. Some of these components are application specific while others are domain-independent problem generation modules. Because of the multiple component structure inherent in the EDT, it is appropriate to use a software architecture that reflects this same philosophy.

One of the requirements for EDT is the ease of implementation of the domain-specific system and the ability to use the same architecture for problem generation in a variety of fields. In Application Island architecture the software is organized into programs that can run independently with their own collection of resources like windows, files and databases. This architecture provides a weak mechanism for building large-scale applications or integrating independent applications. Distributed component architecture (DCA) is a model that enables software to be developed as an assembly of several components. A system built using this model would have great flexibility and would be able to satisfy the requirements for ease of implementation and extension. This chapter discusses the advantages of this architecture, the design of EDT using the DCA model and its implementation for the domain of structural steel design.

DCA serves as a model for integrating multiple software components developed independently into a single package. It can be used effectively for a client server system,

with each of the components being either a client or server or both. In DCA, the operations that can be performed on a server and the parameters that a client needs to pass are defined precisely and unambiguously by an interface. These interfaces yield easily maintainable and re-usable components. The on-line reservation system, for example, could easily be built to use the same reservation database component for several applications -- on-line Web reservations, more complex reservation applications for agents, applications using reservation data to plan resource allocation, and so on. The runaway success of the Internet means that the different components in DCA can exist anywhere in the network and can be accessed by any user. As a result, the architecture not only allows the software to be extended to meet diverse needs but also encapsulates and coordinates the complexity inherent in modern software systems. The price one has to pay for having such a structure is the communication between the different components. For example, if a program wants to use a solver, which is a separate component existing elsewhere on the network, then it has to communicate the necessary data to the solver and get back the results after the solution has been calculated. This communication can be quite expensive and hence has to be minimized so that the advantages of distributed component architecture are utilized in an efficient way.

7.3 DCA Models

Component-based software development borrows its foundations from the object oriented programming paradigm. With the advent of Object Oriented programming, software development is being done in a modularized manner. In OO, code and data are organized into objects that conceptually represent the behavior and state of a phenomenon in the problem domain. Thus, objects in the different modules of a program

encapsulate different functionality. The central notion of DCA is an extension of this. A component is a piece of software encapsulating some service, which can be dynamically attached to different programs at runtime. All that is visible of a component from the outside is its interface and must be wired correctly to make it co-operate with the program. The bottom line of this is that a component is like a black box: the programmer need not know the details of a component's implementation to use it. Because one component needs to use functionality of another, the interaction between the different components of a DCA model is the one that becomes an important issue. As a programmer, this communication protocol of the model being used has to be understood. The different distributed models that are discussed in this section are JAVA RMI, CORBA and DCOM.

CORBA is a distributed component model whose specification is defined by the Object Management Group (OMG). Central to the idea of CORBA is the ORB, which acts as a middleman between clients of a component and the component's implementation. IIOP (Internet Inter ORB Protocol) is the ORB transport protocol, which enables network objects from multiple CORBA-compliant ORB's to inter-operate transparently over TCP/IP. IIOP is a standard protocol whose use guarantees that CORBA-enabled objects can communicate across CORBA runtimes from any vendor. Java has its own, built-in native ORB, called RMI (Remote Method Invocation). You can use RMI to connect to Java components running on different JAVA virtual machines. A client can call a remote object in a server, and that server can also be a client of other remote objects. DCOM is Microsoft's equivalent of CORBA and is specific to Windows

platforms. All three models have a similar working pattern but their structures are slightly different.

Since, components are external to the host application, they need some kind of container in which they can reside. These containers are called libraries or object servers in COM. Libraries can be standalone executables or binary collections, known as Dynamic Link Libraries or DLL's. In JAVA, the classes corresponding to a component can be stored in a jar file, which acts as a container. In CORBA, the individual components can be binaries or executables written in any language and they can be grouped together in a runtime library.

A component's structure is defined by the interface, which it implements. The interface defines all the services that a component provides to a client. In CORBA, IDL (Interface Definition Language) is used for describing the methods supported by an interface. Any object server that implements the interface (defined by IDL) promises to have implementation of the methods defined in the interface. An equivalent of this called Microsoft IDL is used in DCOM for Windows platforms. In JAVA, interfaces are part of the language. Hence any interface that extends the *remote* interface, available in the JAVA API, can be used in RMI implementations.

In order to connect to a component that implements a particular interface, all components and their relevant information should be stored in an accessible location. CORBA uses a database called Interface Repository (IR) to manage this information. The IR has all the components ID's and the interfaces they support. The ORB or Object Request Broker forwards a client's request to an appropriate server. The client that needs to use a CORBA component sends a request to the ORB that contains either the server ID

or the name of the interface. The ORB uses the IR to identify the component that matches this information and returns to the client a reference to an object supporting the requested interface. The client can then use this reference to invoke any methods defined in this interface. In JAVA RMI, the database with the server's information is called RMI registry and is present on each machine where a server is running. A client can obtain a reference to a server component by using the IP address of the server to query the RMI registry running on the same machine as the server component. A similar system database exists in Windows and is used by DCOM.

In a component centric world, method calls becomes an issue because when methods are called on remote server objects, data has to be passed from the client to the server and vice versa. All three DCA models do data transfer by a process called marshalling. The parameters of the method call are bundled, then sent across the network, and then de-bundled on the receiving side. This process is entirely transparent to the client side, which knows only of the interface and the functions it can call on it. In order to obtain this kind of transparency, special code for packing up and restoring parameters is needed. On the server side, there is a component, usually known as a stub. The stub will receive the remote method call packets from across the network and translate them into method calls, which it will invoke on the running server. Any response from the server will be sent back to the client using the same mechanism. On the client side another corresponding component called as proxy will translate the method calls on the interface into packets and ship them across the network to the server stub. Furthermore, the proxy will translate any response from the server into the correct data type or structure.

All of the DCA models have similar features and working strategy for communication across the different components, but they differ in specifications and implementation details for each of the features. As a result, the design of a distributed component system is independent of the DCA model. Furthermore, a system implemented in one model can easily be modified to another model in quick time without affecting the structure of the system.

7.2 Advantages of DCA

Using DCA, a decentralized system with open standards that can scaffold a broad scale implementation can be set up. It ensures that the resulting software system is flexible, scalable and portable.

Flexibility is inherent in DCA systems because the system is a collection of different components. Any of the components can be replaced by an equivalent application that provides the same functionality. Thus an EDT system designed for steel member design can be easily modified for another engineering field just by changing the appropriate components specific to the design field. DCA also makes the system scalable because new components can be added to a working project without affecting the existing system. Finally, DCA supports the integration of components programmed in different languages for different operating systems and ensures that the resulting system can have its components distributed over a variety of platforms. Thus the problem of portability or dependence on platform is also eliminated.

The important advantages of DCA are

- Representing a software as an integration of individual standalone programs
- Reusing a component for different purposes in different applications
- Sharing interface resources such as windows and menus among components
- Inter operating with internet services
- Linking and updating data dynamically among components

These benefits of the DCA make it an ideal architecture to implement a general EDT framework that can then be modified and extended for a variety of engineering design domains.

7.3 Distributed Component Architecture of the EDT

The EDT is structured as a client-server system with different functional modules and hence it can be built using a distributed component architecture. Fig 7.1 shows the distributed software components of the system. The database server and the problem generator are the two main servers in the system. The user interface is the program the instructor uses to generate problems and it communicates to the servers as a client. The problem generator is the server that actually generates the problem description along with the solution. It can also be modeled as a collection of components that are integrated using the DCA. The database server is a component that interacts with two databases, the domain knowledge database and a domain specific database. The domain knowledge interface component of the problem generator uses the database server to obtain the decision table representation of the domain that it used in building its internal representation and set of applicable conditions. The variable identifier component also

uses the database server to obtain the feasible range of values for the variables from the domain specific database. The constraint solver is the server component that is used by the problem generator for constraint satisfaction.

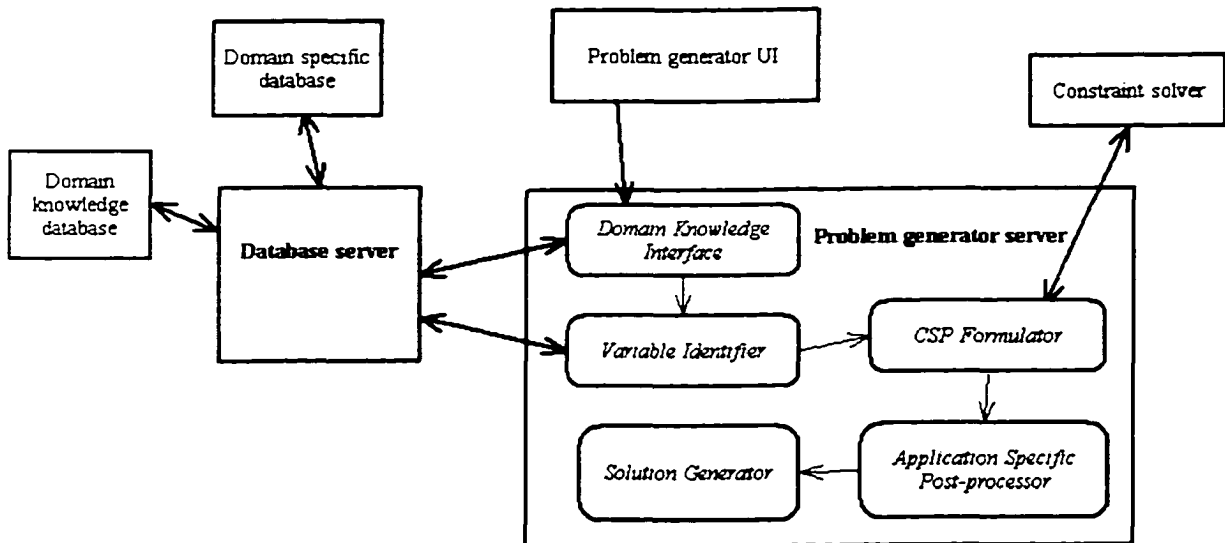


Fig 7.1 Software architecture of the EDT

The decision table representation of the standard in a database, the database and problem generator server, the user interface program, and the interval arithmetic solver are all implemented as components that can be distributed across different computer systems.

This structure makes the EDT software scaleable, flexible and reusable for different needs. The system can easily support different engineering domains since only the domain knowledge, the domain specific database and the application specific post-processor module of the problem generator server need to be modified. The structure of the DCA allows only these components to be modified while all other components remain unaffected. In addition, if for a particular engineering field interval constraints

cannot be used, then an appropriate constraint solver can replace that component without affecting the architecture of the system.

7.4 EDT Implementation

EDT is implemented using the RMI model of DCA. The DCOM was eliminated as choice for the implementation framework since DCOM is applicable only on platforms running Windows operating system and thus would limit the availability of the resulting system.

We decided to implement the entire EDT system in Java since it provides the ability to build object-based, modular and platform independent systems and also enables a close integration with the browser. RMI, in essence, is an extension to the JAVA language. Not only are the clients and (transient) servers written in the Java language, but the Java language is also used as the IDL. RMI depends on many of these features of Java such as object serialization, portability, downloadable object implementations, and Java interface definitions. Thus, the resulting mechanism is very natural for Java programmers to use. They never have to leave the Java programming environment or learn any new "foreign" technology. Hence, RMI gives you a platform to expand Java into any part your system in an incremental fashion, adding new Java servers and clients when it makes sense. As you add Java, its full benefits:- no porting, low maintenance costs, and a safe, secure environment flow through all the components in your system.

While Java RMI is a programming technology, by contrast, CORBA is an integration technology. CORBA is a DCA system that is designed to be general and provides integration support for components written in a variety of languages. It is

specifically designed to be the glue that binds disparate programming technologies together. It does not exist as a point in the programming space; by design, it occupies the spaces between the peaks representing individual languages. In short, while RMI is a viable option for smaller-scale applications to be written entirely in Java, CORBA provides the foundation for integration of existing objects in different languages.

Using CORBA would provide a more general solution but would be unable to take advantage of Java language features for component integration. Since all of our components were written in Java we selected RMI as the implementation mechanism to produce a more efficient implementation. Though RMI's tight integration with Java makes it impractical for use with objects or applications written in any other language, we could take advantage of RMI – CORBA bridge if needed to add to EDT new components written in different languages.

Chapter 8

The EDT for Steel Member Design

The EDT is a general framework that can support multiple engineering domains. In this chapter the implementation of EDT system for steel member design is discussed. The problem generator had to take into account some field specific issues in its implementation.

8.1 Implementation Methodology

We have already seen that with selected actions and some basic variable values being the input, the problem generator uses constraint solving to construct a problem. Multiple problem paths that pass through the nodes corresponding to the selected actions are obtained from the problem tree and the shortest problem path is used for problem generation. The decision table structure facilitates in finding the conditions that need to be satisfied for a specified action to be applicable. Hence a list of constraints corresponding to the shortest path is easily obtained and then a constraint solver is used to solve the constraint satisfaction problem. Given, the domain or possible values for the basic values for the basic variables, the constraint solver finds out all the ranges of the basic variables, in which the set of constraints is satisfied. The domains for the basic variables are problem dependent and can be obtained from the user or from a standard database.

In most applications, the user would want to have a control on the set of problems generated by specifying a main basic variable. This main variable is a primary variable and changing its value affects the solution range of the other basic variables dramatically. Therefore, using different values of the basic variable will lead to generation of different

problems. Taking this into account, the problem generator allows the user to specify a range for the main variable and the number of intervals. The number of intervals is used to split the solution range of the main variable into discrete values and for each of these values, a set of problems is generated. In steel design, the length of a member such as beam or column is used as the main variable. The instructor can control the problem generation by defining a suitable value or range of values for the length of a member.

Interval Arithmetic solver (IASolver), a constraint logic programming language, (T. J. Hickey et al, 1998) was chosen for constraint solving in the problem generator. The constraint contraction operators defined in this program make the algorithm efficient and optimum for obtaining the solution. They also ensure that the computational cost is minimized. The solver, coded in Java, has a straightforward way of accepting the constraint inputs. In some solvers like Prolog, the constraints have to be given to the program in a specified format using a standard prolog notation. However, in the IASolver the mathematical constraints as well as the domain of the variables can be represented just as a set of Strings. The platform independence of Java enhances the portability of the solver and is an added advantage. A wrapper or an API around the IASolver which takes in the String of constraints and domains and returns the feasible intervals of the variables was all that was necessary to use the solver in the problem generator.

Translating the output of the constraint solver to a problem statement can be application specific. In some cases, just enumerating the values for the basic variables from the constraint solver solution might be an acceptable problem statement. But in other applications, these basic variable values may have to be represented as a problem

statement in a specific way. For instance, in steel design the depth and height of web and flange might be the basic variables whose ranges of values are obtained by solving the CSP. Standard beam sections whose web and flange property values are within the solution range should be used for the formulation of the problem description..

In structural design, the maximum moment M_{max} and maximum shear V_{max} due to the load are two basic variables that control the design of a beam or column member. In a problem description the values for these basic variables are not usually specified directly. On the contrary, a member with a set of loads and boundary conditions is specified from which the maximum moment and shear are calculated by using principles of structural analysis. Using the problem generator, the instructor will be able to get values for M_{max} and V_{max} for a problem description. But translating these values to equivalent member loads is not a one to one process. There may be many combinations of loads and boundary conditions that can lead to the same M_{max} and V_{max} . Hence, the problem generator allows the instructor initially to specify the member, the boundary conditions and the loads acting on it. Then a structural analysis program is used to compute the values of M_{max} and V_{max} corresponding to the specified member and these values are later on used in constraint satisfaction for generating an appropriate problem description. Hence M_{max} and V_{max} are not actually part of the problem description, rather the member with specified loads becomes a part of the problem description. When solving the problem, the student has to first compute M_{max} and V_{max} before using the design rules to solve the problem.

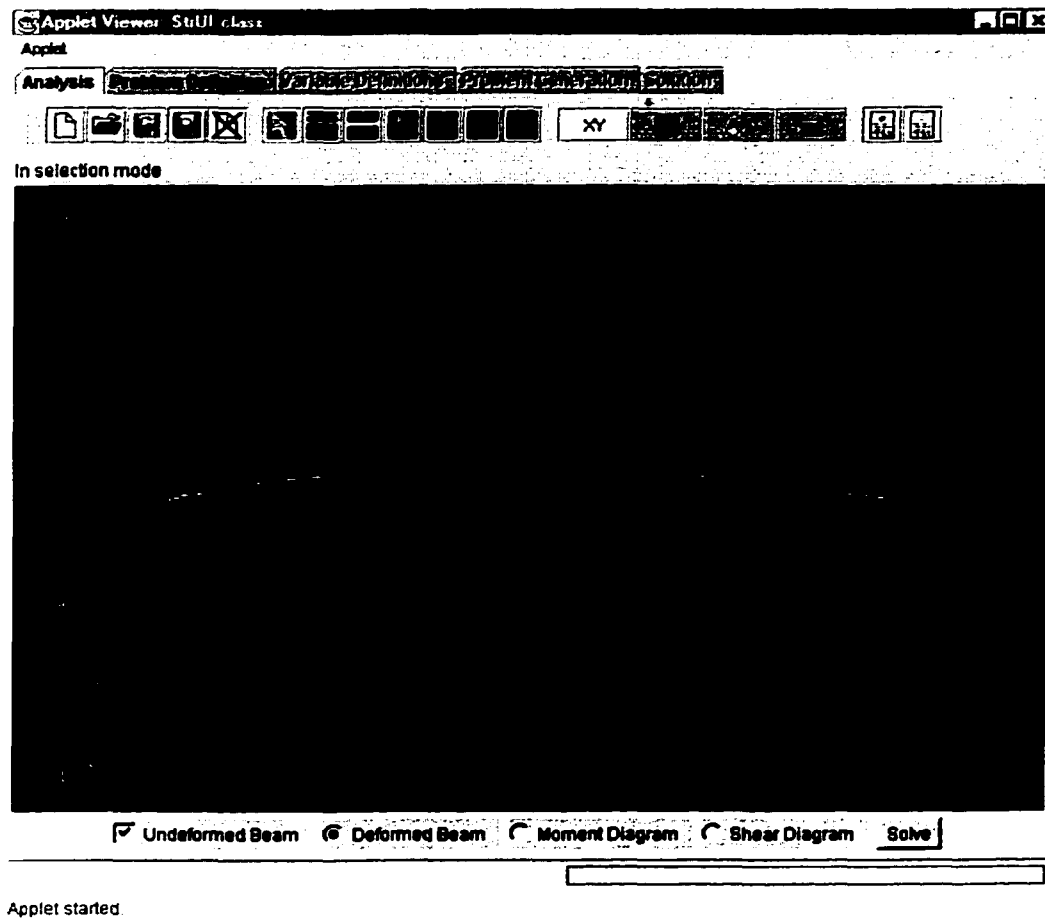
Another application specific problem that could arise is due to the use of the interval arithmetic constraint solver. As explained before, the output of the IA solver may

include a range of values that does not form a part of the solution set. Even in applications where the solution set is continuous, there might be some problems. Every possible combination of values for a variable from the solution set need not necessarily constitute a physically meaningful problem. As an illustration consider the constraint $b_f/t_f < 2$ on the variables b_f and t_f . If the domain for the variables are $[4,15]$ and $[1,6]$ respectively, the solution set using an interval arithmetic solver would be $[4,12]$ and $[2,6]$. The solver guarantees that for every value in the solution set, there is some value in the solution set of another variable that will satisfy the constraints. In this case, it may not be physically meaningful or possible to have $b_f = 6$ and $t_f = 3$. In other words, there might be other small constraints in the problem, which cannot be captured in the decision table structure. For this reason, there is a possibility of making an error while translating the solution set from the constraint solver to a problem statement. In those cases, the generated problems have to be rechecked for satisfaction of constraints.

8.2 Example

This section illustrates the working of the problem generator for steel member design with specific examples of steel beams subject to bending.

The first step in problem generation is the specification of a problem domain. The problem domain enables the instructor to focus on a particular section of the standard. In steel member design, if the domain is beam or column, the instructor also needs to specify the member and the loads acting on it. This ensures that critical design parameters like maximum moment and shear are calculated from a typical problem situation rather than they being specified directly.



Applet started.

Fig 8.1 UI for drawing a structural member and loads on it

Fig 8.1 shows the user interface window that the instructor uses to specify the member and the loads acting on it. A structural analysis program is used for computing the maximum displacement, design moment and shear force.

The next step, called problem definition, is the specification of the design rules that will be satisfied in the problems generated by the system. For instance, let us assume that the instructor wants to test the students on lateral torsional buckling. The instructor would want to generate problems in which the students have to check if a beam is safe with respect to lateral torsional buckling or not. A beam is not safe if its moment capacity is lesser than the maximum moment due to the load. This means that the problem

generator should create problem situations in which the specified beam member has a moment capacity corresponding to the lateral torsional buckling mode. Hence, the conditions that the instructor has to specify are:

- Moment capacity of the beam corresponds to lateral torsional buckling
- The beam is safe with respect to bending

The instructor specifies these actions by choosing actions in the corresponding decision tables. In this case, the relevant actions are that which assigns the moment capacity of the beam in the M_{nLTB} decision table and the action that assigns the beam to be safe in the *Beam* table.

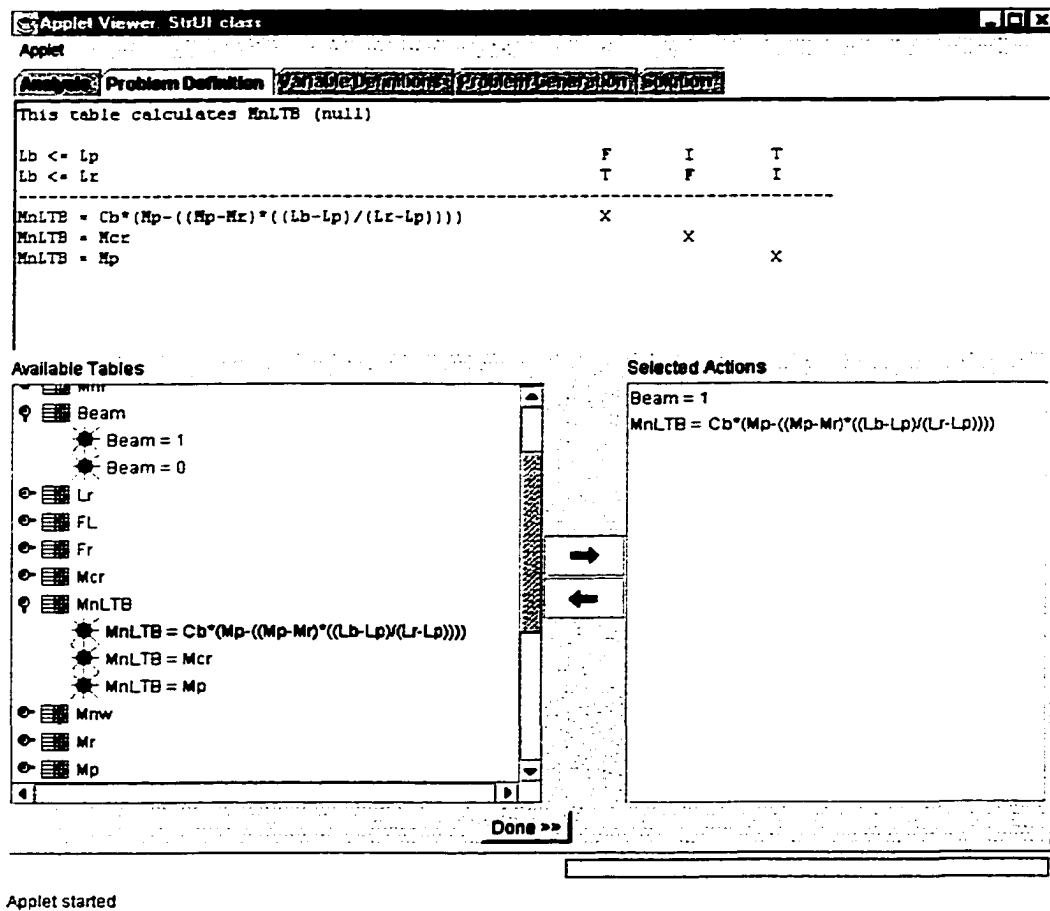


Fig 8.2 UI for specifying testing rules

Fig 8.2 shows the user interface that the instructor can use to specify the necessary action entries. The user interface lists all decision tables in the problem domain and the instructor can choose a set of actions from different tables to specify the type of problems to be generated by the EDT. The actions selected by the instructor correspond to nodes in the problem tree and the problem generator finds out all possible problem paths that pass through these nodes. Once the shortest of these paths is obtained, the instructor has to specify values for the basic variables in the problem path. The instructor can assign a specific value or an interval of values for each of the basic variable in the problem path.

Please enter values for the following variables:

Ishaped	1	Fy		Fyc	36
J		A		Fy	36
Z		homogeneous	1	S	
BT		BF		TF	
Lb	[100,150]	Sc		E	29000
G	11200	Cv		Iy	
rolled	1	Fys	36	Cb	1.00
Emax		RTU		Vmax	
D		ca		interval	5

Next: Choose a Shape >>

Generating Variables Done

Applet started.

Fig 8.3 UI window for defining feasible values for variable

Fig 8.3 shows the window where all the relevant basic variables are listed and the instructor can assign values to them. The unbraced length of the beam L_b is used as the main variable for the beam domain and in this example it is assigned an interval of $[100,150]$. The variable *interval* is used to specify the number of sub intervals the main variable is split into. The value 5 for the *interval* implies that the problem generator generates problems for the unbraced lengths of 100,110,120,130,140 and 150. The domain for all the basic variables whose values are not specified in this step is set some default value or by querying from a database having standard properties of steel beams.

Each of the action nodes in the problem path has a set of conditions that need to be satisfied for the action to be applicable. These conditions along with the variable definition form a constraint satisfaction problem, the results of which are translated into an equivalent problem statement.

Fig 8.4 shows the set of problems generated for the actions as selected in Fig 8.2. The list of shapes for each value of the unbraced length contains the beam designations that satisfy the requirements of the instructor. In this case all the shapes are governed by lateral torsional buckling and they satisfy all the requirements of the standard with respect to bending. This example demonstrates how the problem generator can be used to generate several problem statements that test the same design concept. The students can work out as many problems as they need till they understand and master the relevant design principles.

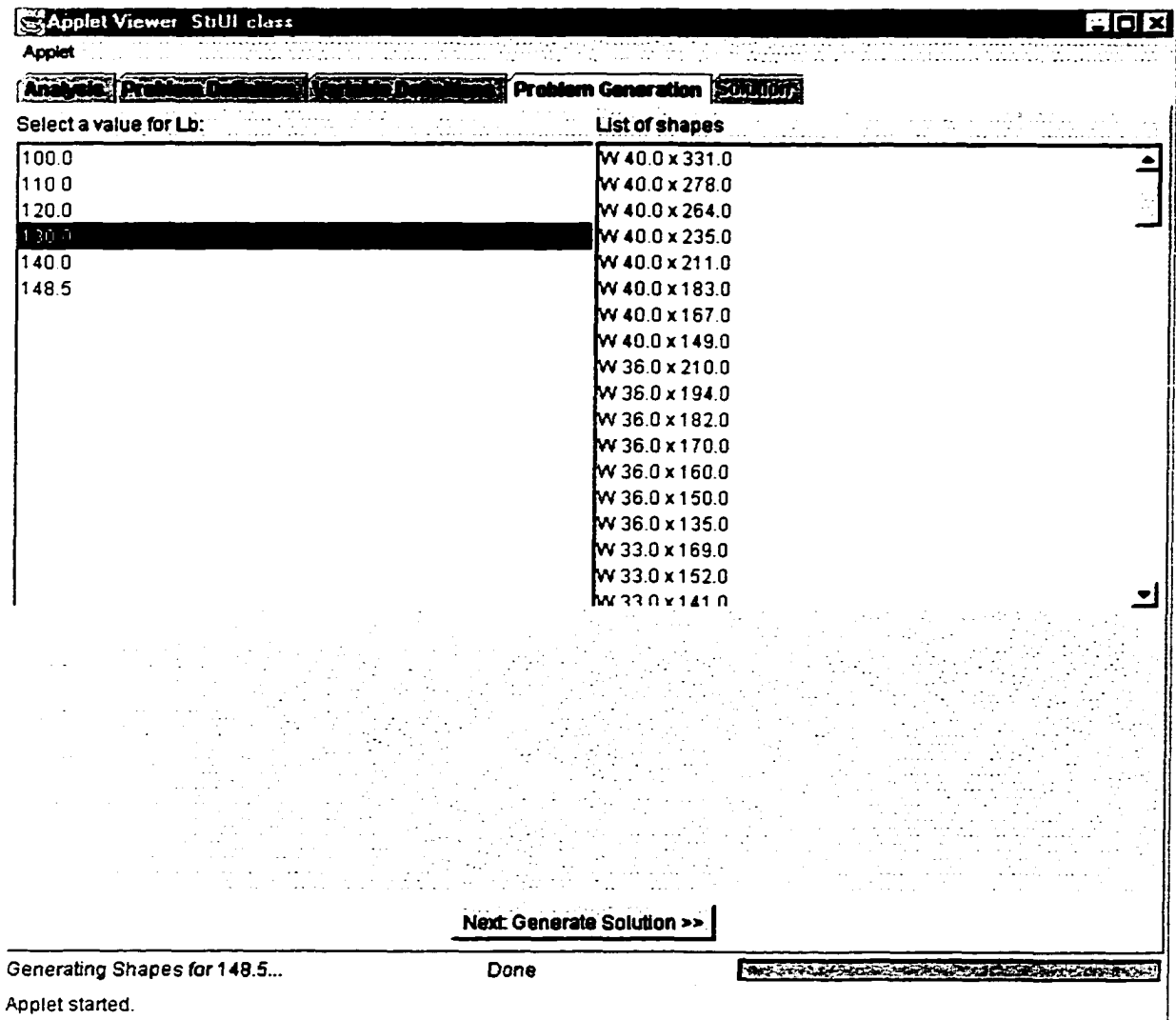


Fig 8.4 UI window that shows the generated problems

The step by step solution procedure can also be obtained for each of the problem generated by the system. The solution describes the logic of the problem solving procedure, the evaluation of each decision table by checking which conditions are satisfied and applying the appropriate action to assign a value to a data item. Fig 8.5 shows the solution for one of the problems generated by the system. This also represents the solution procedure that the student has to follow when solving the problem and hence

can be used to check student's solutions. The instructors can also use this solution to provide sample worked out problems to the students.

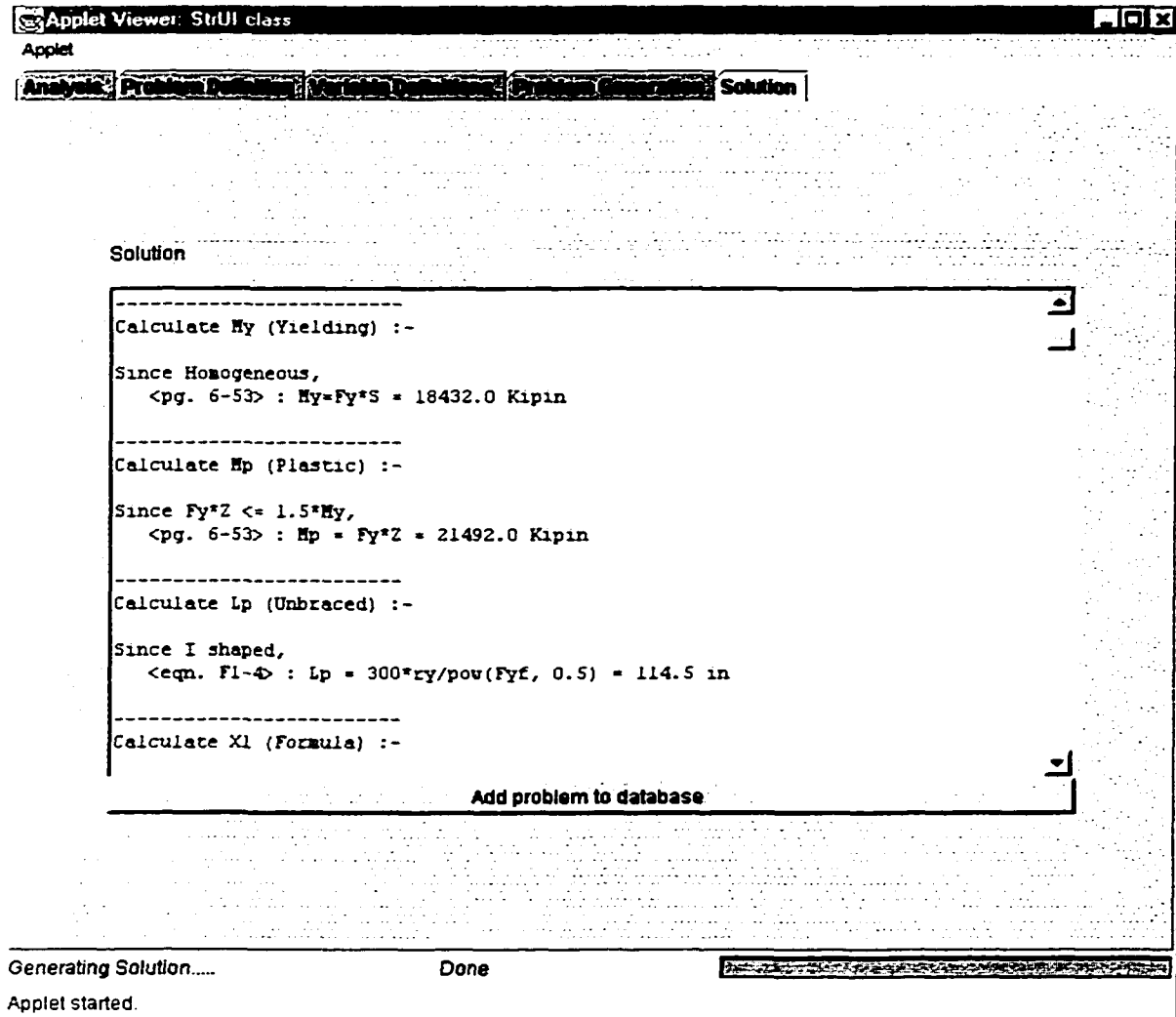


Fig 8.5 UI window that shows the solution procedure

Chapter 9

Conclusions

This chapter provides a summary of the work done in this thesis and also gives ideas for future work.

9.1 Summary

Software technology is revolutionizing the teaching environment by providing various prototype tools that improve the quality of learning and education. The engineering design tutor is one such tool that enables the instructor to construct problem sets to test a student's understanding of routine engineering design procedures. The underlying basic assumption is that solving a problem involves evaluating certain conditions and applying appropriate procedures or rules for that condition.

A decision table structure based on the SASE methodology is used to represent the design rules. By choosing specific actions or rules in the decision table structure, the problem generator can be used to construct problems that test the understanding and applicability of a particular rule or set of rules. The specified actions on the variables are used to obtain a list of constraints that need to be satisfied for the particular action to be applicable. This list of constraints on the basic variables along with the domains of those variables defines a constraint satisfaction problem. The solution to this, obtained by an interval arithmetic constraint solver is used to construct the problem statement. The system also provides the solution to the generated problems and hence reduces the amount of time spent in grading the student's answers. To have the advantages of scalability, portability and maintainability, a distributed component architecture was used in the design of the system.

The system will greatly reduce the instructor's time and effort spent in creating problem sets for the students and grading them. In addition, the software allows students to learn at their own pace, using the software whenever they want to. Thus the engineering design tutor can be a very useful tool for both the instructor and all the students involved in a design course. The working of the EDT was demonstrated by implementing the system for steel member design.

9.2 Scope for Future Development

The EDT was designed as a tool for teaching routine engineering design. This tool can be used in a design course and feedback from the instructor and student's can be used to revise some of the features of the system. Also a comparative study between the existing method of training and the electronic tutoring technique can bring out the advantages and disadvantages of the web based learning environment.

Presently, the system can be used only in engineering design fields. The system can be modified and extended to other fields as well. Intuitively the system can be used in any area where the solution procedure is based on a set of conditions and a corresponding list of procedures defined for each set of condition. A decision table structure should be able to represent the domain knowledge of the system. But there might be field specific problems and issues such as the type of constraint solver to be used and translating the solution to an equivalent problem situation.

While in engineering design the system serves the purpose of rote training by providing several problems of the same type, it can also be used as an assessment tool in other disciplines. The EDT has a flexible and scalable architecture due to the distributed component model of the system. This makes it possible to integrate it with other software

systems such as WebCT to develop large-scale educational environments with multiple capabilities. While WebCT provides facilities for authoring course material, the EDT can be used as a tool for problem and solution generation. The system can be used to generate problems that meet the requirements of the instructor and therefore be used for designing exams for the students. Students who are not confident about their understanding of the course material can use the system to test their knowledge. In short the list of possible future work is

- Feedback from instructors and students using the EDT used to revise some features of the system
- Implementation of the system for other engineering design fields
- Extension of the logic of problem generator to other disciplines
- Integration of the EDT system with other educational software systems

Bibliography

Bazillion, R. & Braun, C. (1998), "Teaching on the web and in the studio classroom", *Syllabus*, vol. 11, p 37-39

Grossman, S. (1999), Introduction Speech in Democratic National Committee Meeting, *Democratic National Committee Meeting*, Mar 20, 1999

Roschelle, J., Kaput, J., Stroup, W. and Kahn Ted, M. (1998), "Scalable Integration of Educational Software; Exploring the promise of Component Architectures", *Journal of Interactive media in Education*, 98(6)

Wong James, C.(1999), "Selecting Internet Technologies to Support Interactive Teaching and Learning at a Distance", *11th Annual ED-MEDIA World Conference on Educational Multimedia, Hypermedia & Telecommunications*, p 1883-84

Jennifer Burg, Yue-Ling Wong, Dan Pfeifer, Ane Boyle and Ching-Wan Yip (1999), "Publishing an imef Journal for Computer-Enhanced Learning", *11th Annual ED-MEDIA World Conference on Educational Multimedia, Hypermedia & Telecommunications*, p 1737-42

Murray G. Goldberg, Sasan Salari, and Paul Swoboda (1996), "World Wide Web Course Tool: An Environment for Building WWW-Based Courses", *Computer Networks and ISDN Systems*, 96 (28)

E. Kashy, D.J. Morrissey, Y. Tsai and S.L. Wolfe, "An Introduction to CAPA, A Versatile Tool for Science Education", *MSU-NSCL, Report 971, September 1995*

Kersley, G., "Artificial Intelligence and Instruction", chap.8, p 165-190, Addison-wesley, 1987

Imbean, G., Gauthier, G., Frasson, C., "Wordtutor: An Intelligent Tutoring System for Teaching Word Processing" in Norrie, D.H., Six, H.W. (eds), *Proceedings of the 3rd International Conference on Computer Assisted Learning*, Berlin, 1990, p 400-419

Brown, J.S. and Sleeman, D., "Intelligent Tutoring Systems", London: Academic Press, 1982

Teege, G, "A system for representation of domain knowledge for intelligent tutoring systems", Univeristy of Munich, Ph.D. Thesis 1991

Doyle, J., Sandewall, E. and Torasso, P., "Principles of knowledge Representation and Reasoning", *Proceedings of the 4th International Conference (KR94)*, San Francisco, CA, 1994

Burton, R.R. and Brown, J.S., "An investigation of computer coaching for informal learning activities", in Brown, J.S. and Sleeman, D., "Intelligent Tutoring Systems", Chap 4, p 79-98, London: Academic Press, 1982

Burton, R.R., "Diagnosing bugs in a simple procedural skill", in Brown, J.S. and Sleeman, D., "Intelligent Tutoring Systems", Chap 4, p 79-98, London: Academic Press, 1982

Regian, J. W., Shebilske, W., and Monk, J. (1992), "A preliminary empirical evaluation of virtual reality as a training tool for visual-spatial tasks", *Journal of Communication*, vol 42, p 136-149

Pimentel, K., and Teixeira, K. (1993), "Virtual Reality: Through the new looking glass", New York: Intel/Windcrest Books/McGraw-Hill, Inc

Pimentel Juan R., "Design of Net-learning Systems Based on Experiential Learning", *Journal of Asynchronous Learning Networks*, vol 3, issue 2, November 1999

Bouchlaghem, N., Beacham, N. and William Sher (1999), "Using Computer Imagery and Visualization in Teaching, Learning and Assessment", *11th Annual ED-MEDIA World Conference on Educational Multimedia, Hypermedia & Telecommunications*, p 1735-36

Erickson, T. (1993), "Artificial realities as data visualization environments", in Wexelblat, A. (Ed.), "Virtual Reality: Applications and Explorations", p 1-22, New York: Academic Press Professional

Chris Dede, R. Bowen Loftin, and J. Wesley Regian, (1994), "The Design of Artificial Realities to Improve Learning Newtonian Mechanics", *Proceedings of the East-West International Conference on Multimedia, Hypermedia, and Virtual Reality*, Moscow, September 14-16, 1994

Fenves, J. S., Wright, R.N., Stahl, F.I. and Reed, K.A (1987), "Introduction to SASE: Standards, Analysis, Synthesis and Expression", National Bureau of Standards, U.S. Department of Commerce

Gearan, A (1999), "A New breed of Internet surfers", Washington: The associated press

Roschelle, M. Koutlis, A. Reppening, etal, (1999), "Developing Educational Software Components", *IEEE Computer*, September 1999, Special Issue on Web based learning and collaboration, p 2-10

Marc Torrens (1997), "An application using the Java Constraint Library: The Air Travel Planning system", *Diploma thesis*, CS department, Swiss Federal Institute of Technology

Timothy J. Hickey, M.H. van Emden, and H. Wu (1998), "A Unified Framework for Interval Constraints and Interval Arithmetic", in Michael Maher and Jean-Francois Puget (eds.), Springer-Verlag, "Principles and Practice of Constraint Programming", *Lecture Notes in Computer Science*, vol 1520, p 250-264

E. Davis, "Constraint propagation with labels", *Artificial Intelligence*, vol 32, p 281-331

BNR, "BNR Prolog user guide and reference manual", 1988

M.Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf and F. Berthier, "The constraint programming language CHIP", *Proceedings of International Conference On Fifth Generation Computer Systems*, 1988

Owen Tallman and J. Bradford Kain, "COM versus CORBA: A Decision Framework", *Distributed Computing*, Sept-Dec 1998

Appendix

This appendix is a documentation of some of the classes and interfaces used in the implementation of the EDT.

PGapi Class (Problem Generator API)

This class is a generic class that is used by the problem generator server. The assumption made is that the problem solving is based on a decision tree structure and decision tables can represent the solution procedure. This class can be used to generate a problem by specifying actions in the decision table and the problem generated will correspond to those actions.

➤ public class PGapi

- static public String setinputs (Hashtable *hacts*, Hashtable *inpv*, String *var1*, String *lbl*, String *lbu*, String *lbr*)

This method translates the actions selected by the user and the user input variables into a form that can be used by the constraint solver. It returns the input data in the form of a string and is used by the subsequent methods.

hacts – Hashtable of actions selected by the user from different tables

inpv – Hashtable of user defined values for certain variables

var1 - the main variable

lbl, *lbu* - limits for *var1*

lbr - number of intervals into which *var1* is to be divided into.

- `static public Vector genIvar (String fname, Hashtable inputval, Hashtable tablecons, csps cs, pgdb pgd, String varl)`

This method takes in the constraints in the form of a string returned by the above method and does the constraint solving. The output is a vector of values of the main variable *varl*.

fname – String of input data returned by above method

tablecons – Hashtable of actions specified by user

cs - constraint solver object

pgd - object of a class that is specific to the problem domain. This class is used to get the ranges for the variables

- `static public Vector getprobs (String constraints, Vector udvar, csps cs, pgdb pgd, String varl, Double d)`

This method returns a set of problems for a particular value of *varl*.

constraints – String of constraints

udvar - Vector of the variables in the constraints

pgd class used to translate the output of the solver into problem statement.

DecTable Class

This is again a generic class, which represents the decision table structure. The class also has some static methods to communicate with all the necessary decision tables. A problem domain can be defined by the user, in which case only the set of decision tables corresponding to that domain are used by the static methods of this class.

➤ class DecTable

- static public Vector allvars()

This method is used to find all the basic variables in the decision table structure

- static public Vector getalltables()

This method is used to find all the table names in the decision table structure for a particular problem domain

- static public DecTable getnewtable(String *condvar*)

This method is used to get an instance of the decision table corresponding to the variable name *condvar*. It returns a null if the variable is a basic variable.

- public static void Checktree(Hashtable *tablecons*, Vector *tree*)

This method is used to find the shortest path to get the constraints in terms of the basic variables. The parameter *tree* is used to return the list of all possible paths and the first element of the tree corresponds to the shortest path. The paths are represented by a

hashtable of table names and an action number similar to the way the user selected actions are represented in *tablecons*.

tablecons – Hashtable of actions specified by user

tree – Vector with the all possible paths, the first element being the shortest path.

- private Vector doaction(int *choice*, int *column*)

This method returns the set of conditions that need to be specified for a particular action. The action is defined by *choice* while *column* defines the decision column number

- public void VarsRec(Hashtable *tablecons*, Vector *udvar*, Vector *dervar*, Vector *constr*)

This is a recursive method that uses the above method to find the entire list of constraints that need to be satisfied for the selected actions. The list of basic variables and derived variables are returned in *udvar* and *dervar* respectively. The set of constraints is returned in the parameter *constr*.

pgdb Class (Problem Generator - database)

This class is domain specific. It is mainly used to translate the constraint solver output into problem statements, get limits for basic variables and for other application specific steps. In this case for steel design, this class connects to a database to get properties and values of variables used in constraint solving. Using the output values from the constraint solver, it also enables choosing sections from the database as problem statements.

➤ public class pgdb

- public String Get(String *sect*, String *var*, Hashtable *consres*, Hashtable *inputval*)

This method returns the value of a variable used in the decision tables either from the database or from the constraint solver output.

sect - section ID.

var - name of the variable whose value is to be found.

consres – hashtable with the output of the constraint solver.

inputval - hashtable of values set by user.

- public Vector getprobs(Vector *udvar*, csps *cs*)

This method returns sections that correspond to the output of the constraint solver from the database.

udvar – Vector of variables used for constraint solving

- public boolean checksects(String *sect*, Vector *udvar*, Vector *constr*, Hashtable *inputval*, csps *cs*, String *var1*, Double *d*)

This method checks if a particular section (problem) is ok or not.

constr – Vector of constraints

d - value for variable *var1*.

Generate interface

This is an interface, which a problem generator server has to implement. The methods in this interface are implemented using the generic classes above and domain specific classes like pgdb. Some preprocessing or post processing may need to be done before using the generic class method implementations depending on the problem domain.

➤ public interface Generate extends Remote

{

- public Object SetInputs(Hashtable *hacts*, Hashtable *inputval*) throws RemoteException;
 - public Object GenLb(String *constraints*, Hashtable *inputval*) throws RemoteException;
 - public Object Getprobs(Double *Lbval*, String *constraints*, Vector *udvar*) throws RemoteException;
 - public boolean sectcheck(String *sid*, Vector *udvar*, Vector *constr*, Hashtable *inputval*, Double *Lbval*) throws RemoteException;
 - public Object Get(String *sid*, String *var*, Hashtable *consres*, Hashtable *inputval*) throws RemoteException;
- }