Session 3: Standards processing and code-related support

Session 3

Standards Processing and Code-Related Support
Traitement des normes et applications relatives aux normes
Normenverarbeitung und normenbezogene Unterstützung

Leere Seite
Blank page
Page vide

Towards a Standard-Independent Design Process

Vers une normalisation du processus de la conception

Der normunabhängige Konstruktionsprozess

James H. GARRETT, Jr.
Assoc. Prof.
Carnegie Mellon University
Pittsburgh, PA, USA

Mohammad MEHRAFZA
Research Assist.
University of Karlsruhe
Karlsruhe, Germany

Christophe MEINECKE
Research Assist.
University of Karlsruhe
Karlsruhe, Germany

Raimer J. SCHERER
Technical Univ. Dresden
Dresden, Germany

SUMMARY
The design process for most engineered products involves the use of knowledge found in product-specific design standards. As a result, creating formal models of these standards and processes has been an active goal of computer-aided engineering. This paper first reviews an early approach to formalising the use of design standards during the design of structural component cross sections and then describes two more recent approaches applied to more difficult detailed design problems for reinforced concrete columns and beams. The last section of this paper discusses the advantages and disadvantages of these three approaches.

RÉSUMÉ
L'élaboration du processus de la conception de la majorité des produits requiert l'utilisation d'une connaissance contenue dans les normes de calcul spécifiques au produit. Un des principaux objectifs de la conception assistée par ordinateur est la création de modèles formels de ces normes et de procédures permettant leur mise en application. Cet article passe en revue une première approche visant la formalisation de l'organisation des normes de calcul dans le cas du dimensionnement d'une section d'un élément porteur. L'article expose ensuite deux approches récentes appliquées à un domaine plus complexe, celui du dimensionnement des poteaux et des poutres en béton. L'article présente enfin les avantages et les désavantages de chacune des trois approches.

ZUSAMMENFASSUNG
Der Konstruktionsprozess der meisten Ingenieurprodukte wird durch Wissen beeinflusst, das aus den produktspezifischen Normen abgeleitet werden kann. Die Entwicklung sowohl formaler Modelle für die Abbildung des Normwissens als auch expliziter Prozesse für deren Nutzung ist damit ein wesentliches Ziel der Forschungsaktivitäten auf dem Gebiet des 'Computer Aided Engineering'. In diesem Beitrag wird zunächst ein früher Formalisierungsansatz für die Nutzung des Normwissens bei der Konstruktion von Bauteilquerschnitten beschrieben. Weiterhin werden zwei Systeme jüngeren Datums für die detaillierte Bewehrungsführung in Stahlbetonstützen bzw. Balken vorgestellt. Im letzten Abschnitt werden die Vor- und Nachteile der drei unterschiedlichen Ansätze diskutiert.

1. Introduction

The design process for most engineered products, such as load–bearing structures, involves the use of knowledge found in product–specific design standards, such as structural safety codes. These design standards, also referred to as codes or specifications, may define limitations on product attributes, prescribe design processes, or define limitations on product performances with respect to safety, reliability, usability, constructability, etc. As a result, creating formal models of these standards and processes for applying them has been an active goal of computer–aided engineering.

Initial research led to systems in which the information provided in the provisions of a standard were hard–coded into design synthesis procedures; the provisions of a standard were represented as lines of code interleaved with the design synthesis and process control knowledge. Early research in standards modeling proposed the use of declarative languages for describing the relationships between design data and requirements described in standard provisions. More recent research activity has focussed on automating the process of evaluating a synthesized design for conformance to such declaratively modeled standards. Early attempts led to systems that separated standards from design procedures, but that only checked designs for standard conformance after these designs have been created. Creating computer–aided design systems that make use of such standard models more proactively during the process of synthesizing a design (i.e., a standard–independent design process) has also been a research objective.

This paper first reviews an early design system that uses design standard models to determine the parameters of structural component cross–sections and then describes two more recent approaches applied to more difficult detailed design synthesis problems for reinforced concrete columns and beams. The last sections of this paper discuss the advantages and disadvantages of these three approaches and present some conclusions.

2. An Early Approach—SPEX

One of the earlier such attempts to formalize the use of design standards in the design process was the SPEX system developed by Garrett and Fenves [2]. SPEX uses a standard–independent approach for sizing and proportioning structural member cross–sections. The system reasons with the model of a design standard, represented as a set of decision tables and functions that each define how to derive a data item value, to generate a set of constraints on a set of basic data items that represent the attributes of a design to be determined. These constraints are then given to a numeric optimization system to solve for an optimal set of basic data item values (see Fig. 1).

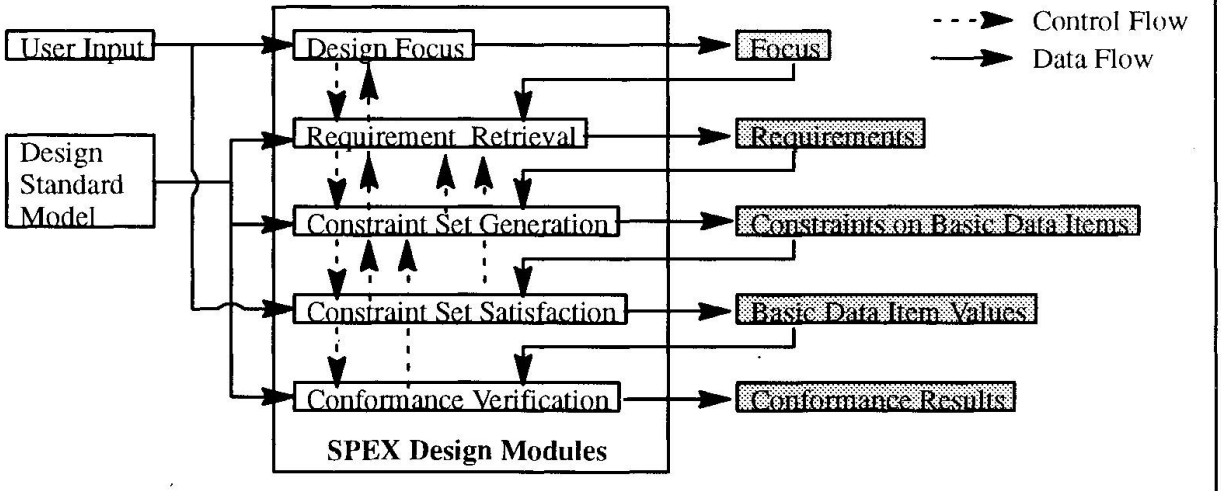


Figure 1 — SPEX Approach to Structural Component Design

SPEX is described as being a standard–independent design process in that both the process of generating the set of constraints from the standard and the process for finding the optimal solution of these constraints is generic and

thus not specific to the design standard being used. The results of the process are certainly dependent on the standard being used, but not the process. The design process in SPEX basically consists of the following steps: (1) the type of structural component being designed, the set of behavior limitations of that component on which to focus the design process, the design context (e.g., loads and support conditions), and the design objectives (in terms of a basic set of design attributes called *basic* data items) are specified by the user; (2) SPEX then finds the requirements in the standard that correspond to the design focus; (3) SPEX then generates a set of constraints from the identified requirements by backward–chaining through the decision tables and functions until only basic data items are referenced (all decision table conditions and functions in the backward chain form the constraint set); SPEX then sends this set of constraints off to a constraint solver and finds a set of values for the set of basic data items referenced in the constraints. Thus, it is assumed that both the objective function and all derived data item definitions in the model of the standard are expressed in terms of the basic data items (design attributes) of the object being designed.

3. More Recent Approaches—KEXPS and KOORDS

More recently, this issue of standard usage in detailed design has been investigated by Meinecke and Scherer in the development of KEXPS, a system for assisting in the design of reinforced concrete members, and by Mehrafza and Scherer in the development of KOORDS, a system for assisting in the design of reinforced concrete frames.

The problem of designing the reinforcement bar layouts for beams, columns, and their connections has many more parameters, mostly geometrical, than do the cross–section design problems that Garrett and Fenves addressed in SPEX. In addition, the EUROCODE 2 [1] for Reinforced Concrete Structural Members contains only a few provisions that relate to the detailed placement of the bars, while there are many more provisions relating to the required area of steel, depth of steel, and other more abstract design parameters. There are, however, several manuals that accompany the EUROCODES with additional design knowledge. As a result, Meinecke, Mehrafza, and Scherer had to develop approaches to their respective design problems that incorporated knowledge not found in a standard with the requirements that are found in EUROCODE 2. The next two sections describe each of these approaches in more detail.

3.1. KEXPS—A System for Design of Reinforcement Layout for Concrete Members

KEXPS consists of two separate components: a design process model and a product model (see Fig. 2). The fundamental design idea of KEXPS is to solve a design problem by generating and evaluating different design alternatives. Each of these alternatives is represented in the object–oriented product model as a hypothesis method. These methods generate the different design alternatives. KEXPS has been verified and fully implemented for the design of the longitudinal and transverse reinforcement of concrete columns.

For reinforced concrete columns, there are a limited number of widely accepted cross–sectional layouts for the reinforcement bars. As such, the design process usually proceeds from selection of basic layout alternatives to a detailed design of that layout for the specific loading context. While these layouts are not specified within the standard, they do greatly influence the details and the practicality of the resulting design. Also, the number and location of various parameters that need to be checked for conformance to a design standard will vary with each of these alternative layouts. Thus, in KEXPS, the representation and checking of EUROCODE 2 was organized around these layout alternatives.

The design process for a reinforced concrete column is performed in KEXPS in four major modules, as shown in Fig. 2: hypothesis, analysis, evaluation, and system selection [7]. First, the hypothesis module proposes several different bar layout design alternatives from a set of standardized layouts for concrete columns. These standard layouts are described in the manuals that accompany EUROCODE 2, such as [5]. Each fundamental alternative layout is represented as a hypothesis method. Before the alternatives are worked out in detail, a fundamental mechanical review takes place and mechanical behavior constraints are checked. These methods are comple-

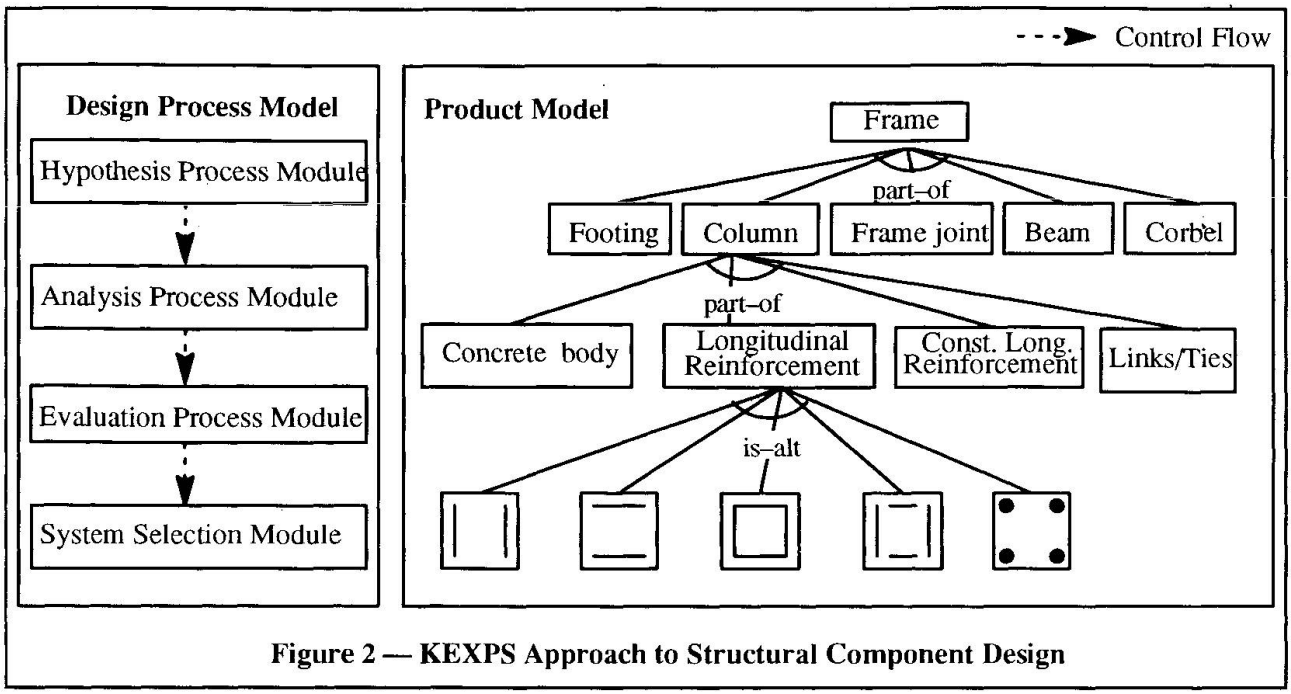


Figure 2 — KEXPS Approach to Structural Component Design

mented by refinement methods that develop the fundamental layout towards a specific design aspect (e.g. minimum of steel area). Together, the hypothesis methods assign values to various attributes such as bar spacing, bar sizes, edge distances.

The analysis module checks the generated design alternatives against standard requirements which are not considered in the hypothesis module. These requirements are represented as rules that evaluate the attributes of the alternative column reinforcement layouts. The evaluation module uses additional non–standard–related criteria, such as ease of construction, cost or bonding, to evaluate the various design alternatives that were found to conform to the standard requirements. The results from the evaluation module for the remaining alternative bar layouts are compared using a fixed formula to select the best alternative. Finally, the selection module gives the user the opportunity to view and select a layout from the remaining alternatives by using the commercial CAD–System UNICAD (Hochtief AG [4]).

3.2. KOORDS—A System for Synthesis of Reinforcement Layouts for Reinforced Concrete Frames

KEXPS is able to take great advantage of the limited number of acceptable layouts for reinforced concrete columns in organizing and using the requirements from EUROCODE 2. However, the design of the reinforcement layout for beams is more complicated than for columns because the layout of the bars for a beam framing into two columns may have very complicated layouts that do not easily breakdown into a small number of standardized layouts. Instead of following a SPEX or KEXPS approach, where the template of parameters is first selected and then the parameters are computed, KOORDS builds up, i.e., synthesizes, the reinforcement bar layout using a generative system. KOORDS must thus deal with a large number of possible design contexts and must determine which standard requirements apply to which contexts. Both the design process and design product models are implemented in a full object–oriented environment. The design process in KOORDS involves: generation of reasonable solution alternatives, verification of alternatives, evaluation of alternatives and selection of the optimal alternative. The optimization of reinforcement layout is based on concepts developed by Mehrafza and Scherer [6].

The synthesis process in KOORDS integrates complicated geometric reasoning with domain and heuristic knowledge for generating layouts. The heuristic knowledge for locating pieces of reinforcement bar refers to standard–derived limiting values (e.g., bar spacing) that are assumed to be defined in any standard for reinforced

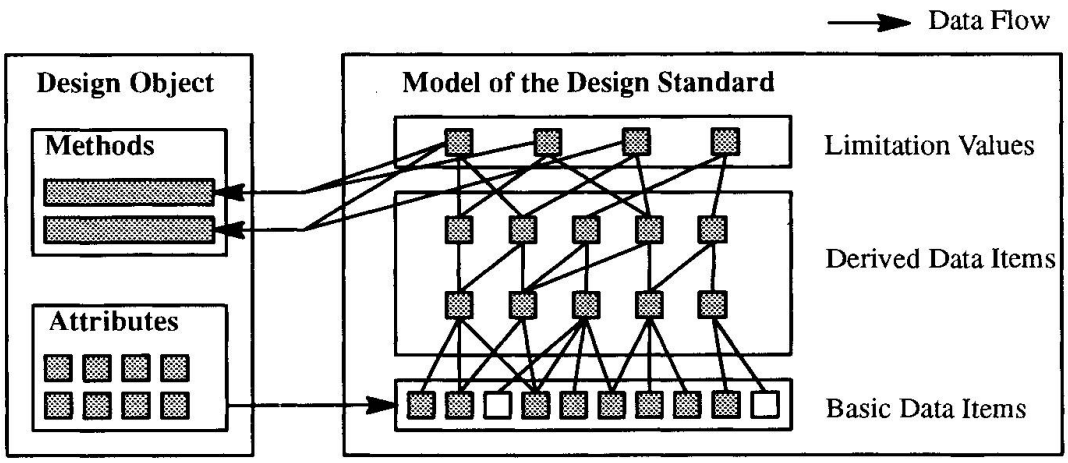


Figure 3 — KOORDS Approach to Structural Component Design

concrete beams (see Fig. 3). If a standard does not provide a definition for such a limiting value, a default value is used (and hence becomes a heuristic and not a requirement). Thus, in this way the design knowledge is integrated with the formal models of design standards through this interface of anticipated limiting values. The standard is thus represented as an object–oriented model similar to that described in [3], with the top level data items not a set of requirements, but rather a set of limiting values defined in terms of a network of other data items.

Object classes for the various components of the layout have slots for these limiting values that are sent a message when the limiting values are needed during the synthesis process. When a limiting value is requested, the method associated with that slot is evaluated, which causes other data items in the standard model to be evaluated. Eventually basic data items are reached, which are mapped back to the object requesting the limiting value. These data items are referred to as mappings. A new standard can thus be added to this system by creating a set of definitions for limiting values, definitions of supporting data items, and the basic data item mappings.

4. Discussion

In this section, the advantages and disadvantages of SPEX, KEXPS and KOORDS are discussed, with special attention paid to the separation of the standard knowledge from other design knowledge.

SPEX was originally described as a standard–independent member design process, because SPEX symbolically reasons with a formally modeled design standard and generates a set of constraints which are then solved generically by a standard–independent optimization process [2]. However, SPEX can be more precisely defined as a system for standard–independent proportioning (a kind of design process) of structural component cross–sections. It performs no synthesis task in the sense that synthesis is the combining some objects to build a meaningful and more complex object which provides some specific functions. SPEX does not reason with arbitrary collections of related objects, but rather assumes a fixed set of interrelated objects to exist as the basis of both the product model and the standard model. Because SPEX is based on the assumption that the product model used by the design process is the same as that used in the standard model, the design process employed by SPEX is not truly standard–independent.

The approach taken in SPEX works well for design problems in which: (1) an appropriate cross–section type can be *a priori* determined for a given design problem, (2) each of these types can be described in terms of basic data items found in the standard, and (3) the standards provide ample constraints on these basic data items describing the cross–section. As a result of these limitations, the SPEX approach for standard independent member design cannot be directly applied for more complicated design contexts, such as the design of reinforcement layout for a concrete column or beam. SPEX could be used to determine some of the more high–level decisions, such as the depth of a concrete beam or the amount of tensile and compressive steel. However, to use SPEX for layout of the

reinforcement, the layout would have to be parameterized so that all parameters are unique variables and the requirements would have to be expressed in terms of these variables, making the solution very problem–specific. This limitation exists because SPEX makes the assumption that the data model (i.e., the set of basic data items) used in the standard is exactly the same as the data model used to describe the resulting design.

Even if a complex reinforcement layout could be parameterized, the objectives and constraints would also have to be expressed in terms of the large set of parameters to make the problem amenable to a SPEX–based approach. Some specific limits, such as flange buckling for steel structural members and bar spacing for reinforced concrete structural members, are expressed in terms of detailed parameters. However, standards do not normally express most requirements in terms of such detailed design parameters. Rather, standards normally express limitations in terms of high–level performance measures (e.g., crushing stress) or sizing parameters (e.g., area and depth of reinforcing steel). As such, even if a more detailed design problem such as reinforcement layout could be parameterized, the objectives and constraints derived from the design standard would mostly be in terms of more abstract design attributes. Additional knowledge describing the relationships between those abstract design attributes and the detailed design attributes would be necessary to solve this problem using a SPEX–based approach. Defining this knowledge in terms of equality constraints would likely be very difficult if not impossible for a complicated bar layout for a reinforced concrete beam. Thus, detailed design problems, such as reinforcement bar layout, cannot be realistically solved by a SPEX–based approach.

Like SPEX, the design approach employed in KEXPS is a standard–independent design approach. The design process invokes the hypothesis methods associated with the alternative templates, and then analyzes, evaluates and selects among these generated alternatives (i.e., using a generate and test approach). Both the hypothesis methods of the product model and the analysis rules are dependent on the constraints and criteria defined within a standard. KEXPS is able to handle more difficult detailed design problems than SPEX, but only after all design alternatives are *a priori* enumerated as templates and the appropriate design standard knowledge is represented for each alternative template.

KEXPS is able to get around the need for a large, complicated set of parameters for describing all alternatives (a problem with SPEX) by using an object–oriented approach to represent each design alternative as a template object. In KEXPS , standard provisions that are appropriate for an alternative are represented as either hypothesis methods or analysis rules associated with that alternative's template object. KEXPS thus organizes the representation of the standard around a product model. For the column design problem, the product model is a small group of the column reinforcement layout templates, as illustrated in Fig. 2. The product model contains all domain and standard–specific knowledge, which is required for the design process, while the process model comprises only generic design process knowledge and is thus standard–independent. Thus, when a new standard is to be used, only the product model must be modified, not the design process. However, KEXPS divides the standard information *a priori* into that used during design and that checked after the design has been created. In other words, there is no redundancy of representation of the standard between hypothesis methods and analysis rules. This fixed, *a priori* division of standard information makes it difficult to use the model of the standard in processes other than the KEXPS design process. For example, if attempting to check designs generated by systems other than KEXPS, some requirements may go unchecked or unconsidered during design. Also, by building the representation of the standard directly into the product model, KEXPS is built upon the same exact assumption as was SPEX — the data models for the product and the standard model are one in the same.

In KOORDS, the standard is declaratively modeled and then used both during the design optimization process and in the post–design standard conformance checking process. In this respect, KOORDS is similar to SPEX in keeping the standard knowledge separated from other design knowledge, thus allowing for independent creation of the standard model(s). The standard model in KOORDS is very similar to that used in SPEX except for one major difference. In KOORDS, the standard is modeled as a set of limiting values, where these limiting values are referred to in the design methods of the objects being designed (e.g., minimum–bar–spacing for reinforced concrete member design). In SPEX, the standard is modeled as a set of requirements representing various behav-

ior limitations. SPEX reasons with the standard model by identifying the relevant requirements for the behavior limitations specified by the designer on which to focus. KOORDS starts by evaluating the design methods in an object and requesting the limiting values its needs from the standard model. The standard model computes the values using data for the existing design situation provided by the requesting design object and returns these values. Thus, for KOORDS, the parts of the standard model to use in designing an object are fixed for that object to be those limiting values referred to in its design methods. In contrast, SPEX provides more flexibility for a designer to specify the requirements on which to focus when designing an object; the user may issue a design focus statement that will greatly influence what part of the standard is considered during design of that component.

KOORDS is different from SPEX in the way that it then processes the information it gets from the standard model. In KOORDS, the optimization process is packaged in methods which apply to the special objectives and constraints of each class of design object while the SPEX optimization process is based on a generic optimization approach. Each object in KOORDS is responsible for all processes which should be performed on itself, including determining optimal proportions. Therefore a reinforcement group in a beam uses a specialized method, different from that used for a reinforcement group in a column, to optimize its proportions. Because the scope of the KOORDS system, in contrast to SPEX , is creating objects, relating them to each other and proportioning them in order to satisfy the design requirements, it is able to handle a complicated assembly of bar pieces, which SPEX cannot.

KOORDS thus maintains the intentions of the SPEX system (i.e., to make use of a separately modeled design standard within a standard–independent design process) and overcomes the limitations of the SPEX approach (i.e., lack of a structured product design data model). The standard independent nature of the design procedure in KOORDS comes not from the usage of a generalized optimization procedure, but rather from the definition of a standard–independent interface consisting of a set of limiting values assumed to be defined in all design standards for the design of reinforced concrete beams.

5. Conclusions

Thus, with KEXPS and KOORDS, we have illustrated how this concept of a standard–independent design process, a concept earlier illustrated in SPEX but for limited types of problems, can be made more robust and able to handle much more complicated, detailed design processes, specifically the layout of reinforcement for reinforcement concrete columns and beams. While SPEX maintained standard–independence by using a domain–independent optimization procedure, it was limited to solving problems defined by a set of scalar parameters and constraints defined in the standard over those parameters (e.g., steel and reinforced concrete cross–sections, but not entire components). KEXPS and KOORDS illustrated that it is possible to still maintain standard–independence when solving more difficult detailed design problems by using a domain–specific, but still standard–independent, design process. KEXPS models the standard within its product model requiring that the product model be changed if the standard being used is changed. In KOORDS, the specific standard being used in the design process is accessed through a standardized set of limiting values assumed to be defined by all standards applicable to the given design domain. Thus, by using this interface, KOORDS keeps the standard model separated from its design objects making it possible to change from one design standard to another without having to modify the design methods. In both KEXPS and KOORDS, the optimal design methods are defined for and stored with the individual design objects, making it much easier than in SPEX to incorporate specific design methods and knowledge .

6. Acknowledgements

The first author thanks the Alexander von Humboldt Foundation and Carnegie Mellon University for providing the opportunity to spend time visiting and interacting with the latter three authors at Universität Karlsruhe. This interaction led to the discussions contained in this paper. The latter three authors thank the DFG (German Research Foundation) for the financial support provided under Contract No. DFG Sche 223/3–1, by the "CAD/CAM Forschungsschwerpunkt" sponsored by the state of Baden–Wuerthemberg and by the European Council with the ESPRIT III Project No. 6609 COMBI.

7. References

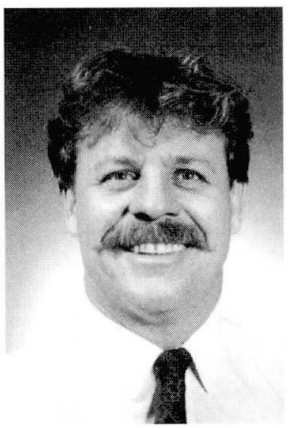
1. EUROCODE 2 Deutsches Institut für Normung e. V., Design of Reinforced Concrete Structures, Beuth Verlag, Berlin, 1992.
2. GARRETT, JR., J. H. and FENVES, S. J. "A Knowledge–Based Standard Processor for Structural Component Design." Engineering with Computers, 2(4), 219–238, 1987.
3. GARRETT, JR., J. H. and HAKIM, M. M. "Object–Oriented Model of Engineering Design Standards." Journal of Computing in Civil Engineering, 6(3), 323–347, 1992.
4. HOCHTIEF AG, UNICAD Programmierhandbuch, in German, Frankfurt/Main, 1991.
5. KORDINA, K. : Bemessungshilfsmittel zu Eurocode 2 Teil 1, Planung von Stahlbeton– und Spannbetontragwerken, DAfStb, Heft 425, (Design with Eurocode 2: reinforced and prestressed concrete structures) in German, Verlag Ernst & Sohn, Berlin, 1992.
6. MEHRAFZA, M. J. AND SCHERER, R. J. "Regelbasierte Formulierung von Variantenelementen in der Stahlbetondetaillierung," Dortmunder Expertensystemtage '91, Verlag TUF Reihnland, page 274–287, 1991.
7. MEINECKE, C. AND SCHERER, R. J. "Blackboard–Based Expert System for the Design of Reinforcement Layout." in Proceedings of Management of Information for Construction, MIT–C, Singapore, 1989.

MiniCode Generator: A Practical Research Application for Standards Processing

Générateur de minicodes: recherche pratique pour le traitement des normes

Ein "Bauvorschriften-Generator" für die praxisbezogene Normenverarbeitung

Dana J. VANIER
Senior Research Officer
National Research Council
Ottawa, ON, Canada



Dana Vanier has degrees in engineering, building science and architecture. For the past 15 years he has been a researcher at the National Research Council, Canada. He works currently on investigating information technology for the construction industry. His research includes electronic technical information and electronic building codes.

SUMMARY
MiniCode Generation permits users to extract project-specific building codes based on a user-assisted selection of building attributes. A rule-based pre-processor queries the user for details on the construction type, building size, and occupancies, as well as features such as sprinklers, fire alarms, and combustibility rating. The MiniCode Engine then extracts the relevant provisions from the building code in less than 2 seconds on a standard personal computer. The user is then free to browse the resulting MiniCode or modify the attributes and generate a new MiniCode.

RÉSUMÉ
Le générateur de minicodes permet au concepteur de consulter une norme de construction d'après un ensemble de caractéristiques du bâtiment. Un pré-processeur demande à l'utilisateur des précisions concernant le type de construction, les dimensions et l'usage du bâtiment, ainsi que certaines particularités, telles que la présence d'extinction automatique à eau, le système avertisseur d'incendie ou la classe de combustibilité. Le système extrait alors de la norme de construction, en moins de 2 secondes, les dispositions applicables, et cela à l'aide d'un ordinateur personnel conventionnel. L'utilisateur peut dès lors parcourir le minicode ainsi obtenu ou changer les caractéristiques de départ pour en obtenir un autre.

ZUSAMMENFASSUNG
Dem Planer wird die Möglichkeit geboten, projektbezogene Bauvorschriften durch benutzergesteuerte Eingabe von Gebäude-Kenndaten abzurufen. Ein regelgestützter Vorverarbeitungsrechner erfragt vom Benutzer Einzelheiten über Bauart, Abmessungen, Höhe und Nutzungsart des Gebäudes sowie über spezielle Bauelemente wie Sprinkleranlage, Brandmelder und Brennbarkeitsklasse. Das Computersystem wählt dann über einen Standard-PC in weniger als 2 Sekunden die einschlägigen Bestimmungen aus den Bauvorschriften aus. Der Benutzer kann in dem auf diese Weise zusammengestellten Bauvorschriftenauszug beliebig recherchieren oder die Ausgangskenndaten ändern und dadurch einen neuen Vorschriftenauszug erstellen.

1. INTRODUCTION

Many research papers describe projects using expert systems, databases, neural networks and other information technologies to process building standards. The facts indicate that, in spite of the theoretical discussions of the advantages of such systems, there are few applications for building code users [1]. This could be attributed to the difficulty in creating knowledge bases for complex technical and legal documents, to the relative inexperience of building code writing bodies in the area of KBES, to the uniqueness of each and every building code or standard, and to the slow-growing marketplace for Information Technology (IT) tools [1]. The term "building code" in this paper means building code or standard and "electronic code" identifies an IT tool providing access to building codes.

It is the view of the author that research endeavours in this field need not apply the most advanced information technologies, some research can use existing and stable techniques and still create innovative, useful applications. The MiniCode Generator [2, 3] is one research project that can have a significant impact on the building code user community. This paper describes past research activities, the current status of the project and future considerations and possibilities for the MiniCode technique at the Institute for Research in Construction (IRC).

1.1 Past Research

The MiniCode Generator was developed in conjunction with a number of code writing bodies to meet their individual needs [3]. Each partner in the research consortium had expressed an interest to include the MiniCode attributes and features as a search option in their proprietary electronic codes. The first phase of the MiniCode project is now complete and this product has been delivered to the research consortium for testing, over forty copies of the software are in use across Canada.

The first phase concentrated on the needs of building plan examiners, - those municipal or regional authorities responsible for building plan verification prior to the issuance of building permits. The MiniCode Generator identifies significant building attributes or features, such as occupancy of the building or height of the building, it contains attribute tags for each of the building code provisions, and it generates abridged versions of building codes based on the associated attributes.

1.2 Current status

The Plan Examiner MiniCode consists of three discrete software components. (1) The National Building Code of Canada [4] Building Classifier is an expert system preprocessor that assists the users to classify their building; (2) the MiniCode Engine culls the 4000 tagged provisions of the National Building Code of Canada (NBCC) using the Classification Description provided by the user, and (3) the MiniCode Viewer enables the user to browse the resulting document in a hypertext environment, and to record the state compliancy of a building for the resulting provisions. The MiniCode Generator works with Windows™ 3.1 or the Macintosh™ operating system.

The NBCC Building Classifier is a set of 20 classification trees using a rule-based expert system to assist the user classify the building. For example, knowing the building is used for residential occupancy and is less than 4 storeys in building height and is less than 600 square metres in area, the NBCC Building Classifier informs the user that this type of building must conform to the NBCC provisions dealing with small buildings. Some classification trees are obviously more complex, consisting of dozens of interrelated rules. In addition, any number of approximately 50 building features such as noncombustible construction, wood shingles, or sprinklers can be selected and, depending on this selection, the relevant provisions are included or excluded. The output of the NBCC Building Classifier is called the Classification Description.

The MiniCode Engine consists of a bitmap of the attribute tags for provisions of the building code and a parsing engine. The Building Code Bitmap contains the provision number and the applicable attribute for each tagged provision. This static file is searched and compared with the Classification Description provided by the user. Generating of the MiniCode is transparent to the user and take approximately 2 seconds to create a MiniCode on a 50 MHertz, 486 personal computer [2]. The exclusion principle, whereby only irrelevant provisions are excluded from the MiniCode, forms the

basis for the search mechanism. The exclusion principle assists the speed of the search and reduces liability for the user and the software developer alike. The MiniCode Engine's output is a sparse description of the classification attributes along with a listing of the provisions for the MiniCode. This file can be saved in ASCII format or can be printed.

The MiniCode Viewer is a WinHELP™ application that reads the file generated by the MiniCode Engine and allows the user to view a listing of the provision numbers and headings for the resulting sparse building code. Double-clicking the appropriate provision number or heading displays the full text of that regulation. Although the MiniCode Viewer can only access provisions that appear in the MiniCode, the hypertext environment of WinHELP™ permits the user to view all linked references and all defined terms, interactively.

2. THE DESIGNER MINICODE GENERATOR

The next anticipated phase of the MiniCode project builds on the past experience and creates a more robust Generator to address the needs of building designers. This extension encompasses a different set of problems and opportunities. For example, the plan examiner already knows if a building is sprinklered: the sprinklers are shown on the drawings. A designer, on the other hand, wants to know if the building must be sprinklered or the advantages of sprinklering when it is not mandatory.

The steps required to develop the Designer MiniCode are included in this paper. These steps should serve as examples for others intending similar MiniCode projects with their building codes.

2.1 Prerequisites

The prerequisites for starting a similar project include an in-depth knowledge of the building code or direct access to a building code expert. In our case, an engineer with 35 years of code writing experience was contracted to perform this function. Encoding the MiniCode Engine also requires someone with "C" programming skills, but this skill set is readily available in both the academic and industrial communities. Access to an expert system shell to encode the classification rules is essential, as is familiarity with WinHELP™. The remainder of the tasks can be handled by a knowledge engineer familiar with building codes and architectural engineering.

The task are broken to four discrete areas. The NBCC Building Classifier contains the rule base and creates the Classification Description. The Document Tagging provides the Building Code Bitmaps. The MiniCode Engine compares the Classification Description to the Building Code Bitmaps and creates the sparse list of the MiniCode provisions. And finally, the MiniCode Viewer allows the user to browse the MiniCode and record the state of compliancy of a building for each provision.

2.2 NBCC Building Classifier

The initial selection of which attributes or features are included for classification is somewhat subjective, and dependant on building code document and the knowledge of the domain expert. It is recommended to start with a small number of classification attributes, twenty is a workable number, and with a limited number of features, approximately 50 are easy to maintain. The numbers can always be increased as and when required.

Generally the building attributes relate to different types of buildings or to a faceted classification for the building type. Faceted classification involves attributes such as major occupancy where the building may be any one of a selection of values such as A1, A2, A3, A4, B1 or B2. Building features, on the other hand, are normally a type of construction or component that is contained in the building. These include items such as elevators or favourable soil conditions. An additional distinction between the two is that building attributes require detailed classification trees to assist the user, whereas features may not. In some instances features can be suppressed if they are not applicable for a type of building. This, in itself, is a simple logic rule: if the building is a residence then do not display potential features such as escalator or electrical vault in that dialog box.

The classification attributes and features for the NBCC MiniCode include the following:

Building Attributes: Major occupancies, subsidiary occupancies, building height, firewall, building area, standpipe and hose, occupant load, number of facing streets, interconnected floor space, high building, sprinklers, noncombustible, fire alarm, barrier-free access, etc.

Building Features: Stucco, wood shingles, house, dwelling units, garage, carport, elevator, escalator, electrical vault, air-supported structure, farm building, etc.

Detailed classification trees must be developed for each building attribute, these are developed in conjunction with the building code expert. Examples of a classification tree for small buildings and the associated rule base are shown in Figs. 1 and 2, respectively.

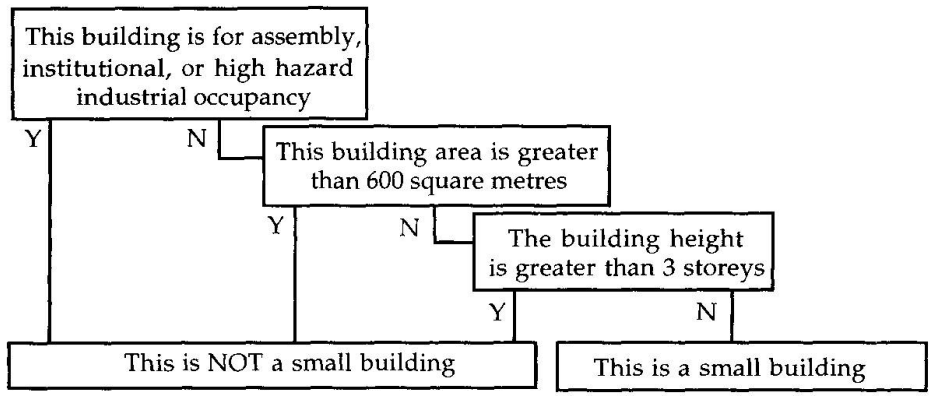


Fig. 1 Classification Tree for Small Buildings

Part9:-p9Occ(All),p9area(All),p9storeys(All).
"Small Buildings (Part 9) must meet occupancy AND area AND storey height requirements"

p9Occ(c):-MO(c).
p9Occ(d):-MO(d).
p9Occ(e):-MO(e).
p9Occ(f2):-MO(f2).
p9Occ(f3):-MO(f3).
"C, D, E, F2, or F3 are valid occupancies"

p9area(600):-setarea(b600).
"600 square metres or less is a valid area"

p9storeys(1):-NumberofStoreys(1).
p9storeys(2):-NumberofStoreys(2).
p9storeys(3):-NumberofStoreys(3).
"1, 2, or 3 are valid storeys heights"

Fig. 2 Rule base for Classification Tree for Small Buildings

In some instances these classification trees should be augmented with additional rules that are above and beyond the scope of the building code. For example, if the building is only one storey in building height there is no need to query the user about whether or not the building is a high building.

Entering the classification trees and the validation of the rules requires considerable attention to detail. Typically one day of work is required for the building code expert to establish classification trees for each building attribute. An additional day is required for each attribute to encode this information and to validate the rule base. In total, this could entail four person-weeks of work.

2.3 Document Tagging

The tagging of the building code document can commence once the building attributes and features have been identified, and the tagging can be done in parallel to the encoding of the rule base.

For the original Plan Examiner MiniCode the tagging was done manually on sheets prepared for the operation, and transcribed to the electronic format. For the Designer MiniCode the existing tagging must be updated to include additional building attributes and features. Typically the information could be entered in the format shown in Fig. 3

Provision Number	Major Occ.	No. Stry.	High Bldg.	Bldg. Area	Part 3	Sprk.	Occ. Load	Fac. Str.	Fire Alr.	Fire Wall	Features
1.1.1.1.	C,D										
1.1.1.2.		GT2									
1.1.1.3.	C+D				T			EQ2			
1.1.1.4.					F						Farm
1.1.1.5.	EX B2							1		T	
1.1.1.5.	EX B2	GT6						2		T	
1.1.2.1	C	LE3						GT1			

Fig. 3 Tabular Layout for Data Entry of Document Tagging

The building code expert enters the building attribute information in the proper columns, each row contains information for one provision. Keep in mind that multiple rows can be required for the same provision, as with provision 1.1.1.5. Features are entered in the last column in free text format. Any input mechanism can be used to record this information. In the original MiniCode, a word processor with tabular cells provided a user-friendly interface. This file can be easily exported to any format such as tab delimited ASCII or Rich Text Format (RTF) and can be parsed to the proper format for the MiniCode Engine. The parsing entails decoding the shorthand data entry required for the format presented above. For example "LE3" means less than or equal to 3 storeys in building height. This would be parsed into three tags for provision 1.1.2.1.: a tag is required for one, two or three storey buildings. Other shorthand notation includes "C,D,E" meaning C or D or E major occupancy, "C+D" for C and D major occupancies, "EX C" for all major occupancies except C, "EQ2" meaning equals 2, and "GT2" meaning greater than 2. The resulting file is called the Building Code Bitmap.

The Document Tagging is a time-consuming, labour-intensive task. Depending on the complexity of the building code and the knowledge level of the individuals involved, the time required to tag the document can average at 3 minutes per provision. In the case of the NBCC with 4000 provisions this equals over 5 person-weeks of work. Although this may seem to be a considerable amount of time, remember it is probably the first attempt to classify and structure the information contained in that building code. The writing of the parser may require a day or two.

2.4 MiniCode Engine

The MiniCode Engine is described in detail in Cornick and Thomas [2]. Basically, the Engine compares the Classification Description to the Building Code Bitmap and excludes those provisions that do not met the criteria.

The output from the MiniCode Engine includes the classification of the building along with the list of the remaining building code provisions, as shown in Fig. 4.

Project Name: **IABSE Demo** Major Occupancies: **C**
Minor Occupancies: **none** Number of Storeys: **1**
Fire Wall: **Not present** Fire Alarm: **Not present**
Building Area: **500** Building Code: **Part 9**

1.1 1.1.1 1.1.1.1 1.1.2 1.1.2.1 1.1.3 1.1.3.1 1.1.3.2 . . . . .

Fig. 4 MiniCode Listing

As mentioned earlier, the MiniCode generation takes approximately 2 seconds on a standard personal computer. In actual fact there are 183 bits required to encode the information in the Building Code

Bitmaps; so the Engine compares 4000 x 183 bits to extract each MiniCode. The writing of the Engine may involve a day or two.

2.5 MiniCode Viewer

The MiniCode Viewer reads the output of the MiniCode Engine and presents these to the user in the form shown in Fig. 5.

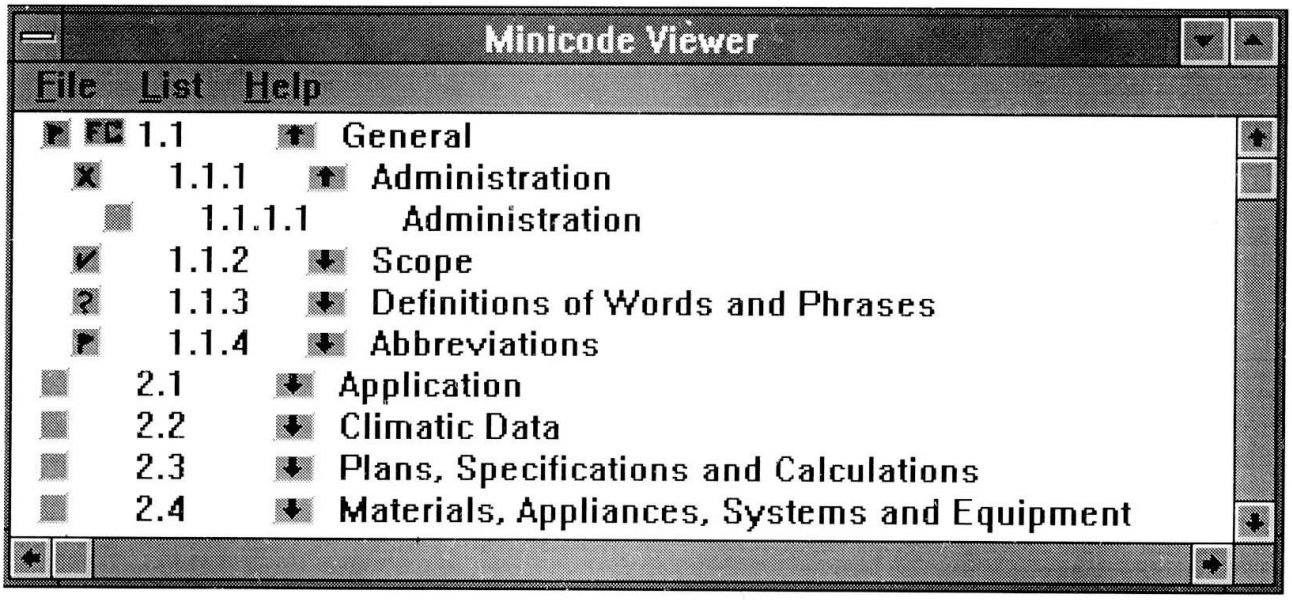


Fig. 5 MiniCode Viewer

The toggle boxes to the left of the Fig. 5 allow the user to record the status of compliancy for each of the building code provisions in the MiniCode. These toggles include an check mark, question mark, an x, or a flag, as shown in Fig. 5. The arrow immediately to the right allows the user to expand the contents as shown in Fig. 5.

The NBCC Document Viewer uses WinHELP™ to view a hypertext version of the building code. Fig. 6 demonstrates some of the possibilities of the environment. This viewer is similar to other hypertext environments available on personal computers [5, 1]. There are many tools now available to assist software developers create WinHELP™ applications. It is difficult to estimate the time required to develop such a tool, but less than 2 person-weeks were required for the NBCC Viewer.

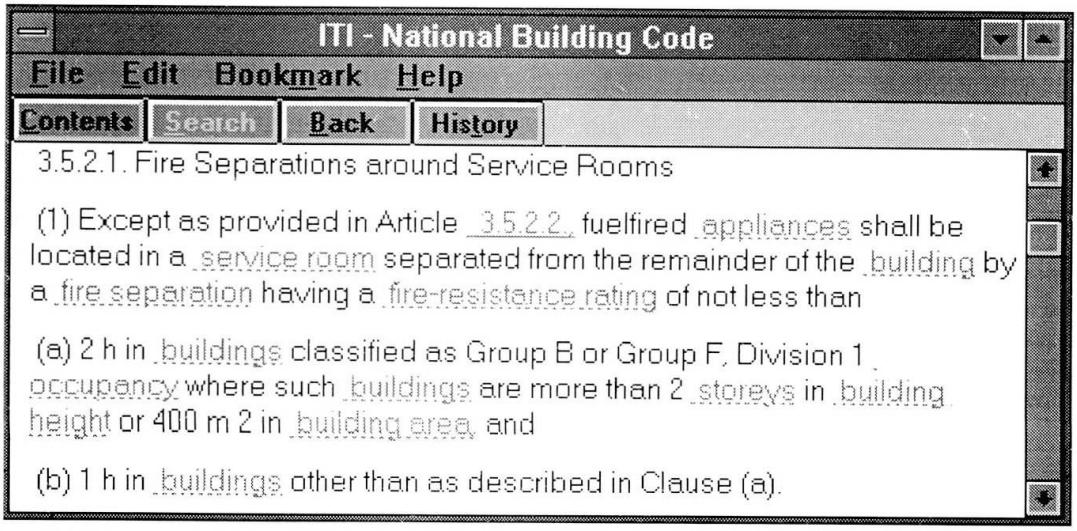


Fig. 6 NBCC Document Viewer

3. DISCUSSION

The purpose of this paper is to describe a product resulting from research into the generation of sparse building codes, and to encourage other researchers to develop similar tools. There are numerous advantages to this type of technology for researchers, software developers and users involved with building codes.

3.1 Researchers

Although this paper may resemble a cookbook for creating an electronic code software product, there is significant research required to identify the building attributes and features for individual building codes. In addition, there is knowledge engineering required to decipher the classification trees for the NBCC Building Classifier. This cannot normally be handled by a building code expert, as the expert understands the document too thoroughly and may not be able to relate the information to strict "if_then_else" rules. The researcher, *cum* knowledge engineer, must direct the activities of the expert to create a clean, concise, and accurate representation of the classification process.

On completion of any new NBCC Building Classifier, the code writing body must also be part of the feedback loop. If a specific building attribute cannot be accurately classified in the rule base by the knowledge engineer and building code expert, this normally implies that it is difficult for ordinary users of the building code to understand. This was encountered a few times in the Plan Examiner MiniCode when it was impossible to decipher when a specific provision could be excluded, or when it couldn't.

Unanswered problems still exist for the researcher. For example, how deep and how broad can the building attributes and features become? Can the MiniCode classification be extended to hundreds of features, or will a limitation be reached at 256 bits or 2 Kbits in the Bitmap?

3.2 Software Developers

Profit is the driving force in the development of software products, and there are few available electronic code tools. Typically the potential client for software developers is a code writing body, as most building code users desire some stamp of approval from the jurisdictional authority [6]. Code writing bodies are then the logical initiator for these software products because they have a vested interest and they have the expertise on staff to validate the rule base and create the tags for the provisions. To defend this position, the code writing bodies have been selling their products for decades and they provide updates, errata, supplemental instructions and other services: a software product is almost identical. It is strongly recommended that electronic products be part of the product offerings from code writing bodies [7]. It will not be long before most similar information is available electronically, and before users start demanding that type of service and product.

3.3 Building Code Users

The users are the main beneficiaries of electronic codes. Building codes are complex documents [8], and have remained so for decades. As construction techniques become more complex and as construction regulations increase because of increased liability and risk, free-trade, new products, etc., there is an ever-increasing need to assist the users in this building code quagmire [8].

The MiniCode Generator is one of the few attempts to codify the information in building codes [9, 10, 11, 12]. The technique offers advantages to novice and experienced users alike. For example, the NBCC Building Classifier serves as a tutorial for many of the factors concerning building classification. Help messages are available at every step in the classification tasks to explain the signification of specific selections, as shown in Fig. 7.

Novice users can learn as they browse, whereas experienced users can quickly gloss over the supporting literature and concentrate on data entry. Without sounding too much like promotional literature, using the NBCC Building Classifier is like having a building code expert as a colleague.

Generating sparse building codes in seconds reduces the amount of work for the plan examiner and the building designer alike. MiniCodes considerably reduce the amount of information to peruse, in some case MiniCodes are 20 % of the original document length. This is an added benefit to the users: they are guaranteed that the excluded provisions are not applicable, and need not be reviewed.

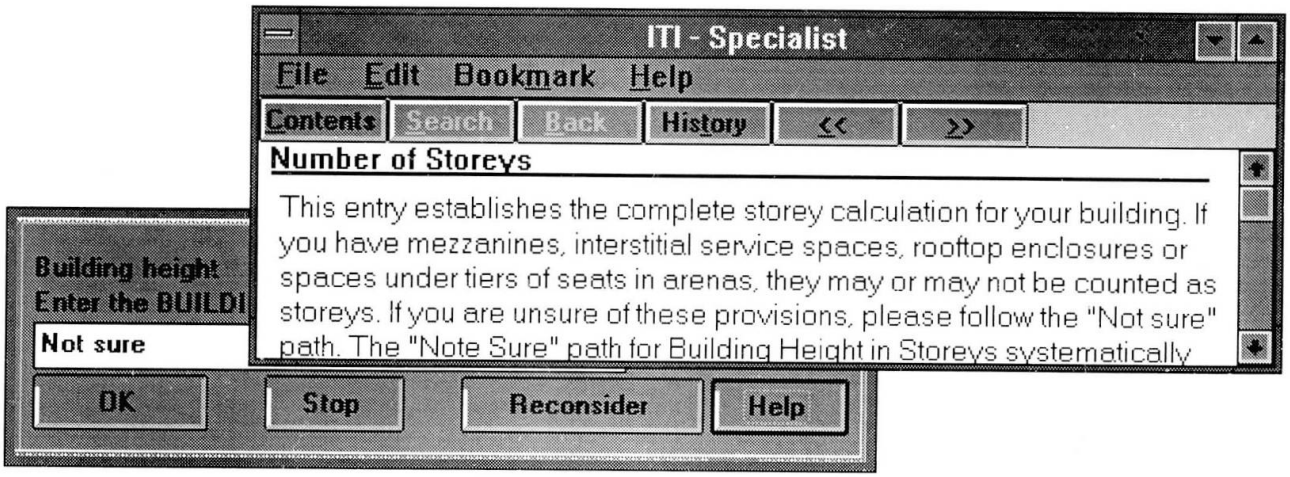


Fig. 7 NBCC Building Classifier Help System

The WinHELP™ version of the MiniCode Viewer is a simple-to-use, effective, browsing interface to building codes. It provides hot-links to related provisions and interactive defined terms.

3.4 Problems

Although the MiniCode Generator appears to be a robust, useable tool, the author feels that the significant contribution to the community is the development and structuring of the building code information for the NBCC Building Classifier. The generation of MiniCodes is interesting as a concept; however, even if the building code is reduced considerably, even 20 % of the 400 pages of the NBCC is a formidable building code.

The Document Tagging is labour-intensive, requiring a high level of expertise from the building code expert. In addition, there is need for constant validation of the data entry in the Document Tagging. For example, although one provision definitely applies to sprinklered buildings, the provision may identify a solution that is of interest to designers contemplating no sprinklers. Therefore the sprinkler tag should not appear for that provision.

Updates to building codes may be a problem. That is, changes to a building code have to be reviewed and retagged, and the NBCC Building Classifier may have to be restructured. However, experience has shown at IRC that changes to the NBCC, with respect to the classification tags, do not occur in significant numbers to warrant concern. In some instances changes in building codes may even alter the rule base: the current debate in Canada concerning sprinklers for all buildings would alter the NBCC Building Classifier significantly.

The NBCC is a model code for Canada that is modified by three provincial counterparts to suit regional requirements [4]. As such, the MiniCode technology can be easily transferred to these documents. However, it is extremely difficult to transfer the NBCC MiniCode application to other National codes. It would be equally difficult to transfer this application directly to other Canadian codes such as the National Fire Code of Canada.

3.5 Future Extensions

Although the Plan Examiner is still under *Beta* review, the current users have been very specific in their demands for future extensions. As a result, a number of modifications have been suggested: the Designer MiniCode could include "What if?" analysis and subject index comparisons.

These two features basically work on the same principle: the user can generate a number of MiniCodes and then compare any two at a time. In the "What if?" scenario the user might compare a two storey with a three storey variation. The "logical difference" between these alternatives would identify additional provisions that have to be investigated.

The subject index comparison can also be a valuable tool for designers: (1) create a MiniCode for a standard three storey townhouse; (2) create a MiniCode of provisions dealing with basements, and (3) the logical difference is a listing of all provisions for basements in three storey townhouses. This can have numerous applications for designers such as finding provisions dealing with sprinklers for commercial buildings, identifying mezzanine provisions for low-rise residential construction, or locating all provisions about fire alarms and two storey offices under 2000 square metres

Additional functionality in the Designer MiniCode could include validation of input by the user, the ability to append user-specified notation, the addition of more building attributes and features, and increased depth in the classification of building attributes. The first two of these features must be explained. The validation of input addresses potential erroneous or trade-off decisions by the designer. For example, a building may not require sprinklers; however, the designer may decide that sprinklers may permit combustible construction, a cheaper construction type. The NBCC Building Classifier informs the user that sprinklers are not mandatory, but still recognizes that all sprinkler provisions cannot be excluded. On the other hand, if novice users select this option incorrectly, they are informed of their error. The user-specified notation permits national, regional or local agencies to append additional notes or interpretations to specific provisions. For example, the Fire Commissioner of Canada might have a specific ruling on NBCC Section 1.1. In this case a special icon would appear in the NBCC Viewer beside that provision. This feature has been implemented for one government agency, as shown with the [FC] icon in the upper portion of Fig. 5. To access the relevant literature, the user clicks on the [FC] icon.

Other future extensions include interfaces to any version of electronic code, such as the provincial building codes. In the case of the NBCC MiniCode Generator, it could be linked to commercial packages currently available or to future products such as the upcoming Compact Disk (CD) version of the NBCC [13]. There is also the obvious possibility of including Building Code Bitmaps for provincial building codes in one MiniCode Generator, thus allowing users to select the appropriate provincial MiniCode.

4. CONCLUSIONS

The MiniCode Generator described in the paper is a simple implementation of existing technologies. The paper details a sequence of steps in the development of an information technology tool for building designers. These steps can be used by code writing bodies or software developers to create their own versions of the MiniCode Generator for their national or regional building codes.

The general goal of researchers in the area of electronic codes, or at least the envisioned panacea, is an interface to Computer Aided Design (CAD) systems [14, 1]. This is only possible through the proper classification and structuring of building code information. The MiniCode Generator is a step in the right direction, but falls short of the CAD interface envisioned by designers. Future developments such as detailed classification systems [1] could provide additional steps towards this goal. In any case, the classification or structuring of building code information can only assist the user community in the long term.

This paper describes the second stage [3] in the development of tools to assist architects, engineerings and building officials access building codes. The *Beta* trials generally indicate that users are pleased with the tool and its capabilities. However, the users have indicated the need for more functionality. More research is envisioned for the creation of the Designers MiniCode.

REFERENCES

1. VANIER, DJ, *A Parsimonious Classification System to Extract Project-Specific Building Codes*, PhD Thesis, Université de Montréal, Montréal, Québec, October 1994.

2. CORNICK SM, and THOMAS JR, HyperCode, MiniCode and ExperCode: Evolution of a Code Users Environment, Paper presented at *Management of Information Technology for Construction*, eds. Krishan S. Mathur, Martin P. Betts, and Kwok Wai Tham, Singapore, (n. pag.), August 1993.

3. VANIER DJ, Minicode Generation: A Methodology to Extract Generic Building Codes, *CAAD Futures '93: Proceedings of the Fifth International Conference on Computer-Aided Design Futures*, 7-10 July, Pittsburg, PA, pp. 225-239, 1993.

4. NBCC, Associate Committee on the National Building Code, *National Building Code of Canada*, 10th ed., National Research Council Canada, Ottawa, Ontario, 1990.

5. YABUKI N, and LAW K, Hyperdocument Model for Design Standards Documentation, *Journal of Computing in Civil Engineering*, Vol 7, No. 2, pp. 218-237, 1993.

6. Market Study, Institute for Research in Construction, *CD-ROM Market Study: Characteristics of the Market for Methods of High Density Electronic Storage and Delivery of Code and Standards Information*, by Opinion Search, Inc., National Research Council Canada, Ottawa, Ontario, 27 p., 1993.

7. VANIER DJ, THOMAS JR, and WORLING JL, Standards Processing 2000, In *Proceedings of ASCE First Congress on Computing in Civil Engineering*, Washington, DC, 20–22 June 1994.

8. FENVES SJ, RANKIN K, and TEJUJA HK, *The Structure of Building Specifications*, NBS Building Science Series, No. 90, Center for Building Technology, National Bureau of Standards, Washington, DC, 77 p., 1976.

9. CRONEMBOLD JR, and LAW KH, Automated Processing of Design Standards, *Journal of Computing in Civil Engineering*, Vol. 2, No. 3, pp. 255-273, 1988.

10. GARRETT JH Jr., Object-Oriented Representation of Design Standards, In *Proceedings of International Association of Bridge and Structural Engineering (IABSE) Colloquium*, Bergamo 1989, IABSE-AIPC-IVBH, ETH-Honggerberg, Zurich, pp. 373-382, 1989.

11. DE WAARD, M, *Computer Aided Conformance Checking*, Self Published Ph.D Thesis, Melis Stokezijde 11, 2543 CA, The Hague, The Netherlands, 203 p., 1992.

12. SHARPE R, OAKES S, HASELDEN P, and DAVIDSON I, Automation of the Building Code of Australia, Proceeding of the Third World Congress of Building Officials, New Orleans, LA, Session 5, 1-6 May (n. pag.), 1993.

13. VANIER DJ, MELLON BS, WORLING JL, and THOMAS JR, Management of Construction Information Technology, In *Proceedings of Management of Information Technology for Construction*, eds. Krishan S. Mathur, Martin P. Betts, and Kwok Wai Tham, Singapore, , pp. 75-84, August 1993.

14. DYM CL, HENCHEY RP, DELIS EA, and GORNICK S, A Knowledge-Based System for Automated Architectural Code Checking, *Computer-Aided Design Journal*, Vol. 20, No. 3, pp. 137-145, 1988.

Representation and Processing of Structural Design Codes

Représentation et élaboration des normes de construction en génie civil

Darstellung und Verarbeitung von Normen des Bauwesens

Vlasis K. KOUMOUSIS
Assoc. Prof.
Nat.Techn. Univ. Athens
Athens, Greece

Vlasis Koumousis graduated in civil engineering at NTUA in 1975, and received his M.Sc. and Ph.D in applied mechanics from Polytechnic University of New York in 1976 and 1980 respectively. He has worked as consultant and currently is an associate professor at NTUA.

Panos G. GEORGIOU
Ph.D. Student
Nat.Techn. Univ. Athens
Athens, Greece

Panos Georgiou graduated in civil engineering at NTUA in 1988 and currently is working on his doctoral thesis. His research is involved with technical code based optimal design of multi-storey buildings. He has also worked in consulting firms designing reinforced concrete buildings.

Charis J. GANTES
Assist. Prof.
Nat.Techn. Univ. Athens
Athens, Greece

Charis Gantes graduated in civil engineering at NTUA in 1985, and received his M.Sc. in Civil and Ph.D in structural engineering from MIT in 1988 and 1991 respectively. Since 1994 he is assistant professor at NTUA. His research interests are in computer-aided structural design.

SUMMARY
A logical scheme is presented for the representation and processing of structural design codes. The subdivision of codes into several distinct, strongly interrelated parts, and the frequent cross-references between different codes make it difficult to grasp the structure and the domain of applicability of different provisions. The declarative nature of the code provisions and the sequence of their demands can be implemented directly in logic programming. Moreover, the search for solutions can be performed with the sequential backtracking algorithm of Prolog language, while the design space can be formed using several relational databases accommodated in Prolog. The proposed scheme is illustrated with the design of a steel roof using Eurocode 3.

RÉSUMÉ
Un schéma logique est proposé pour la représentation et l'utilisation de normes de construction en génie civil. La subdivision de ces normes en plusieurs chapitres, très interdépendants, et les fréquentes corrélations entre ces normes rendent difficile la compréhension de la structure et le domaine d'application des différentes règles. La nature descriptive des règles et la succession des prescriptions peuvent être mises en oeuvre directement par un logiciel. La recherche de solutions peut être réalisée en utilisant les algorithmes écrits en langage Prolog. Le schéma proposé illustre le projet d'une toiture métallique utilisant l'Eurocode 3.

ZUSAMMENFASSUNG
Beschrieben wird ein logisches Schema für die Darstellung und Verarbeitung von Normen des Bauwesens. Durch die Unterteilung der Normen in verschiedene, miteinander verbundene Teile und wegen der vielen Querverweise zwischen verschiedenen Normen ist es sehr schwer, die Struktur und den Bereich der Anwendbarkeit einzelner Bestimmungen zu verstehen. Der deklarative Charakter der Bestimmungen und die Reihenfolge ihrer Forderungen können aber direkt logisch programmiert werden. Insbesondere kann die Suche nach Lösungen mit der Programmiersprache Prolog durchgeführt werden, während das Lösungsgebiet unter Benutzung verschiedener relationaler Datenbanken auch in Prolog formuliert werden kann. Das vorgeschlagene Schema wird für den Entwurf eines Stahldaches mit Eurocode 3 erläutert.

1. INTRODUCTION

This work summarizes the general philosophy that governs modern structural codes, and outlines the major steps towards the computerized representation of codes together with their incorporation in automated structural design software. The complex structure of design codes and the frequent cross-references in them make it difficult for the users to follow the line of reasoning and establish the interconnection between interrelated specifications. A similar situation exists also within an integrated computer system that contains analysis and design modules. Generally, codes contain both qualitative information and algorithmic procedures to perform various conformance checks, or the design of specific members. Therefore, different means are used to represent these two distinct types of information. A general scheme based on logic programming is presented for the interpretation of the algorithmic requirements imposed by structural codes, as a part of a structural optimization process. This approach is illustrated with specific clauses that describe some of the requirements of "Eurocode No. 3" [1] for the design of steel structures.

2. GENERAL PHILOSOPHY OF STRUCTURAL CODES

As opposed to older structural codes, that were based on the allowable stress concept, most of today's codes follow the principles of ultimate limit state design [2]. This means that the structure can exhibit plastic deformations under extreme loading conditions. Moreover, the structure must be designed for specific limit states defined in the code. These limit states are the ultimate limit states, that ensure non-collapse of the structure, or other forms of failure, and the serviceability limit states, that provide a control on the damageability of the structure. Besides the limit states, important features of the codes are the partial coefficients, that replace the overall safety factor of the allowable stress design philosophy. There are the partial coefficients for the loads and partial coefficients for the materials used in the structure. Partial coefficients are important in all the conformance checks required by a code. Such checks follow a specific safety format that usually takes the following form:

$$S_d \leq R_d \tag{1}$$

where S_d refers to the design value of the actions, that are needed for a particular conformance check, and R_d is the design value of the corresponding resistance. Relation (1) can be used either in explicit, or implicit form to calculate a geometric or other parameter of a component of a structure. The design values represent deterministic estimates of the stochastic functions that describe the actions and resistances, and are expressed in terms of their corresponding nominal or characteristic values S_k and R_k in relations of the type:

$$S_d = \gamma_F S_k, \quad R_d = R_k / \gamma_m \tag{2}$$

where γ_F is the partial coefficient for the action, and γ_m the partial coefficient for the material.

The reliability of the produced designs must be proven on the basis of the structural codes that apply for each case. Reliability analysis studies are important to validate structural design codes ([3], [4]). From the existing structural codes very few have been checked for consistency [25].

3. CODE REPRESENTATION

The information contained in a code is traditionally addressed to the designer, who needs to have complete control of the code, so as to fully interact with it during the process of designing a structure. Today, codes constitute also important parts of integrated computer systems, where, depending on the form of their representation, they interact with other software modules towards the integrated design of structures.

Current integrated systems that perform the analysis and design of structures contain a set of routines to carry out the necessary checks, or calculate the required quantities. These are linked to the analysis module to perform the so-called "one shot analysis". The outcome of such an analysis is a conformance check with the algorithmic part of the code. This is the "hard-coding" approach and is the most widely used in existing automated structural design software packages.

To adapt their orientation to the current needs, modern codes tend to provide information in an algorithmic way. Thus, besides the traditional format that expressed the information in the form of design graphs and tables, more and more flow charts and algorithms, that were used to produce these graphs and tables, accompany modern codes.

Several efforts on the computerized representation of codes in general, and structural codes in particular, are reported in the bibliography ([5]-[14], [20]-[22]). Pertinent investigations started in the late sixties by Professor Steven Fenves [5] and continue until today.

What is interesting today, is the development of integrated design systems that simulate the course of actions taken by experienced designers. The fundamental difference of current methods lies in their ability to use parts of the program on an "if needed" basis. With greater computational power becoming increasingly available, new methods and programming environments make this goal feasible. The knowledge about a domain can be incorporated into computer systems in the form of knowledge based expert systems (KBES) that, when properly structured, can manipulate a knowledge base using the inference engine to prove a list of goals and subgoals. All the forms of knowledge representation, used in AI applications, have been utilized to interpret design codes. Among these, the most popular are the production systems or rule based systems, the use of frames, which are being replaced by object oriented programming, and other forms of semantics such as decision-tables, information networks and organisation systems.

The decision tables, proposed by S. Fenves, as an extension to tabular decision logic, provide a systematic way to represent and process the requirements of design codes. Moreover, the data and the application rules can be logically channelled in the form of a network of decision tables. Special pre-processors have been developed to manipulate the decision tables and to incorporate them into analysis programs [11].

Information networks are a collection of nodes and branches, where each node represents a data item and each branch represents a relation between two nodes. The provisions of a code are represented in the form of a graph which results after parsing the provision. The leaves of the graph represent the basic data, while the items at higher levels represent evaluated data.

Various systems have also been proposed that correspond to various forms of Database Management Systems (DBMS), and aim at offering environments for multi-purpose access systems in integrated design. Among these, the object oriented DBMS systems seem to be the most popular [9], [21].

To render the "if needed" character of a code representation in a way that corresponds to the design of structures, appropriate design strategies must be employed. These may be either under the full control of the designer, leading to interactive design systems [14], or may be computerized design strategies that simulate the design process on the basis of decomposition, structural optimization, numerical experimentation etc., to attain a list of design goals. The development of ad-hoc design strategies, and heuristic or general design theories, presents considerable difficulties due to their synthetic nature. The existing design strategies tend to simulate the course of actions taken by designers, or address the problems of conceptual design and preliminary design of structures.

Moreover, the system must be in a position to reveal the appropriate list of conformance checks that are needed for the design of a particular component on the basis of its type, state of stress, its particular features and the type of analysis. In addition, hypertext system technologies may be used in

a productive way to enhance the usability of such systems, especially regarding the descriptive parts of a code [18].

In another direction, new tools and methods need to be developed to assist design standards writing organizations in checking the completeness, uniqueness, i.e. absence of redundancy and contradiction and correctness of the codes [7].

4. LOGICAL SCHEMES FOR THE REPRESENTATION AND PROCESSING OF DESIGN STANDARDS

The representation and processing of structural design codes can be performed also in a way that follows the intrinsic logic of the code, and transforms code clauses into formal logic using logic programming. This approach has been applied extensively in the form of rule based systems for descriptive codes, such as architectural codes [10], but its potential can be extended to cover also the algorithmic parts of codes [22], [23].

A logic program is a set of rules that define relations between data structures, while computation, in the context of logic programming, means to prove that a goal statement (rule) is true using the other defined rules. This process is constructive and provides the values for the goal variables, which constitute the results of the computation [15]. The implementation of logic programming in Prolog is performed by using a control mechanism based on sequential search with backtracking on a restricted class of logical theories, namely Horn clause theories. Prolog can be viewed also as a relational database programming language, which manipulates and alters the database information, that describes a particular domain.

The main advantage of a Prolog representation of the code requirements, is that the structure of the language is such that it sequentially demands the truth of the predicates of a rule. Generically, this enables the implementation of a sequence of requirements in one predicate by combining descriptive, i.e. declarative, and procedural predicates. Rules in Prolog may have several definitions, thus the appropriate rule is matched and processed on a "if needed" basis.

As a result, using Prolog as the means of expressing the design philosophy of the code, clarity and modularity are preserved as main characteristics. This implies that a separate module must exist that defines the partial safety factors for the actions and the material properties. Another module must define the loading cases and the load combinations that apply to different types of structures that correspond to the ultimate limit states. Yet another module must define all the resistances covered by the ultimate limit states of the code, together with the domain of their application. Finally, another module must control the serviceability limit state checks. These correspond to the standard modules that exist in every code. In addition, there may exist other modules that contain the particular information of the code that refers to the specific material, the detailing of the structural components, the connections, and the rules of good design practice that the code recommends. This information is tailor-made and corresponds to the particular structure of the specific code. These separate modules can be maintained easily by incorporating changes in the standards, and can be placed in parallel with the relevant parts of other codes in integrated systems that accommodate different design codes.

Feijo et. al. ([17],[18]), Jain, Law et. al. [22], and Rasdorf and Lakmazaheri[23] presented alternate models for the representation of design codes based on first-order logic and their incorporation into design automation systems. Feijo et. al. suggested also a formal link with hypertext systems, in which data is stored in a network of nodes connected by links. Rasdorf and Lakmazaheri put the emphasis on the axiomatic formulation of the organizational submodel and the representation and processing submodel of the code, which can be interrogated via theorem proving.

After building the separate modules, dominant part of the entire process becomes the control of the information flow. This is based on the sequential backtracking algorithm which is the searching mechanism of Prolog Language. Backtracking can be considered as a searching technique of all possible solutions in a systematic manner. Its description in procedural terms is as follows:

Having a predicate that depends on a list of n variables $[x_1,x_2,..,x_n]$ that participate in a particular check and are stated according to the safety format of the code. This list must satisfy a requirement or property $P_n[x_1,x_2,..,x_n]$. In general, we assume that x_i can take values from a set X_i. The assumption is, that all choices in a set X_i are linearly ordered. Once the values of $x_1,x_2,..,x_{k-1}$ are fixed, we select the smallest value x_k among the set X_k, which leads to a feasible list which satisfies $P_k[x_1,x_2,..,x_{k-1},x_k]$. The subset of X_k which represents feasible choices for x_k is denoted by S_k. Since X_k is ordered, all choices in S_k are also ordered. With these assumptions the general backtracking algorithm can be stated as follows [24], where S_k is formed simultaneously by checking the feasibility of x_k.

We start by examining S_k (k=1,2,...,n) sequentially.

Step 1. If S_k is not empty, set x_k to be the smallest value in S_k which has not been tried previously. If k<n, increase k by 1 and repeat the step. If k=n, record the list as a feasible list. If we want all feasible solutions, decrease k by 1 and repeat this step. Otherwise stop.

Step 2. If S_k is empty and k=1, no more feasible lists exist, and therefore stop. If S_k is empty and k>1, decrease k by 1 and return to step 1.

During the design process, following a backtracking algorithm, usually we associate a cost with a particular list which satisfies a relationship of the following form:

$$\text{cost of } [x_1,x_2,..,x_{k-1}] <= \text{cost of } [x_1,x_2,..,x_{k-1},x_k] \text{ for all } x_k \qquad (3)$$

During searching we do not branch from a node whose cost is higher than the minimum cost solution found so far. Of course, the bound is updated if a better solution is found. For a maximization problem we do exactly the opposite. We do not branch from a node whose value is less than the value of the maximum value solution. This corresponds to a branch and bound algorithm that usually prunes the search.

In processing code requirements, usually we have to satisfy a conjunction or disjunction of more than one demands on the structure. This can be accomplished using a sequential backtracking algorithm, which is based on the structure of a Horn clause, and has the following form:

$$A \rightarrow P_1, P_2, \ldots, P_n, \quad n > 0 \qquad (4)$$

The important feature of Horn clauses is that they can be read both declaratively, saying that A is true if P_1 and P_2 and... and P_n is true, and procedurally, saying that to solve problem A, one can solve subproblems P_1 and P_2 and ... and P_n. In order to establish whether A is true, a Prolog program attempts to prove that P_1 and P_2 ... and P_n are true, starting from left to right. It uses the internal unification routines of the language as a form of a bi-directional pattern matching. It assigns values to the variables of the rule P_1 in order to prove that it is true and subsequently moves to P_2 leaving a marker in between to remember that up to that point the truth of clause A has been established. Hence, the program, by invoking the internal unification routines which sequentially search for the right values of the variables, establishes the truth of P_1 and P_2 and ... P_n, and thus the truth of A, binding in the mean time the feasible list of design variables. All rules are of the form given in relation (4) where A is referred to as the head of the rule and the conjunction of P_1 and P_2 and ... P_n as the body of the rule. Every subrule P_i is a predicate or categorim which is either true or false.

In order to restrict backtracking and thus reduce unnecessary search, the predicate "cut", or symbolically "!", is used which in effect deletes the markers for backtracking up to that point and

thus works as a barrier that doesn't allow backtracking to previous subgoals. The predicate "fail" of Prolog is one that makes the rule to fail and thus causes backtracking.

Following the sequential backtracking algorithm, binding of the list of variables takes place and affects the subsequent search. This issue is very critical and must be handled with care during the development of the system, in relation with the structure of the databases used. As discussed before in the procedural description of the backtracking algorithm, the feasible sets must be ordered. Therefore, if the database is built in increasing order with respect of the design objective, and the user has selected a number of different types of sections, a number of alternative solutions will be deduced. The existing conflict can be resolved either on the basis of secondary criteria, i.e. list of preferences in profiles, or other technological constraints, or by retaining all the solutions.

The logic approach has been used in the design of steel roofs using Eurocode 3 [19], where also the analysis is based on a logical program. The conformance checks used for the ultimate limit states correspond to tension, compression, bending, and bending with shear checks.

The code requirements can always be posed in the form of Horn clauses as for example the compression requirements of Eurocode 3. Compression members are designed according to the requirements of paragraph 5.4.4 and 5.5.1 of Eurocode 3 for buckling resistance. The relevant part in Prolog is as follows:

```
compression_requirements(Type,Cnt,_,Min,Length,Iarea):-
        section(Type,Name,Height,Width,Thickness,Area,Weight,Jx,_),
        chosen_section(_,_,_,WeightOld),
        Cnt * Area >= Iarea,
        Cnt * Weight < WeightOld,
        Tmp = Jx / Area,
        Radius_of_gyration=sqrt(Tmp),
        buckling_curve_axis(Type,Height,Width,Thickness,Curve,_),
        imperfection_factor(Curve,Alfa),
        Lamda=Length/Radius_of_gyration,
        Lamda1=pi*(sqrt(modulus_of_elasticity/yield_stress) ),
        Lamda2=(Lamda/Lamda1)*sqrt(Alfa),
        F=0.5*(1+Alfa*(Lamda2-0.2)+Lamda2*Lamda2),
        X=1/(F+sqrt(F*F-Lamda2*Lamda2)),
        NcRd=Cnt * Area * yield_stress,
        NbRd=X*NcRd,
        abs(Min) <= NbRd,
        CntArea = Cnt * Area,
        CntWeight = Cnt * Weight,
        save_chosen_section(Type,Name,CntArea,CntWeight),fail.
compression_requirements(_,_,_,_,_,_):- !.
```

The names of the variables are chosen close to the names of the quantities in the code to facilitate understanding of the relations. The entire process works by interrogating the database "section" with the attributes of all the profiles. This relational database, that Prolog language accommodates, is used in the searching of a new section as compared to a previous one. After the comparison of a pair of sections, the process selects the best section and fails at the end to force searching of the entire database for the particular group of members. The descriptive part in this provision corresponds to the selection of a buckling curve according to the type of section, hollow, welded rolled, etc. and the thickness to height ratios listed in Table 5.6 of the code. Therefore, the above predicate does not only perform the relevant conformance check but is used to select the optimal section within a specific type of profile.

Thus, a conformance check based on the provisions of a code and the selection of the optimal section with respect of a design objective, are rather straightforward for a given member. The important issue though is the identification of the required list of conformance checks for a member or a list of members. This issue was first addressed by S. Fenves introducing the decision tables which can be implemented efficiently in Prolog. The main difficulty faced in this respect is the vague information contained in a code that rarely specifies the domain of application of particular checks especially under exceptional environmental conditions. The list of actions and relevant conformance checks that are referred to a specific component form the design list for the component. In this work this list is fixed, but can result from the consistency checks of the code as a separate task.

The conformance checks can be applied to an already analysed structure of given dimensions, layout member sizes and loads. The way of incorporating the code provisions to the design process is a more general problem and relates to the strategy used in altering the structural configuration. This can be addressed in the context of structural optimization methods incorporating the constraints imposed by the codes of practice. In a logical approach, these can be stated in the form of a sequence of demands imposed on the design. These demands can be expressed directly in the form of Horn clauses given in relation (4).

The area of the so-called technical code based structural optimization is seeking to define the optimal design of a structure with respect to a single or multi criterion objective, subject to the constraints imposed by the selected code. In this context, the identification of the list of active constraints-provisions that correspond to particular conformance checks has a particular importance. It can be used as an active set to evaluate the sensitivities of the design with respect of the design variables and thus, depending on the form of optimization problem, alter the lay-out, shape or sizing of the components of the structure. Moreover, for a number of iterations it can be used to limit the design process to local modifications and therefore accelerate the entire design process. Therefore, code requirements can be interpreted not only as barriers on the design, but also as boundaries reflecting important information about the response of the structure. The identification of the active constraints can be performed easily in Prolog.

The proposed general scheme has been used for the design of a roof of an industrial building or warehouse using plane trusses and continuous purlins [19].

Although the system described above can be transported to other applications developed within the same design philosophy, is not totally independent from the remaining modules of the entire system. A more formal representation scheme, as for example the one based on the object oriented paradigm presented by Garrett et. al. [21] can stand as an independent module. In that case, a quite cumbersome interface must be built to provide information needed for the optimization process. A similar situation exists in structural optimization programmes. Some of which use a separate module to calculate sensitivities and others use a combined approach. The differences are in the software development but affect also the computation time of the system, the combined one being faster.

5. SUMMARY AND CONCLUSIONS

A general scheme based on logic programming is employed in the representation of code provisions. The control of the flow of information, that is the dominant part of the conformance checks and the design procedure of a structure, is based on the sequential backtracking algorithm used in Prolog as a searching algorithm. The design space of a particular problem is formed as the product of the relational databases of the members of a structure and the constraints. Even though logic programming has been used extensively in code representation and processing the unified scheme presented addresses the overall problem within the structural optimization perspective of structural design.

REFERENCES

[1] EUROCODE NO 3: "Design of Steel Structures," Part 1, General Rules and Rules for Buildings, Final Draft, Issued to Liaison Engineers, February 1989.

[2] JOINT COMMITTEE ON STRUCTURAL SAFETY, CEB - CECM - CIB - FIP - IABSE - IASS - RILEM: "General Principles on Reliability for Structural Design", International Association for Bridge and Structural Engineering (IABSE), 1981.

[3] HWANG, H.H.M., USHIBA, H., and SHINOZUCA, M., "Reliability Analysis of Code-Designed Structures under Natural Hazards", Technical Report NCEER-88-0008, 1988.

[4] HWANG, H.H.M., and HSU, H-M., "A Study of Reliability-Based Criteria for Seismic Design of Reinforced Concrete Frame Buildings", NCEER-91-0023, 1991.

[5] FENVES, S.J., "Tabular Decision Logic for Structural Design", ASCE Journal of Structural Division, Vol. 92, No. ST6, December, pp. 473-490, 1966.

[6] HARRIS, J.R., and WRIGHT, R.N., "Organization of Building Standards: Systematic Techniques for Scope and Arrangement", Building Science Series NBS BSS 136, National Bureau of Standards, Washington, D.C., 1980.

[7] FENVES, S.J., and WRIGHT, R.N., "The Representation and Uses of Design Specifications", NBS Technical Note 940, 1977.

[8] GARRETT, JR.J.H., and FENVES, S.J., "Knowledge-Based Standard-Independent Member Design", ASCE J. of Structural Engineering, Vol. 115, No. 6, pp. 1396-1411, 1989.

[9] RASDORF, W.J., and WANG, T.E., "Generic Design Standards Processing in an Expert System Environment", ASCE Journal of Computing in Civil Engineering, Vol. 2, pp. 68-87, 1988.

[10] ROSENMAN, M.A, and GERO, J.S., "Design Codes as Expert Systems", Computer-Aided Design, 17(9), pp. 399-409, 1986.

[11] CRONEMBOLD, J.R., and LAW, K.H., "Automated Processing of Design Standards", ASCE Journal of Computing in Civil Engineering, Vol. 2, pp. 255-273, 1988.

[12] TOPPING, B.H.V., and KUMAR, B., "Knowledge Representation and Processing for Structural Engineering Design Codes", Engineering Applications of AI, Vol. 2, No. 3, pp. 214-228, 1989.

[13] BEDARD, C., and GOWRI, K., "Automatic Building Design Process with KBES", ASCE J. of Computing in Civil Engineering, Vol. 4, No. 2, April, pp. 69-83, 1990.

[14] TYSON, T.R., "Effective Automation for Structural Design", ASCE Journal of Computing in Civil Engineering, Vol. 5, No. 2, April, pp. 132-140, 1991.

[15] BRATKO, I., "PROLOG Programming for Artificial Intelligence", 2nd Edition, Addison-Wesley Publishing Co., 1990.

[16] STERLING, L. and SHAPIRO, E., "The Art of Prolog: Advanced Programming Techniques", MIT Press, Cambridge, Massachusetts, 1986.

[17] FEIJO, B., DOWLING, P.J., and SMITH, D.L., "Incorporation of Steel Design Codes into Design Automation Systems", Expert Systems in Civil Engineering, IABSE Colloquium, Bergamo, 1989.

[18] FEIJO, B., KRAUSE, W.G., SMITH, D.L., and DOWLING, P.J., "A Hypertext Model for Steel Design Codes", J. of Constructional Steel Research, Vol. 28, 167-186, 1994.

[19] KOUMOUSIS, V.K., and GEORGIOU, P.G, "An Expert System for Steel Roof Design", Structural Engineering Review, Vol. 5, No. 2, pp. 169-181, 1993.

[20] GARRETT, JR., J. H. and S.J. FENVES, "A Knowledge-Based Standard Processor for Structural Component Design", Engineering with Computers, 2(4) pp. 219-238, 1987.

[21] GARRETT, JR. H. and M. MAHER HAKIM, "An Object-Oriented Model of Engineering Design Standards", Journal of Computing in Civil Engineering, 6(3), pp. 323-347, 1992.

[22] JAIN D., K. H. LAW and H. KRAWINKLER, "On Processing Standards with Predicate Calculus", Proc. of the Sixth Conference on Computing in Civil Engineering, ASCE, Atlanta, GA, 1989.

[23] RASDORF, W.J. and S. LAKMAZAHERI, "Logic-Based Approach for Modelling Organization of Design Standards", ASCE Journal of Computing in Civil Engineering, 4(2), pp. 102-123, 1990.

[24] HU T. C. , "Combinatorial Algorithms", Addison-Wesley Pub., pp. 138-151, 1982.

[25] FENVES, S.,GARRETT, J.,HAKIM, M., "Representation and Processing of Design Standards: A Bifurcation between Research and Practice", 1994 ASCE Structures Congress, Atlanta, GA.

Leere Seite
Blank page
Page vide