Generic Design Standards Processing in an Expert System Environment

William J. Rasdorf[1], M. ASCE and TsoJen E. Wang[2]

Abstract

Checking design conformance with governing standards, codes, specifications, regulations, and manuals is a mandatory step in designing engineering systems. Automating this design checking process, therefore, is an important goal that has far-reaching design productivity implications. Current conformance checking automation generally involves the use of application programs into which the standards, or an interpretation of their intent, have been directly coded. This approach is extremely inflexible and often error-prone. This paper investigates the feasibility of casting design standards in an alternative form - one suitable for flexible processing in a knowledge-based expert system environment. The proposed form not only enables a designer to readily check the conformance of designs, but it also allows him to evaluate the effects of changing design variables as well as evaluate the impact of down-stream activities and events, such as change orders, on the initial design.

Two prototype standards processing systems utilizing the production system approach have been constructed and are described herein. Although the obvious direct translation from a design requirement in a standard to a rule in a rule-based expert system has its advantages, a more generic and flexible computer representation for design standards is desirable. The approach advocated in this paper is to represent standards as databases of facts which are used on an as-needed basis by a rule-based design conformance checking program to monitor design evolution. The representation is derived from a unified view of standards obtained by using the standards modeling tools proposed by previous researchers in this field during the past decade. A generic knowledge-based standards processing architecture is proposed and described. An implementation of this architecture is also discussed.

---

[1]Associate Professor of Civil Engineering and Computer Science, North Carolina State University, Box 7908, Raleigh, NC 27695.

[2]Systems Engineer, Micro Computer Systems, Inc., 4405 East West Highway, Suite 510, Bethesda, MD 20814. Formerly a Graduate Research Assistant, Department of Electrical and Computer Engineering, North Carolina State University, Box 7911, Raleigh, NC 27695.

1 Introduction

1.1 Objectives

Design standards play an important role in the design of building, bridges, and other engineering systems. A design configuration must be checked against all applicable standards to ensure that it is acceptable. Throughout this paper, the term **standards** will be used to represent design standards, codes, specifications, regulations, and similar legal documents defining the proper design for a system.

In the past, standards were interpreted and converted into application programs written in procedural programming languages such as FORTRAN. The first objective of this study was to investigate the feasibility of casting standards in a different representational framework, namely the knowledge-based expert system environment. The purpose of exploring this approach was to determine its utility in the standards processing environment. Such utility had been suggested and anticipated but had not yet been demonstrated.

The second objective was to develop a single generic standards processing system which performs design conformance checking and provides allowable value ranges for undetermined data. The term *generic* implies that the system must be independent of a particular design standard. The separation of the standards processing capability from the representation of a standard is essential to achieving the following advantages:

- Standards governing different domains and even different design disciplines can be processed using a single generic system.
- A standard needs to be transformed from text into the internal format processible by the system only once. Hence the transformation can be done by a group of experts in a systematic way.
- Existing standards can be updated by simply replacing the portion(s) revised.
- Only those portions of a standard(s) that are applicable at a given time during design need to be processed and these can be selectively chosen by the designer.

1.2 Background

Previous research on design standards has been conducted to improve: (1) the representation and organization of standards, (2) the analysis of standards, and (3) the use of standards. The following three sections summarize selected research results from these areas. An additional section discusses the problems associated with existing standards processing software.

The synthesis of standards, which deals with creating new standards, is another noteworthy research area. But because the main theme of this paper concerns the use of standards

rather than their creation, synthesis is not addressed herein.

1.2.1 Representation of Design Standards

Standards are usually organized as a collection of categorized provisions [8]. Each provision contains information dealing with a specific design property. If the evaluation of a provision yields a value of *satisfied* or *violated*, such a provision is referred to as a **requirement**. Those provisions that are not requirements are called **determinations** [14]. The basic element found in a design standard is **datum**. A datum is a variable representing one of the following incidences in a standard: (1) the status of a requirement, (2) the result of a determination, or (3) an ingredient needed to evaluate a requirement or determination. The set of data involved in a design standard plus a representation of the interrelationship among the data comprise the necessary information used in checking design conformance with the standard. As a result of more than a decade of research, standards are most often modelled using three tools: decision tables, information networks, and an organization system [8,14,18]. The following paragraphs briefly describe each of these tools.

A **decision table** is used to represent complex antecedent-consequence relationships among datums. Figure 1 shows a **limited-entry decision table** corresponding to a portion of Table 910 of the Building Officials and Code Administrators (BOCA) Building Code [2]. The upper-left portion of the table contains a set of conditions which consist of datum-value expressions. The upper-right portion indicates whether (Y) or not (N) a condition is satisfied. The lower-left portion contains a set of actions to be performed. The lower-right portion indicates which action is to be taken if all of the corresponding conditions in a given column are satisfied. As opposed to limited entry decision tables, the condition/action statements in an **extended-entry decision table** contain only datums themselves; the associated values are recorded in matching entry positions [6]. Figure 2 illustrates an extended-entry decision table.

An **information network** is a collection of nodes and branches where each node represents a distinct datum and each branch represents a relationship between two nodes. The datum on top of a branch is commonly referred to as **parent** and the lower level one is referred to as a **child**. A datum may have more than one child and more than one parent. If a datum does not have a child, it is a **basic datum**. Those datums that have at least one child are called **derived datums**. For each network, there should be exactly one derived datum that has no parent; it is the **terminal datum** of the network. In any case, all of the children must be present before a derived datum can be evaluated, this is called the **precedence relationship** that must be observed in an information network. Figure 3 shows an information network corresponding to Provision 910.2 of the BOCA Building Code.

The basic element in an **organization system** is referred to as a **classifier**. Classifiers indicate the subject involved in requirements and are cast in the following form:

≪subject≫ shall have ≪required quality≫

A_t: Roof tributary loaded area
L_{ur}: Minimum roof live load
L_s: Roof snow load

roof-shape = pitched	Y	Y	Y
$0 <$ roof-slope < 4 in/ft	Y	Y	Y
$0 \leq A_t \leq 200$ ft^2	Y	N	N
$200 < A_t \leq 600$ ft^2	N	Y	N
$A_t \geq 600$ ft^2	N	N	Y
$L_{ur} \geq 20$ psf	X		
$L_{ur} \geq 16$ psf		X	
$L_{ur} \geq 14$ psf			X

Figure 1: Limited-Entry Decision Table (Table 910 of BOCA)

roof-shape	pitched		
roof-slope (in/ft)	> 0		
	< 4		
A_t (ft^2)	≥ 0	> 200	> 600
	≤ 200	≤ 600	
L_{ur} (psf)	≥ 20	≥ 16	≥ 14

Figure 2: Extended-Entry Decision Table (Table 910 of BOCA)

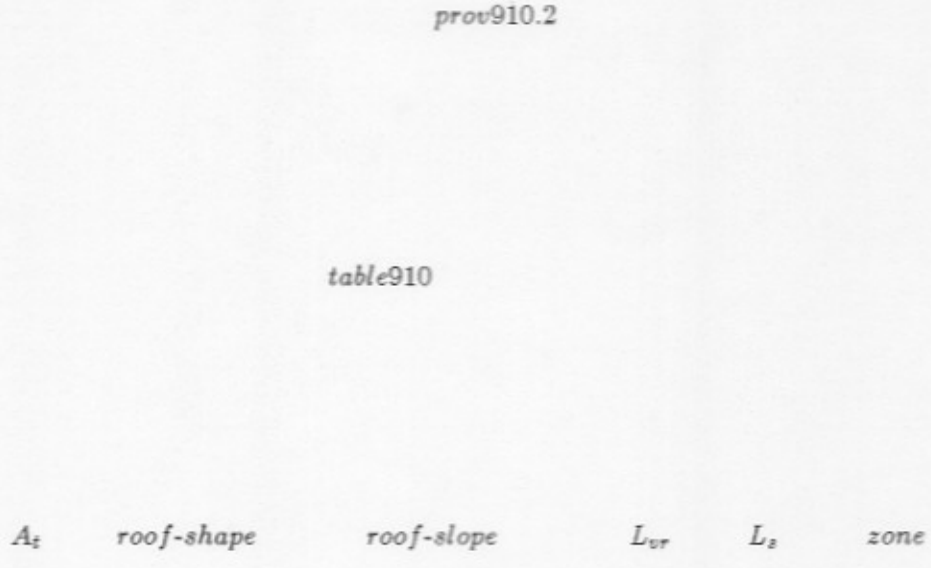

Figure 3: Information Network (Provision 910.2 of BOCA)

Every classifier has a **scope list** which is the set of all requirements associated with it. In a like manner, every requirement has an **argument list** which is the set of all pertinent classifiers. An assembly of classifiers constitutes a **classifier tree** wherein each node represents a classifier and each branch represents the precedence relationship between two nodes. Classifiers can be listed in two distinct ways: using an index or an outline. An **index** is formed by listing the classifier/scope list pair alphabetically. A requirement thus has multiple entries in an index. An **outline** is obtained by ordering the requirements according to their relative positions in the corresponding classifier tree. Hence each requirement only appears once in an outline. Figure 4 illustrates a portion of an organization system corresponding to Section 910.0 of the BOCA Building Code.

1.2.2 Analysis of Design Standards

Although design standards are written by domain experts, they are not necessarily cast in the best format possible when they are created. The purpose of analyzing existing standards is to pinpoint where weaknesses, omissions, or contradictions might be located, thus providing a solid basis for further improvement. Among the three modelling tools for standards, decision tables provide the most complete and precise description of provisions. The use of decision tables ensures the completeness (every possible situation is covered) and clarity (no redundant or contradictory entry) of standards. Unfortunately, the scope of individual decision tables is limited. To represent a provision, for example, several decision

Scope List of the classifier "L_{ur}:"

(Table 910, Provision 910.2, Provision 910.3, Provision 910.5)

Argument List of "the BOCA Building Code Table 910:"

(A_t, roof-shape, roof-slope, L_{ur})

Outline:

Section 910.0
- Provision 910.1
- Provision 910.2
  - Table 910
- Provision 910.3
- Provision 910.4
- Provision 910.5
  - Provision 910.5.1
  - Provision 910.5.2

Index:

Figure 4: Organization System for Section 910.0 of BOCA

tables may be required. On the other hand, an information network can readily display the ingredients needed to evaluate all the derived datums in a provision. Nevertheless the exact functional relationship among these datums is not evident in an information network. Recognizing the need to integrate the representation of decision tables and information networks, Rasdorf developed a mechanism to perform information subnetwork generation and compression [18]. **Subnetwork generation** converts a decision table, which represents a single information network node, into multiple lower level nodes in the information network. **Subnetwork compression** converts all of the information contained in two or more related subnodes into a single higher level node.

1.2.3 Use of Design Standards

Research on the representation and analysis of design standards has the goal of enhancing the completeness and clarity of standards. Such objectives are essential for enabling designer to make better use of standards. Because standards are usually presented in textual form, the interpretation of the meaning of the provisions of a standard is often subject to the experience of the user. The modelling tools described in Section 1.2.1 enable users as well as analysts to gain a better perspective of standards. Indexes and outlines provide reference guidance; decision tables and information networks prescribe the functional relationships among datums.

As discussed in Section 1.2.2, a standard can be viewed as a network of decision tables. The process of checking a design against a standard involves locating the governing decision tables in the decision table network and evaluating the prescribed datum relationships. Depending upon how the decision tables are evaluated, the design conformance checking process can be carried out at three levels [9].

At the lowest level, the governing provisions are manually translated into code in a Computer-Aided Design (CAD) application program using a procedural language such as FORTRAN. The completed program accepts as input a list of data items that describes the current design and operates on them in a predetermined program execution sequence. Although this *hard-coding* approach has been widely used by the CAD industry, it has serious drawbacks which are discussed in Section 1.2.4.

At another level, rather than doing it manually, the translation from provisions to program code can be done by a preprocessor. The CAD application programs thus generated operate in a manner similar to manually coded programs and possess the same drawbacks.

At the third level, a new generic standards processor can replace the application programs used for design conformance checking. Provisions need no longer translated into program code. Instead, the network of decision tables and the list of data item values can be directly fed into a processor for evaluation. It is this standards processing structure that is proposed herein and is discussed in detail in Section 3.

1.2.4 Problems with Existing Design Standards Processing Software

A large number of CAD programs have been written by directly embedding provisions of governing standards into the program logic. ICES/STRUDL (Integrated Civil Engineering System/STRUctural DEsign Language) and GENESYS (GENeral Engineering SYStem) are two of the more widely used, automated structural design software packages utilizing this so-called *hard-coding* approach [25]. Just as the use of standards is subject to interpretation so too is the coding of standards. In most cases, the programmer of a CAD program is not the writer of the governing standard, thus there is a chance of misinterpretation. It is obvious that if a CAD program contains a misinterpretation error, the program will not perform its prescribed conformance checking function correctly. This is a serious concern.

The fact that standards may be misinterpreted makes the hard-coding approach less than optimal. Even if the correctness of interpretation is guaranteed, a more severe handicap of the approach deals with its ability to accommodate change. When the standard associated with a program is updated (including insertion of new requirements, revision of existing requirements, and deletion of obsolete requirements), the portions of the program incorporating the affected provisions must also be updated in order to accommodate the changes. Such revision of software may be difficult and expensive, especially with programs of considerable size.

Furthermore, if a designer needs to consult standards that are different from the one coded into the current software package, a new program must be written or purchased. Incorporating new programs into an existing CAD system can also be difficult and expensive. Even if the difficulties of doing so can be overcome, the time spent waiting for the new program to be designed and developed may be too great.

In light of the problems discussed here (which are summarized in Figure 5), the current approach of hard-coding standards into CAD programs is agreed to be far from ideal. The development of a standards processing scheme that attempts to overcome some of these limitations is the main theme of this paper.

2 Two Prototype Design Standards Processing Expert Systems

To date, numerous expert systems have been built to explore applications in domains as varied as medicine, law, geology, engineering, military, and agriculture [5,15]. Waterman in [30], *A Guide to Expert Systems*, reports over 160 expert systems that have reached at least the stage of a research prototype. For example, HI-RISE assists engineers in preliminary structural design by selecting feasible building design configuration(s) based on a set of constraints [17]; SACON helps engineers determine an appropriate finite element structural analysis strategy [3]; SPERIL-I diagnoses earthquake damage of existing structures [12]; and Load-Consultant determines the loads imposed on structures according to building

Problems	Cause	Results
Misinterpretation	standards are not interpreted by the original authors	the checking process is based on erroneous information
Lack of Modularity	sections of standards are directly coded into programs	both updating old and incorporating different sections are difficult
Too many programs	different programs have to be written for different applications	high demand in man-power, computing resources

Figure 5: Problems with Existing Design Standards Processing Software

code requirements. Figure 6 lists several existing knowledge-based expert systems (KBESs) used in Civil Engineering applications. The list is by no means complete.

The potential of KBES methodology in engineering applications has been given positive recognition [10,20,22,24]. More specifically, other research results indicate that a KBES is a practical tool for standards processing applications [13,19,23]. The feasibility of casting engineering design standards in a KBES environment is the subject of this paper. This topic was initially investigated by building two standards processing KBESs: **Query Monitor**, which addresses the issue of the semantics of data retrieval from engineering databases; and **Roofload Checker**, which performs design conformance checking utilizing a standard.

2.1 Query Monitor

The AISC Specification addresses a number of different types of stresses which can occur within a structural steel member including tension, shear, compression, bending, and bearing [1]. Depending upon constraints on shape, cross-section, loading, etc., any one of a number of equations can be used to determine the allowable stress for a specific structural steel member. Figure 7, for example, lists eight equations applicable to calculating allowable bending stress. Determining which equation is applicable is the difficulty presented by the specification. A database problem arises when the engineer issues an F_b data retrieval request. The response to such a query requires a determination of which F_b is applicable and involves the subsequent derivation of its value. Query Monitor was proposed as a framework to combine a database with a set of design specification constraints that govern the retrieval of data from engineering databases [21].

System	Developer	Institution	Function
HI-RISE	Maher	CMU	assists engineers in preliminary structural design configuration
SACON	Bennett	Stanford U	helps engineers determine proper finite element analysis strategy
SPERIL	Yao & Futura	Purdue U	diagnoses earthquake damage of existing structures
Load Consultant	Rasdorf & Chern	NCSU	determines the loads imposed on structure according to codes

Figure 6: Existing KBESs in Civil Engineering

$$F_b = 0.60F_y$$

$$F_b = 0.66F_y$$

$$F_b = 0.75F_y$$

$$F_b = \frac{12 \times 10^3 C_b}{ld/A_f}$$

$$F_b = \frac{170 \times 10^3 C_b}{(l/r_T)^2}$$

$$F_b = F_y \left[1.075 - 0.005 \left(\frac{b_f}{2t_f}\right) \sqrt{F_y}\right]$$

$$F_b = F_y \left[0.79 - 0.002 \left(\frac{b_f}{2t_f}\right) \sqrt{F_y}\right]$$

$$F_b = \left[\frac{2}{3} - \frac{F_y (l/r_T)^2}{1530 \times 10^3 C_b}\right] F_y$$

Figure 7: Formulas to Determine Allowable Bending Stress

$Axis = major$	Y	Y	Y	Y	Y	N
$ConOK$	Y	N	Y	Y	Y	Y
$B_f\_T_f\_RatioOK$	Y	Y	N	Y	Y	Y
$D\_T\_RatioOK$	Y	Y	Y	N	Y	Y
L_cOK	Y	Y	Y	Y	N	I
$F_b = 0.66 * F_y$	X					X
$Except\_Ten$		X	X	X	X	
$Except\_Comp$		X	X	X	X	

Figure 8: Decision Table for AISC Specification Provision 1.5.1.4

A prototype implementation of the Query Monitor architecture was developed using the M.1 expert system building tool [26]. The knowledge representation scheme used by M.1 consists of production rules and facts. Its inference engine utilizes a goal-driven control strategy. The rules in Query Monitor were directly converted from a number of decision tables corresponding to Provision 1.5.1.4 of the AISC Specification. As an example, consider the decision table of Figure 8 which is one of the tables from Provision 1.5.1.4. The first column of the table was recast in production rule format as follows:

If the axis about which a member is being bent = major and
the connection of the web and flange is continuous and
the width thickness ratio for exceptions is ok and
the depth thickness ratio is ok and
the laterally unsupported length is ok
Then the allowable bending stress = $0.66F_y$

A complete program listing as well as several sample execution logs of sessions can be found in the Query Monitor User's Guide [27].

2.2 Roofload Checker

The **Roofload Checker** incorporates a portion of the BOCA Building Code [2] to check the design of building roofs for conformance with its provisions. The Roofload Checker was developed to study the performance of a production system employing a data-driven control strategy to check designs. It consists of two subprograms, **Roof Checker** and **Roof Reporter**. The components as well as the operation of Roof Checker are similar to a typical KBES. The engineer describes the roof design using datum-value pairs. These data are stored in the context. Roof Checker then matches the input against the production rules to determine whether or not the design conforms to the standards it incorporates. The test runs showed that Roof Checker is capable of checking a roof design according to

the knowledge it possesses, i.e., the production rules converted from the BOCA Building Code. Nevertheless, Roof Checker, which is a pure data-driven system, does not provide any feedback after its operation. Therefore, another KBES called Roof Reporter was developed to translate the checking results from the internal KBES representation to a form suitable for monitor display.

After the checking of a design is completed by Roof Checker, the checking result is stored in an external file. After Roof Reporter is invoked, the file containing the checking result is read and stored in the context. The data are then reformatted and displayed on the monitor screen. Several sample execution logs of Roofload Checker sessions can be found in Reference [29].

Roof Checker and Roof Reporter were written in the OPS5 knowledge engineering language [11]. The knowledge representation scheme used by OPS5 consists of production rules. Either the data-driven or the goal-driven control strategy can be implemented in OPS5. The Roof Checker incorporates, among other things, Table 910 of the Code. As an example, the requirements of Table 910 are directly cast in production rule format in the Roof Checker as follows:

If	the shape of the roof = pitched and
	$4 \leq$ the slope of the roof < 12 in/ft and
	$0 \leq$ the tributary loaded area for structural member < 200 ft^2 and
	the designed roof load ≥ 16 psf
Then	the roof is OK

A complete program listings of the Roof Checker as well as the Roof Reporter can be found in Reference [29].

Both the Query Monitor and the Roofload Checker deal with the use of standards. On one hand, the Query Monitor addresses the semantic issue of retrieving standards knowledge from engineering databases. On the other hand, the Roofload Checker addresses the issues involved in utilizing the knowledge of design standard to check whether a roof design conforms to the governing standard's provisions.

2.3 Observations

The two prototypes introduced in Sections 2.1 and 2.2 demonstrate that standard provisions can be cast as production rules. The major advantages of casting standards as production rules are the modularity and the direct correspondence between decision table rules and production rules. The rules are separate from CAD programs, thereby enhancing the ease of updating and maintaining the standard. Modularity also faciliates the execution of the processing system because a standard can be partitioned into blocks of the proper size for optimal performance in terms of pattern matching and computer memory management.

Based on our experience in developing the prototypes discussed above we found that

although the approach of casting standards as production rules has its merits, it is not currently practical because the resulting knowledge base contains so many rules that processing time is prohibitive. For instance, the Query Monitor incorporates over one hundred rules representing only a portion of Section 1.5.1.4 of the AISC Specification. The Roofload Checker incorporates thirty rules to represent Table 910 of the BOCA Building Code. The large number of rules is due to the fact that the translation from a decision table column to a production rule is, for the most part, a one-to-one mapping and the large number of decision table rules result in a large number of production rules.

Regardless of how concise a production rule may be, the collection of rules representing an entire standard will inevitably occupy an unmanageable portion of computer memory. In addition, the construction of such a knowledge base demands a significant effort from the developer, even with the help of a knowledge acquisition facility, and especially if the standard is not already cast in decision table form.

Section 3 presents an alternative standards processing system, based on a factual representation scheme, that has the potential to overcome the drawbacks of casting standards as production rules.

3 A Generic Knowledge-Based Standards Processing Architecture

The research being reported in this paper resulted in the development of a generic knowledge-based standards processing system that performs design conformance checking and provides allowable value ranges for undetermined design components. This section presents an architecture for such a system. Section 4 then describes an implementation of the architecture.

3.1 Factual Representation of Standards

As opposed to the popular notion of directly casting standard's provisions as production rules, this paper advocates an alternative approach of representing standards as provisional and organizational facts. A **provisional fact** corresponds to the combination of a datum and its associated decision table condition entry. Thus, it is relevant to either a requirement or a determination. 'The minimum roof load shall be 12 psf', for example, is a provisional fact. Figure 9 illustrates a set of provisional facts. An **organizational fact**, such as 'Table 910 is included in Provision 910.2', represents the hierarchical relationships among the datums of a standard.

All the provisional facts related to the same decision table column are loosely grouped together by a common attribute. Because there exists no explicit precedence among provisional facts, they can be manipulated by *one and only one* concise set of generic, standard-independent production rules! On the other hand, there are precedence relationships among organizational facts and these must be observed in a standard-dependent manner. Never-

Decision Table for a Portion of Table 910 of the BOCA Building Code

roof-shape	pitched
roof-slope (in/ft)	≥ 4
	< 12
the tributary loaded area for structural member (ft^2)	≥ 0
	< 200
the designed roof load (psf)	≥ 16
the roof is	OK

Factual Representation for the Table 910 Decision Table Rule

the shape of the roof = pitched
$4 \leq$ the slope of the roof < 12 in/ft
$0 \leq$ the tributary loaded area for structural member < 200 ft^2
the designed roof load ≥ 16 psf

Figure 9: Illustration of the Factual Representation of a Standard

theless, all of the data of a standard, both basic and derived, are represented explicitly in a well-defined information network. Since every datum can be located from the information network, the traversal of the network can also be handled by a single, concise set of standard-independent production rules.

The separation of facts from the rules that manipulate them is of great significance, resulting in a generic and modular system which (1) can be applied to any standard, and (2) requires one and only one set of rules to process a decision table regardless of its size or the standard from which it was derived. This flexibility cannot be achieved by the traditional approaches of casting standards as rules or hard-coding them into CAD programs. The representation is modular in the sense that it is uniform and in the sense that the facts can be partitioned into independent modules such that changes to one module does not affect the others. Because the number of modules and the size of each individual module involved in a process are not restricted, the representational scheme is to some extent dynamic. Furthermore, the explicit and separate representation of facts allows multiple accesses to the knowledge base for different purposes, e.g., indexing, query, and explanation.

The advantages of casting standards as facts are not limited to those described above. Additionally, the correctness of the interpretation of a standard is guaranteed as long as the underlying decision tables are correct. Also, since a standard needs to be converted into factual form only once and processed by a set of rules over an extensive period of time, the conversion from a standard to decision tables can be done by domain experts; thereby

insuring the correctness of interpretation of the standard. The programming involved in coding standards is thus reduced to the transformation from decision tables to factual knowledge bases. Because the proposed representational scheme is based on the unified view of design standards, such a transformation is relatively straightforward and can therefore be readily automated. This saving in programming efforts cannot be achieved if standards are cast in production rules or CAD programs.

In light of all of these merits, the approach of casting standards as facts is adopted as the knowledge representation in the generic knowledge-based standards processing architecture presented in the next section.

3.2 Architecture of a Generic Knowledge-Based Standards Processing System

This section presents the architecture of a generic knowledge-based standards processing system based on a factual representation of standards. This architecture serves two main purposes: (1) performing design conformance checking, and (2) determining allowable value ranges for undetermined design datums. Essentially, the architecture supports Standards Processing In a Knowledge-based Expert system environment and is referred to hereinafter by the acronym **SPIKE**. As shown in Figure 10, SPIKE contains the components of a typical knowledge-based expert system.

For its knowledge base, SPIKE utilizes provisional and organizational facts. To store organizational facts, pointers are used to preserve the links of the information network. To store provisional facts, relational operators are used to describe the requirements or determinations. Because the knowledge base is implemented in the factual format, it is called the **Standards Factbase** of SPIKE.

As in a typical KBES, the standards factbase is used by an inference engine as it manipulates the context. In SPIKE, the set of production rules encoded specifically for processing the generic standards factbase is referred to as a **Standards Processor**. The knowledge acquisition facility in SPIKE is termed a **Transformer**. The major task of the transformer is to translate the knowledge from the decision table format of a standard to the internal representation of the factbase. The **Context** is the short term memory containing design-specific information entered by either of the two interfaces (interactive and program) or generated by the inference engine. The **Interactive Interface** provides a command language to enable the designer to communicate with the system. The command language can be used to describe a design, to demand an explanation about system operation, or to query the system to obtain information about the design or the governing standard. The **Program Interface** provides a similar functionality for CAD programs. The next section discusses the implementation and operation of SPIKE in greater detail.

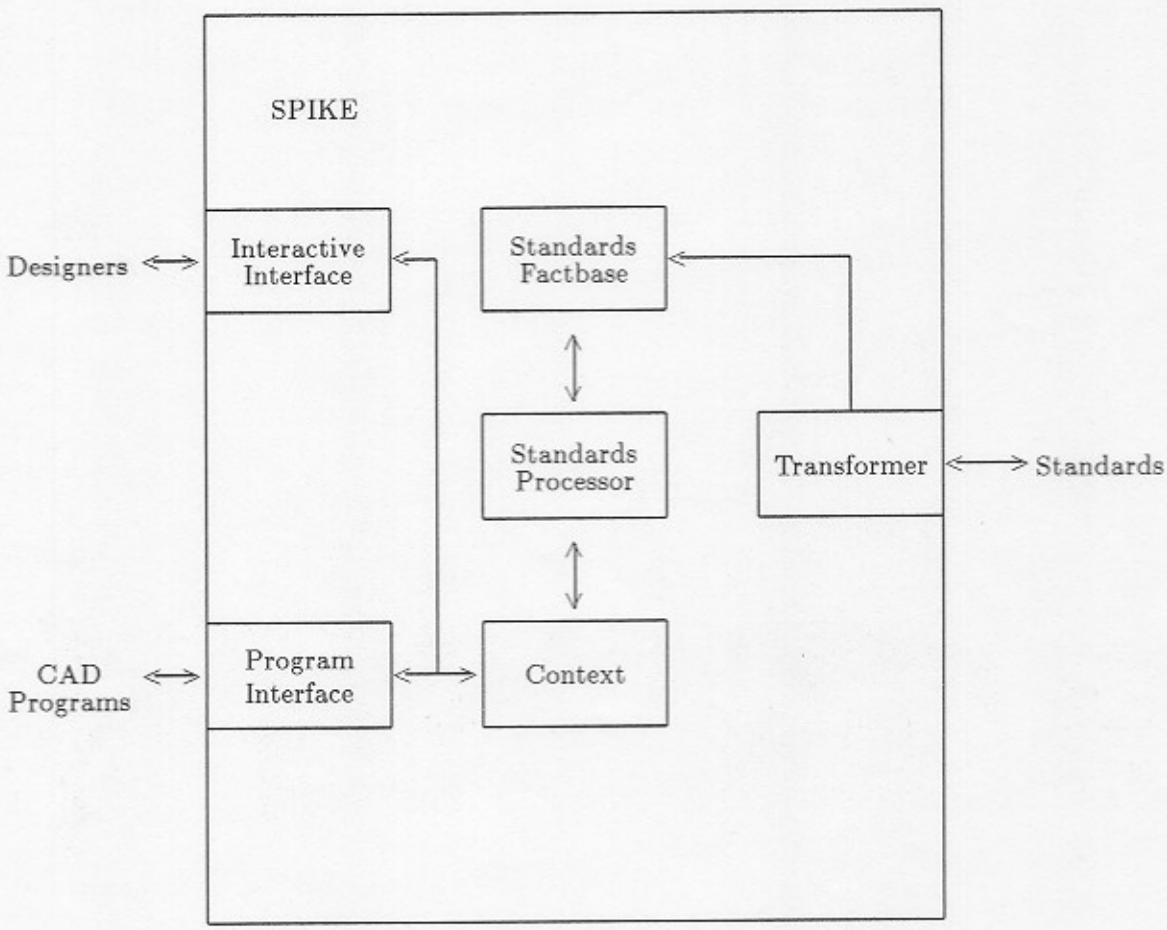

Figure 10: Architecture of the SPIKE Standards Processing System

4 Implementation of the Generic Knowledge-Based Standards Processing System

The generic knowledge-based standards processing architecture (SPIKE) introduced and described in Section 3.2 has been successfully implemented as a research prototype [28]. This section describes the implementation of the SPIKE architecture.

4.1 General Description

The SPIKE prototype is an interactive program that supports user communication via a terminal instead of through a program subprocess call. The user enters, as input, sets of datum-value pairs describing the design under review. SPIKE validates the input to prevent failures resulting from erroneous data (including misspelled datum names, out-of-range values, contradictory entries, etc.). In order for analysis to proceed the user must enter at least one datum-value pair. After the input has been validated, it is stored in the context.

When the user indicates there is no additional input (by hitting the carriage return without preceding data), SPIKE begins its analysis. Whether it performs design conformance checking, provides allowable value ranges, or both depends on the amount of input it has received for each provision, i.e., it depends on how complete the initial specification of the design was.

During the operation of SPIKE, depending on how much input data was given, derived datums may be generated and stored in the context, thus triggering further operations. After SPIKE completes its data generation, analysis, and conformance checking, the results are displayed on the screen. The user can then elect to quit or continue, revising the design by either entering known datums coupled with updated values or entering new datum-value pairs according to the recommendations made by the system in the previous session. Upon receipt of a known datum, SPIKE automatically replaces the old value with the new one. A new design will be checked as soon as the user indicates the end of input revisions. The cycle can be repeated as many times as necessary until a design is derived that completely conforms to the governing standard.

4.2 Implementation Language - OPS5

A knowledge engineering language—OPS5 has been used to implement the generic knowledge-based standards processing architecture presented herein. OPS5 was designed to be a development tool for building production systems [11]. Interpreters for OPS5 have been written in BLISS, MACLISP, and FRANZLISP. The one used to build SPIKE is a BLISS-based compiler installed on a VAXSTATION-II running the VMS operating system [7].

table910	OK			NG
zone	snow		no-snow	-
L_{vr} (psf)	$\geq L_s$	$< L_s$	-	-
prov910p2	OK	NG	OK	NG

Figure 11: BOCA Provision 910.2

4.3 Implementation of the Standards Factbase

The modularity of the SPIKE architecture is evident in the structure of its factbase. Prior to implementing the standards factbase, the **granularity** of the factbase, i.e. the size of each individual module that the system operates on, must be determined. The granularity adopted in the SPIKE prototype corresponds to the size of a section of the BOCA Building Code. The decision to choose this level of granularity was based on operating efficiency considerations and the belief that sections of the code were of sufficient size to demonstrate the principles proposed herein. For a system like SPIKE, the fewer the number of pattern-matchings, the faster a system will be. A section in most of the existing standards contains fewer than one hundred datums, thus appearing to be of reasonable size for an individual module.

4.3.1 Provisional Facts

The most important class for representing provisional facts in each module of SPIKE is cond:

```
(literalize cond
        item
        provid
        col
        opr1
        val1
        opr2
        val2)
```

where item records the name of a datum; provid refers to which provision the datum appears in; col is the column number in the governing decision table; opr1 records the relational operator $=$, $>$, or $\geq$; opr2 records the relational operator $<$ or $\leq$; and val1 and val2 are scalars that represents data item values. Each instance of an element of the cond class represents a provisional fact.

provision 910.1	OK
provision 910.2	OK
provision 910.3	OK
provision 910.4	OK
provision 910.5	OK
section 910.0	OK

Figure 12: BOCA Provision 910.0

As an example, consider the decision table shown in Figure 11 corresponding to Provision 910.2 of the BOCA Building Code. This decision table is represented by the following entries in the standard factbases of SPIKE:

```
(make cond ↑item table910 ↑provid prov910p2 ↑col 1)
(make cond ↑item zone ↑provid prov910p2 ↑col 1 ↑opr1 = ↑val1 snow)
(make cond ↑item Lvr ↑provid prov910p2 ↑col 1 ↑opr1 ¿= ↑val1 Ls)

(make cond ↑item table910 ↑provid prov910p2 ↑col 2)
(make cond ↑item zone ↑provid prov910p2 ↑col 2 ↑opr1 = ↑val1 no-snow)
(make cond ↑item Lvr ↑provid prov910p2 ↑col 2 ↑opr1 -)
```

As shown above, all the provisional facts in each column of the governing decision table are grouped together by the attribute col. Each provisional fact is explicitly represented by the combination of item, opr1, val1, opr2, and val2. Because the matching process in each recognize-act cycle is performed concurrently for all productions, the condition elements in each rule must be clearly specified. Therefore, two relational operators (opr1 and opr2) are provided to distinguish conditions like 'roof-slope $<$ 4 in/ft' from '4 $\leq$ roof-slope $<$ 12 in/ft'. When a datum is equal to a scalar value it is represented by item, the operator "=," and val1. A provision is considered to be satisfied when all the instances of cond with the same provid and col are satisfied simultaneously.

At times, the value of a datum may be immaterial to deciding the outcome of the decision table column to which it belongs. This situation is noted by the symbol "-" in opr1. Another matter worth noting is that of inapplicability, i.e., a datum may possess a value that is not confined by the standard. Section 910.3 of the BOCA Building Code, for example, contains requirements that must be satisfied when a building has roof projections. If the building being checked does not have roof projections, Section 910.3 is not applicable to this design. To deal with this situation, the SPIKE prototype records the datum in a class named index to support SPIKE's query capability, but the datum is not stored in the class cond.

4.3.2 Organizational Facts

The organizational facts of a standard are implemented using two classes: cond and index. As stated in the last paragraph, cond is used primarily to represent provisional facts that appear in decision table condition entries. If organizational facts are expressed in the form of decision tables, they too can also be represented using cond. For example, one rule of Section 910.0 of the BOCA Building Code can be expressed as shown in Figure 12. The organizational facts of the information network that are embodied in this decision table can be represented in cond as follows:

```
(cond ↑item prov910p1 ↑provid sec910 ↑col 1)
(cond ↑item prov910p2 ↑provid sec910 ↑col 1)
(cond ↑item prov910p3 ↑provid sec910 ↑col 1)
(cond ↑item prov910p4 ↑provid sec910 ↑col 1)
(cond ↑item prov910p5 ↑provid sec910 ↑col 1)
```

According to the definition of cond, when all the provisional facts in this group are satisfied, Section 910.0 is considered to be satisfied. When a provision is satisfied, a corresponding derived datum is created by the standards processor and stored in the context. The value OK is unnecessary because the standards processor recognizes the presence of a derived datum as the sign of satisfaction.

The index element class serves as a data dictionary for a standard. It is used by the standards processor to determine the relevance of each module with respect to the design under consideration. The definition of index is as follows:

```
(literalize index
      stand
      item
      tdb
      provid
      type
      legalvalue
      lowerbound
      upperbound
      unit)
```

where stand is the attribute identifying the current standard in use by the standards processor; item and provid are the same as in the cond element class; tdb is the individual module where the relevant decision table is stored; type of the datum is either basic (including both numeric and symbolic) or derived; and legalvalue stores the input validation information for symbolic data. Vector-attributes are not used for representing multiple levalvalues for a datum because it is impossible to retrieve the value of a vector-attribute on the LHS of a production (except using a set of awkward retrieving routines). Instead, they are stored in

several instances of the element class index. The input validation information for numeric data is stored in lowerbound and upperbound and the attribute unit records the appropriate measurement units for the datum.

The information network of a standard is represented by grouping the datums level by level using the attribute provid in the index element class. The method used is simply an implementation of pointers. Every direct descendent of a derived datum is recorded in the attribute item with the derived, parent datum stored in the attribute provid. Section 910.0 of the BOCA Building Code, for example, is expressed as the following factbase module:

```
(index ↑stand BOCA ↑item prov910p1 ↑tdb BOCAsec910 ↑provid sec910 ↑type derived)
(index ↑stand BOCA ↑item prov910p2 ↑tdb BOCAsec910 ↑provid sec910 ↑type derived)
(index ↑stand BOCA ↑item prov910p3 ↑tdb BOCAsec910 ↑provid sec910 ↑type derived)
(index ↑stand BOCA ↑item prov910p4 ↑tdb BOCAsec910 ↑provid sec910 ↑type derived)
(index ↑stand BOCA ↑item prov910p5 ↑tdb BOCAsec910 ↑provid sec910 ↑type derived)
```

In this example, only prov910p1, . . . , prov910p5 are directly associated with sec910, hence no other descendents are recorded in this group. Because some of the children in each group, such as prov910p2, are also parents of their descendents, such groupings can be extended to cover the entire information network of a standard.

The instances of the element class cond are stored in individual data files. The size of each file, or module, corresponds to the size of a code section. Each file is independent of all of the other components of SPIKE, including other modules. Each module is loaded as needed. The loading algorithm is further described in the next section. The instances of the element class index are stored in an **index file**. Every standard has a distinct index file. The index file that governs the standard of interest is loaded at the beginning of a consulting session.

4.4 Implementation of the Standards Processor

The generic nature of the SPIKE architecture is realized by the standards processor that manipulates the standards factbase. The standards processor is a collection of productions. The data-driven problem-solving strategy was adopted for this prototype. Nevertheless, because pure data-driven systems tend to result in obscure control flow, a **stage-driven** strategy has been used to complement it. The execution follows a sequence of stages, where each stage delineates a subset of all the rules in the production system that are candidates for firing during that stage.

Stages achieve partitioning of the knowledge base. Switching among different stages is accomplished by creating a **trigger** for the next stage while removing the trigger for the current stage. Triggers have been implemented as element classes which are named after the stage with which they are associated. A trigger is included as the first condition element in every production that is involved in the stage-driven process. Because SPIKE must respond to the most recent change in its memory for stage switching, the MEA conflict-

resolution strategy is used. Within each stage, the data-driven strategy takes over to select a production out of a set of productions with similar but slightly different condition elements.

Inefficiency in execution speed is a major drawback for a production system. The bottleneck is the matching step in the recognize-act cycle. The execution of a production system speeds up as the number of pattern matches decreases. Therefore, in implementing the standards processor, the following guidelines were used to keep the number of matches to a minimum [4]:

- Use an explicit control sequence where possible.
- Add element classes to clearly distinguish similar productions.
- Within the left hand side of each production, place more restrictive condition elements before less restrictive ones.
- Within each condition element, place more restrictive attributes before less restrictive ones.
- Avoid unnecessary changes to the working memory.
- Eliminate unnecessary variable bindings.

A detailed description of the implementation of the Standards Processor can be found Reference [29]. A complete listing of the SPIKE prototype and additional sample execution sessions are listed in the SPIKE User's Guide[28].

5 Summary

Although design standards must be satisfied during the design of engineering systems such as buildings and bridges, the process of design conformance checking using standards is often very tedious. The successful automation of conformance checking is one of the key components of a comprehensive computer-aided design system.

Prior to the 1980s, standards were most often directly coded into application programs. As discussed in Section 1.2.4, this hard-coding approach has several serious drawbacks. To support a fully automated CAD system, standards must be incorporated into the design process in a more generic and flexible manner.

The emergence of expert systems from artificial intelligence research has provided a technology that readily lends itself to the automation of design standards. Knowledge-based expert systems have become a powerful tool in tackling domains like design where some of the problem-solving knowledge is diverse, ill-structured, and heuristic in nature. Clearly, using an expert system tool, a standard can be represented and processed independent of a CAD application program. The Query Monitor and Roofload Checker prototypes presented

in Section 2 demonstrated this. They show that the knowledge contained in standards lends itself to processing in a knowledge-based expert system environment. However, they also demonstrate that casting standards as production rules is not the best way to build a generic, flexible, or even efficient standards processing system.

As opposed to directly coding standards as production rules, this paper proposed and explored the alternative approach of representing standards as facts which was derived from the standards modelling tools developed by previous researchers in this field. It was felt that developing an expert system knowledge representation scheme that was founded on well-established standards modelling principles and practices was a sound approach. Building on existing technology provided a catalyst for the development of the generic knowledge-based standards processing architecture that was introduced in Section 3. The implementation of this architecture in the form of the SPIKE prototype demonstrated that the proposed standards processing approach is generic, modular, and flexible and solves some difficult problems that could not otherwise be overcome. Therefore, it is believed that the result of this work has a promising role in the future development of comprehensive computer-aided engineering design systems.

6 Acknowledgements

Portions of this work were sponsored by the National Aeronautics and Space Administration under Grant NAG-1-604 entitled "Constraint Semantics in Engineering Database Use" and by the National Science Foundation under Grant CEE-8319869 entitled "Generative Database Systems for Structural Engineering Design." The support of these organizations is gratefully acknowledged. The cooperation of the Center for Building Technology at the National Bureau of Standards is also gratefully acknowledged.

7 References

1. American Institute of Steel Construction, Inc. *Manual of Steel Construction*, Eighth Edition, Chicago, IL (1978).

2. Building Officials and Code Administrators International, Inc. *The BOCA Basic Building Code*, Ninth Edition, Country Club Hills, IL (1984).

3. J.Bennett, L.Creary, R.Englemore, and R.Melosh. *SACON: A Knowledge-Based Consultant for Structural Analysis*, Stanford Heuristic Programming Project Report HPP-78-23, Stanford University, Stanford, CA, September (1978).

4. L.Brownston, R.Farrell, E.Kant and N.Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*, Addison-Wesley, Reading, MA (1985).

5. B.Buchanan and E.H.Shortliffe, (eds.). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, MA (1984).

6. Decision Table Task Group, Conference on Data Systems Languages. *A Modern Appraisal of Decision Tables*, Association for Computing Machinery, New York, NY (1982).

7. Digital Equipment Corporation. *VAX OPS5 User's Guide, VAX OPS5 Reference Manual*, Version 2.0, Maynard, MA (1985).

8. S.J.Fenves. "Software for Analysis of Standards," *Proceedings of the Second Conference on Computing in Civil Engineering*, Pages 82-91, American Society of Civil Engineers, Baltimore, MD, June (1980).

9. S.J.Fenves. "A Methodology for the Evaluation of Designs for Conformance With Standards," *Proceedings of the IABSE Colloquium on Informatics in Structural Engineering*, International Association of Bridge and Structural Engineers (IABSE), Bergamo, Italy, October (1982).

10. S.J.Fenves, M.L.Maher, and D.Sriram. "Knowledge-Based Expert Systems in Civil Engineering," *Proceedings of the Third Conference on Computing in Civil Engineering*, Pages 248-257, American Society of Civil Engineers, San Diego, CA, April (1984).

11. C.L.Forgy. *OPS5 User's Manual*, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, July (1981).

12. H.Furuta, K.S.Tu, and J.T.Yao. "Structural Engineering Applications of Expert Systems," *Computer-Aided Design*, Volume 17, Number 9, Pages 410-418, November (1985).

13. J.Garrett. *SPEX - A Knowledge-Based Standards Processor for Structural Component Design*, PhD thesis, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA (1986).

14. J.R.Harris and R.N.Wright. "Computer Aids for the Organization of Standards," *Proceedings of the Second Conference on Computing in Civil Engineering*, Pages 112-121, American Society of Civil Engineers, Baltimore, MD, June (1980).

15. F.Hayes-Roth, D.A.Waterman and D.B.Lenat, (eds.). *Building Expert Systems*, Addison-Wesley, Reading, MA (1983).

16. N.M.Holtz and S.J.Fenves. "Using Design Specifications for Design," *Proceedings of the Second Conference on Computing in Civil Engineering*, Pages 92-101, American Society of Civil Engineers, Baltimore, MD, June (1980).

17. M.L.Maher. "HI-RISE and beyond: Directions for Expert Systems in Design," *Computer-Aided Design*, Volume 17, Number 9, Pages 420-427, November (1985).

18. W.J.Rasdorf and S.J.Fenves. "Design Specifications Representation and Analysis," *Proceedings of the Second Conference on Computing in Civil Engineering*, Pages 102-111, American Society of Civil Engineers, Baltimore, MD, June (1980).

19. W.J.Rasdorf and S.C.Chern. *Load Consultant: A Knowledge Management Investigation*, Civil Engineering Technical Report, Number CE-002-85, North Carolina State University, Raleigh, NC, May (1985).

20. W.J.Rasdorf and L.M.Parks. "Expert Systems and Engineering Design Knowledge," *Proceedings of the Ninth Conference on Electronic Computation*, American Society of Civil Engineers, Birmingham, AL, February (1986).

21. W.J.Rasdorf and T.Wang. "Expert System Integrity Maintenance for the Retrieval of Data From Engineering Databases," *Proceedings of the Fourth Conference on Computing in Civil Engineering*, American Society of Civil Engineers, Boston, MA, October (1986).

22. D.R.Rehak and S.J.Fenves. "Expert Systems in Construction," *Proceedings of the 1984 International Computers in Engineering*, Pages 228-235, American Society of Mechanical Engineers, Las Vegas, Neveda, August (1984).

23. M.A.Rosenman and J.S.Gero. "Design Codes as Expert Systems," *Computer-Aided Design*, Volume 17, Number 9, Pages 399-409, November (1985).

24. D.Sriram, et.al. "Applications of Expert Systems in Structural Engineering," *Proceedings of the Conference on Artificial Intelligence*, Oakland University, Rochester, MI, April (1983).

25. F.I.Stahl. "The Standards Interface for Computer-Aided Design: An Overview of Some Technical Problems Associated With Automated Design Checking," *Proceedings of the Third Conference on Computing in Civil Engineering*, Pages 560-567, American Society of Civil Engineers, San Diego, CA, April (1984).

26. Teknowledge, Inc.. *M.1 Training Materials*, *M.1 Reference Manual*, version 1.3, *M.1 Sample Knowledge Systems*, version 1.3, Palo Alto, CA (1985).

27. T.Wang and W.J.Rasdorf. *Query Watcher User's Guide*, Computer Studies Program, North Carolina State University, Raleigh, NC, October (1985).

28. T.Wang and W.J.Rasdorf. *SPIKE User's Guide*, Version 1.0, Computer Studies Program, North Carolina State University, Raleigh, NC, July (1986).

29. T.Wang. *Generic Design Standards Processing in a Knowledge-Based Expert System Environment*, MS thesis, Computer Studies Program, North Carolina State University, Raleigh, NC, July (1986).

30. D.A.Waterman. *A Guide to Expert Systems*, Addison-Wesley, Reading, MA (1986).