OBSTACLES TO THE DEVELOPMENT OF COMPUTABLE MODELS OF DESIGN STANDARDS

Han Kiliccote[1] and James H. Garrett, Jr.[2]

ABSTRACT: Design standards play a significant role in civil engineering. Evaluating a design for conformance using such a large number of design standards is a tedious, laborious, and difficult task. One major research issue in CAE (Computer–Aided Engineering) is to develop software systems to assist in the usage of the design standards in the design process. Many obstacles to the development of such a tool exist; the major obstacles are: design standards are not self sufficient documents; design standards are indeterminate; processing design standards requires non–monotonic reasoning; and design standards contain higher–order provisions. An approach to dealing with these obstacles is proposed that treats standards processors as distributed black box abstractions able to communicate using a standardized protocol.

1 INTRODUCTION

Because of the complexity and dynamic nature of design standards and the need to incorporate standards into computer–aided design systems, a major goal of researchers in Computer–Aided Engineering (CAE) has been to develop software tools to assist in the usage of design standards in the design process (Fenves, Garrett, and Hakim 1994). Many investigators have already suggested various models and automated standards processing environments to operate on these models and evaluate designs for compliance to governing design standards (Fenves et al. 1987; Rasdorf and Lakmazaheri 1990; Jain, Law, and Krawinkler 1989; Yabuki and Law 1993; Garrett and Hakim 1992; Kiliccote et al. 1994). However, there has been limited success in transferring these environments into practice (Fenves, Garrett, and Hakim 1994). We believe the reasons for t is that the models were too simple compared to the actual complexity of the design standards. While expressing building models of the Eurocode No.2: Design of Concrete Structures (ENV 1992), the 1993 BOCA National Building Code (BOCA 1993) and Design Guide DG–1110–3–145 (Department of the Army Office of the Chief Engineers 1986), we realized that the following issues were not adequately addressed in previous models (Kiliccote 1995): design standards are not self sufficient documents; design standards are indeterminate; design standards contain exceptions; and design standards contain higher–order provisions. Each of these problems is examined in the next four sections.

2 SELF–SUFFICIENCY

Design standards are not self sufficient documents. To interpret and apply them correctly requires: knowledge about the world in which we live; knowledge about the domain of the design standard; and knowledge accumulated in other other design standards. In the next paragraphs, we discuss the problems associated with acquiring these three types of knowledge.

Knowledge about the world. Most of the provisions of a design standard refer to knowledge that all humans are expected to know. Unfortunately this type of knowledge is not formally expressed anywhere. Everybody just knows it. Guha and Lenat call this type of knowledge *prescientific*

1. Res. Assist., Dep. of Civil Eng., Carnegie Mellon University, Email: kiliccote@cmu.edu
2. Assoc. Prof., Dep. of Civil Eng., Carnegie Mellon University, Email: jgarrett@cmu.edu

(Guha and Lenat 1994). For example the knowledge that unsupported objects fall down is indirectly used but not expressed in the 1993 BOCA National Building Code (BOCA 1993). In AI, the study of this type of knowledge is called *commonsense ontologies* (Guha and Lenat 1990). As fundamental and commonsensical as the concepts may be, modeling them turns out to present some very difficult problems (Rich and Knight 1992).

Domain knowledge. Other than the *prescientific* knowledge that everybody is expected to know, understanding a design standard requires some knowledge about the domain of the design standard. For example, it is not possible to interpret most of the provisions in a design standard related to steel structures without some structural engineering knowledge. Some domain knowledge, along with some *scientific* knowledge (for example, the knowledge that an engineer acquires in college) is expected from the users of an engineering design standard.

In this case, a totally valid question would be "Is it possible to correctly interpret and apply a model of a building code without also having a model of most of the knowledge that an engineer learns in college?" Also, even if someone was able to transform a design standard into its computable form, would that model be different than, as Kowalski and Sergot (Kowalski and Sergot 1991) describe their work, "a layman's reading of provisions"?

Unfortunately the knowledge that is acquired in an engineering curriculum is not yet formally and completely represented. That means the person who prepares the computable model of a design standard becomes responsible for preparing the computable representation of this knowledge (or at least the portions required by the design standard that is being modelled). Obtaining the computable representation of this knowledge can be sometimes more difficult than obtaining the computable representation of the design standard. For example, we identified that to model Chapter 5 and one third of the Chapter 10 of the 1993 BOCA National Building Code, a building model containing 886 terms, which represent classes, attributes of these classes, and relationships between these classes, is required. Based on this subpart, we extrapolate that BOCA would require about 10,000 classes, attributes or class relationships in the building model to be able to represent the various parts of this code. Since most of these terms do not exist in most building models, methods to define these terms, which is clearly domain knowledge, need to be defined as part of the standards modeling process.

Knowledge accumulated in other other design standards. A design standard references a very large number of other design standards. For example, the 1993 BOCA National Building Code references 35 promulgating standard agencies and 393 design standards applicable to buildings (BOCA 1993). Referencing and accessing another design standard has not been investigated in previous work.

3 INDETERMINACY

By indeterminacy, legal scholars mean the ability to argue either side of a question using accepted methods of legal discourse (Berman and Hafner 1988). We claim that design standards, in that sense, are indeterminate.

We live in a world of "noisy" information (Lauritsen 1991). Design information can be approximate, incomplete, inaccurate, inconsistent, or uncertain. There may not be enough time to get all the relevant information about a design. Some information about the design (such as historical and environmental information) cannot be ascertained at all. Assumptions about the design must often be made. In such circumstances, it is very easy to obtain conflicting interpretations from a design standard. Sometimes, it is possible to obtain conflicting interpretations even if there is an agreement on the facts and the applicable provisions of the design standard. The reason for this occurrence of conflicting interpretations is that the design standards contain provisions that dif-

ferent people may interpret differently. Usually those interpretation problems are easily resolved by informal processes. But sometimes disputants seek the assistance of attorneys and go to court.

The most frequent cause of indeterminacy of provisions is caused by *open–textured concepts* used in expressing the provisions. An open–textured concept is one in which application to factual situations cannot be automatic, but which requires judgment and is context–dependent (Berman and Hafner 1988). Consider the following requirement from the 1993 BOCA National Building Code (BOCA 1993), henceforth referred to as BOCA93.

> **1017.4..4 Horizontal sliding doors:** [...] The door shall be openable from both sides without special knowledge and effort; [...]

Interpretation of this provision requires interpretation of terms such as "openable without special knowledge" and "openable without special effort." Even if these terms are defined in great detail, there will always be cases that lie outside the definition, which means it is not possible to describe in advance which knowledge or effort is special.

Such open–textured concepts are used very frequently in design standards, . Unfortunately, it is not possible to replace open–textured concepts with determinate concepts because, as we show in the rest of this section, the usage of open–textured concepts is inevitable and sometimes preferred.

Although ambiguity in design standards leads to several fundamental problems (see (Kiliccote 1994) for a discussion on this topic), ambiguous statements have a very important advantage over precise statements: ambiguous statements allow more freedom to the designer. Consider the following provision from the 1992 Pennsylvania Municipal Waste Regulations (Department of Environmental Resources Bureau of Waste Management 1992).

> **279.212.b Access control.** The operator shall construct and maintain a fence or other suitable barrier around the site sufficient to prevent unauthorized access.

The expression "suitable barrier to prevent unauthorized access" is an open–textured concept. The author of this provision could have used an expression that requires a barrier with a list of required properties or an enumeration of all possible barriers. But the form that uses the open–textured concept offers more freedom to the designer (operator in this case), because the designer is not restricted by a list of required properties or by an enumeration of all possible barriers.

4 PREVALENCE OF EXCEPTIONS

Design standards are complex documents (for example, BOCA93 contains over 5000 provisions). Designers cannot be expected to know all the provisions by heart. Designers become familiar with a limited number of provisions and use thus as their abstraction of the design standard; they look up to the other portions only when they are needed.

The organization of most design standards reflects this usage method. The provisions of a design standard are ordered from abstract to specific. The abstract provisions are the general provisions and the more specific provisions are the exceptions to these general provisions (and of course, exceptions may also have exceptions and so on). A general provision is a definition or requirement that is applicable in general context. For example, consider the following provision from BOCA93.

> **403.1 Sprinkler system:** All [high–rise] buildings and structures shall be equipped throughout with an automatic sprinkler system [...].

This provision is an abstraction, because immediately after the provision is described, its totology is invalidated by the following provision.

Exception [to 403.1]: An automatic sprinkler system shall not be required in [...]

Based on the above discussion, we claim that a design standard without exceptions is very unlikely. Thus, provisions will often have exceptions.

A possible way to eliminate exceptions would be to add some conditionals to the general provision that check if the more specific provision is applicable. For example, consider the following provisions from BOCA93:

1022.2.2 Height: Handrails shall not be less than 34 inches nor more than 38 inches, [...]

Exceptions [to 1022.2.2] 1. Handrails that form part of a guard shall have a height not less than 34 inches and not more than 42 inches.

Exceptions [to 1022.2.2] 2. Handrails within individual dwelling units shall not be less than 30 inches nor more than 38 inches [...]

To avoid such exceptions, provision 1022.2 could be represented as follows (English is used for simplicity):

1022.2.2 If handrails do not form part of a guard and are not used within individual dwelling units then they shall not be less than 34 inches nor more than 38 inches.

But as one can see, repeating conditional parts of the exceptions in the general provision can result in redundant and unreadable provisions. By using exceptions, such redundancy is unnecessary.

In current design standards, another problem is that provisions and their exceptions are not always cross–referenced. The main reason is that (except for simple oversights) the authors may decide not to cross–reference from a general provision to its exceptions because the cross–referencing would further complicate already complex definitions. For example, in BOCA93, the authors decided not to cross–reference the following provision from approximately 87 provisions that allow stairways to be used as elements of a means of egress.

1014.6.4 Spiral stairways: Spiral stairways shall not be used as an element of a *means of egress* except: in occupancies in Use Group R–3; within a single dwelling unit; from a mezzanine area not more than 250 square feet (23.25 m^2) in area which serves not more than five occupants; and in penal facilities from a guard tower, observation station or control room not more than 250 square feet (23 m^2) in area. [...]

5 HIGHER–ORDER PROVISIONS

First–order predicate logic is the only logic used in currently available standard representation languages. Unfortunately, first–order predicate logic is not sufficient to completely represent most of the current design standards. About 40% of the provisions in BOCA93 use second– or higher–order logic. Although some of them can be represented by ignoring the second–order part, some of them cannot be expressed at all using first–order logic. This means that the standard models and processors that currently exist cannot properly represent and reason with some of the provisions in the 1993 BOCA National Building Code.

Approximately 20% of the provisions of the BOCA93 use higher–order logic for which current tools can be used to represent them by "cheating" in building the representation. The following example illustrates one such provision.

421.9 Enclosures for public swimming pools: [...] The enclosure [of public swimming pools] shall meet the provisions of Sections 421.9.1 trough 421.9.3.

Because this provision specifies other provisions that shall be met, it is not first–order logic. This means that this provision uses second– or higher–order logic mechanisms that cannot be directly represented in current standards processing models. But, all of the first–order logic models can be used to represent this provision. A possible way to represent this provision is to ignore it. Because provisions in sections 421.9.1 trough 421.9.3 will always be evaluated, the purpose of the provision would be accomplished.

Again, because the following provision specifies other provisions that shall be met, it is not first–order logic.

> **503.1 General:** The height and areas of all buildings and structures [...] shall not exceed the limitations fixed in Table 503, except as specifically modified by this chapter and the following sections: 402.7, 403.3.3.1, 414.2, 416.3, 418.3.1.1, 3103.3.5

Again, by "cheating," this provision can be represented as follows (for simplicity English is used to represent the provision):

> **BOCA503.1:** If the building is **NOT** a covered mall that does not exceed three floor levels in height at any point *(402.7)*, is **NOT** a high rise building having occupied floors located more than 75 feet above the lowest level of fire department vehicle access *(403.3.3.1)*, is **NOT** an airport traffic control towers that does not exceed 1500 square feet per floor occupied only for air traffic control, electrical and mechanical equipment rooms, radar and electronic rooms, office spaces incidental to tower operation and lounges for employees, including restrooms *(414.2)* and does **NOT** use hazardous production materials (HPM) *(416.3)* and is **NOT** a grain elevator or similar structure of Type 1 or Type 2 construction *(418.3.1.1)* and is **NOT** an air–supported, air–inflated, membrane–covered structure erected for a period of 90 days or longer *(3103.3.5)*, then the height shall not exceed the limitations fixed in Table 503.

Like the problem of dealing with exceptions, trying to represent higher–order provisions in first–order logic can result in unnatural, redundant, and unreadable provisions. Also, approximately 20% of the provisions of the BOCA93 use higher–order logic and cannot be represented in first–order logic without some great effort.

The higher–order provisions that we encountered in design standards can be classified as control provisions and deeming provisions. In the next two sections, we discuss these two more commonly encountered types of higher–order provisions.

5.1 Control provisions

Control provisions control the usage of other provisions. As described in section 4, design standards are organized in a multi–layered organization where provisions are ordered from abstract to specific. Thus, some of the provisions are more abstract than the others. For example, the following provision from BOCA93 is a control provision that controls the application of referenced standards.

> **102.4 Referenced standards:** The standards referenced in this code and listed in Chapter 35 shall be considered part of the requirements of this code to the prescribed extend of each such reference. Where differences occur between provisions of this code and referenced standards, the provisions of this code shall apply.

As one can see, the second part of this provision ("Where differences occur between provisions of this code and referenced standards, the provisions of this code shall apply") is applied to resolve a conflicting situation.

Other than the obvious difficulties associated with control provisions, the "chicken and egg" problem presented below requires special attention.

Example 4.

> **906.2 Equipped throughout:** Where the provisions of this code require that a building or portion thereof be equipped throughout with an automatic sprinkler system, the system shall be designed and installed in accordance with Section 906.2.1, 906.2.2 or 906.2.3.

Because this provision references other provisions, it is obviously not first–order logic. The reference to other provisions occurs in two parts of the provision. In the last part (the portion that includes the sentence "in accordance with Section 906.2.1, 906.2.2 or 906.2.3"), the reference to other provisions can be simply ignored. But the first part of the provision ("Where the provisions of this code require that a building or portion thereof be equipped throughout with an automatic sprinkler system") cannot be represented in first–order logic without substantial changes to the provisions. It requires that all of the provisions of the design standard be browsed and the provisions that require that a building be equipped throughout with an automatic sprinkler system be identified and evaluated. If a provision that requires that a building be equipped throughout with an automatic sprinkler system is found, then the requirement 906.2 is applicable.

The current standards processing systems cannot be easily augmented to perform a search that browses all other provisions. The only way to represent this provision using first–order logic is to change all the provisions that require that a building be equipped throughout with an automatic sprinkler system.

5.2 Deeming provisions

Deeming provisions are the the provisions that allow things which are not true to be treated as if they were (Routen 1989). Consider the following provision from the Eurocode No.2: Design of Concrete Structures (ENV 1992):

> **2.5.2.1.P(6)** A wall should have a horizontal length of at least four times its thickness. Otherwise it should be treated as a column.

The last part of this provision ("otherwise it should be treated as a column") requires that a wall, which is already classified as a wall and not as a column, should be treated as a column. The author of this provision agrees that a wall is not a column but it may be assumed as such if its horizontal length is not more than four times its thickness.

In the rest of this section, we illustrate the difficulties of expressing deeming provisions.

A possible way to express a deeming provision is to substitute another term for *column* in the provisions that uses the term "column" (Routen 1989), which means we need to define another term (such as "deemed_column") as follows:

```
deemed_column: if column
deemed_column: if wall and if horizontal-length-is-not-at-least-4-times-
thickness
```

This is not entirely ad hoc because one can argue when the legislation uses the word "column", it implicitly (because of the effect of the deeming provision) appeals to a concept other than the real–world concept "column." This technical legal concept does not have a name already and "deemed_column" is as good a name as any (Routen 1989). Unfortunately this solution has three problems: logical validation; legal validation; and practicality.

Logical validation. As one can see, a wall should not be treated as a wall if it does not have a horizontal length of at least four times its thickness. Not treating a wall as a wall is not possible in first–order logic because this requires a statement similar to the following which is inconsistent.

```
if x and y then not x
```

The above statement is inconsistent because after it is applied, one of the conditions required for its application is lost, which means it should not have been applied at the first place.

Legal validation. Routen points out this is not good from the point of view of legal validation (Routen 1989):

> "However, the proliferation of such peculiar concepts is not good from the point of view of legal validation. One could as easily argue that the whole reason for the existence of deeming provisions is to allow the drafter to avoid having to appeal to such concepts when defining something with great precision but instead be able to couch definitions in terms of ordinary concepts with patches."

Practicality. This solution requires that all the occurrences of the term "column" be replaced by "deemed_column" and all future usages of the term "column" be caught and signaled. Also, please notice that such a change could have profound effects on the explanation capabilities of the computable model.

To make the things worse a deeming provision can also be joined with a control provision. For example, consider the following provision from the Eurocode No.2: Design of Concrete Structures (ENV 1992).

> **2.5.2.1.P(5)** Ribbed or waffle slabs may be treated as solid slabs for the purposes of analysis, [...]

This provision contains a deeming provision ("may be treated") and a control provision ("for the purposes of analysis"). From a practical point of view, to express this knowledge we have to: indicate for each provision its purpose; and have a mechanism that would prevent provisions that do not have "analysis purposes" to treat ribbed or waffle slabs as solid slabs. Such a mechanism is a control provision as it prevents the application of other provisions.

In summary, design standards contain a large number of second– or higher–order logic in their provisions. For example Chapter 10 of BOCA contains 429 provisions. About 167 of them are not expressed in first–order logic. Thus design standards are not first–order logical constructs. Representing them in first–order logical systems is very hard and not natural. Also because the maintainability is decreased substantially with each instance of representational "cheating," such representations lead to error prone and inefficient computable models.

6 A DISTRIBUTED APPROACH FOR REPRESENTING STANDARDS

In the previous sections, many challenging obstacles were presented that need to be addressed when developing a standards representation and processing approach. One approach that we propose to solve these obstacles is to use a distributed, black box abstraction–based framework to represent and reason with design standards. In the framework we are developing, we allow multiple representation and reasoning methods to exist in the same agent. Provisions can be modeled using any of these methods. From outside of the agent, the provisions are perceived as a black box abstraction. A common communication language will be used to request services (such as computation and conflict resolution) from these agents. This means that the best way to represent each kind of provision can be made an internal decision hidden within the abstraction boundary of the agent. A major advantage of such an approach is that it will allow for the incorporation of a wide variety of representational techniques into the same standards model, thus providing a broader, more powerful set of representations to use in modeling a design standard.

Such a framework will accommodate various types of knowledge bases and provide access to multiple design standards, thus, providing a method to overcome the lack of self–sufficiency in

standards. Guides and heuristics supplied by the designers, companies or standard organizations will be agents in this distributed framework. Indeterminacy may be overcome by using the concept of "locality of ambiguity." In legal reasoning, even if a provision is indeterminate, its consequences must be determinate. Thus, even if an open–textured concept is used in a provision, such as "special machinery or equipment" used in BOCA93, after the decision about the applicability of the open–textured concept is decided, the indeterminacy of the provision goes away and is not carried to other provisions that use the result of this provision. Using this property of legal reasoning, the agents in the framework representing guides and heuristics supplied by the designer may be used to determine the truth value of open–textured concepts. The framework we are developing will also allow for agents to control other agents, such as controlling the execution of other agents.

7 CLOSURE

In this paper, we described the results of a preliminary analysis of the obstacles that may be encountered while representing design standards in a computable model. The majority of these obstacles are caused by lack of the self–sufficiency, indeterminacy, the presence of exceptions and higher–order logic components of design standards. A distributed, black box abstraction–based approach to delivering standards usage support appears to address some of the problems found in existing approaches. A prototype system employing this approach is currently being implemented and will then be tested extensively in the next year.

8 ACKNOWLEDGEMENTS

This material is based on work supported by the National Institute of Standards and Technology (Computer Integrated Construction Program of Building Fire Research Laboratory). Project No. 08775.

9 REFERENCES

Berman, D., and C. Hafner. 1988. Obstacles to the development of Logic–Based Models of Legal Reasoning. In *Computing Power and Legal Language,* ed C. Walter. Westport, CT: Greenwood Press

BOCA (Building Officials and Code Administrators). 1993. BOCA National Building Code/1993. Twelfth Edition. Country Club Hills, IL

Department of the Army Office of the Chief Engineers. 1986. Fire Station Design Guide No: 1110–3–145, Washington, D.C.

Department of Environmental Resources Bureau of Waste Management. 1992. *Commonwealth of Pennsylvania, Pennsylvania Code. Title 25. Environmental Resources.* Harrisburg, PA

ENV (European Economic Union). 1992. *Env–1992–1–1 – Part 1. Eurocode 2: Design Of Concrete Structures. General Rules & Rules For Buildings*

Fenves, S. J., R. N. Wright, F. I. Stahl, and K. A. Reed. 1987. *Introduction to SASE: Standard Analysis, Synthesis and Expression.* Technical Report NBSIR 872513. National Bureau of Standards. Washington, DC

Fenves S. J., J. H. Garrett Jr., and M.M. Hakim. 1994. Representation and processing of design standards: a bifurcation between research and practice. To be published in *ASCE Structural Congress.* April 1994

Garrett, J. H., Jr., and M.M. Hakim. 1992. Object–oriented model of engineering design standards. *Journal of Computing in Civil Engineering.* 6(3): 323–47

Guha, R.V., and D. B. Lenat. 1990. Cyc: A Mid–Term Report. *AI magazine*, 11(3): 31–59

Guha R.V., and D. B. Lenat. 1994. Enabling Agents to Work Together, *Communications of The ACM*. 37(7): 126–42

Jain, D., K. H. Law, and H. Krawinkler. 1989. On processing standards with predicate calculus. In *Proceedings of the Sixth Conference on Computing in Civil Engineering*. ASCE. Atlanta, GA. 259–266

Kiliccote H. 1994. The context–oriented model: a hybrid approach to modeling and processing design standards. Master's thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

Kiliccote, H., J. H. Garrett, Jr., T. Chmielenski, and K. A. Reed. 1994. The Context–Oriented Model: An Improved Modeling Approach for Representing and Processing Design Standards. In Proceeedings of the First Congress in Computing in Civil Engineering, Jun 1994, Washington, DC, ed. K. Khozeimeh. 145–52. Washington, DC: American Society of Civil Engineers

Kiliccote, H. 1995. The Standards Processing Framework. Ph.D. proposal, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA.

Kowalski R., and M. Sergot. 1991. The use of logical models in legal problem solving. In *Law, computer science, and artificial intelligence*, ed. A. Narayanan and M. Bennun. 99–118. Norwood, NJ: Ablex Publishing Corporation

Lauritsen M. 1991. Knowledge–based approaches to government benefits analysis. In *Proceedings of the Third International Conference on Artificial Intelligence and Law*. 25–28 June 1991, Oxford, UK. 98–10

Rasdorf, W. J., and S. Lakmazaheri. 1990. Logic–based approach for modeling organization of design standards. *Journal of Computing in Civil Engineering*. 4(2): 102–123

Rich, E., and K. Knight. 1992. *Artificial Intelligence*. 2d ed. McGraw Hill. New York, NY

Routen, T. 1989. Hierarchically organised formalisations. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*. 13–16 June 1989. Vancouver, BC, Canada. 242–50

Yabuki N. and K. H. Law. 1993. An object–logic model for the representation and processing of design standards. *Engineering with Computers*. 9: 133–159.