Zeitschrift: IABSE reports = Rapports AIPC = IVBH Berichte
Band: 58 (1989)

Artikel: Object-oriented representation of design standards
Autor: Garrett, James H. Jr.
DOI: https://doi.org/10.5169/seals-44924

Object-Oriented Representation of Design Standards

Représentation «orienté objet» des standards de conception

Objektorientierte Darstellung einer Entwurfsnorm

James H. GARRETT Jr.
Assistant Professor
University of Illinois
Urbana, IL USA

James Garrett, born in 1961, received his BSCE, MSCE, and Ph.D. from Carnegie Mellon University in Pittsburgh, PA. He joined the faculty at the University of Illinois in 1987 and has been performing research in the areas of standards processing, object-oriented building modeling and neural networks.

SUMMARY

This paper describes an object-oriented standards representation in which the logic and data items of a standard are all represented as objects and the methods for manipulating and using the standard are stored within these objects. By using the object-oriented framework described in this paper, it is possible to build a modular, flexible, and powerful representation of a design standard. The benefits of having this natural and declarative description of a design standard are: it makes the logic of the design standard much more apparent than a pure textual representation, it facilitates the automated checking of design objects against a design standard, and greatly enhances the ability to reason about and apply the requirements of the design standard during computer-aided design.

RESUME

Cet article décrit une représentation «orienté objet» des standards de conception dans laquelle les éléments logiques et les variables d'un standard sont tous les deux représentés sous la forme d'objets contenant par ailleurs la manière de manipuler et d'utiliser ces standards. L'utilisation d'un cadre «orienté objet» tel que celui décrit dans cet article, permet de construire une représentation modulaire, flexible et puissante des standards de conception. Les avantages de l'utilisation de cette description naturelle des standards sont: la logique des standards de conception comparativement à une représentation uniquement textuelle: contrôle automatique facilité des objets vis a vis des standards de conception, importante amélioration des possibilités de raisonner avec des standards de conception et de remplir les exigences définies par ces standards pendant la conception assisté par ordinateur.

ZUSAMMENFASSUNG

Dieser Beitrag beschreibt die objektorientierte Darstellung einer Norm. Die Norm ist in der Form von Objekten gespeichert, welche die Methoden für deren Anwendung enthalten. Dadurch wird es möglich, Entwurfsnormen modular, flexibel und sehr anwendungsfreundlich darzustellen. Es ergeben sich folgende Vorteile: Die Logik der Norm wird wesentlich besser ersichtlich als bei einer reinen Textdarstellung. Durch die dadurch mögliche automatisierte Ueberprüfung von Entwurfsobjekten wird die Anwendung im Computer-Aided-Design stark verbessert.

1. INTRODUCTION

As civil engineers, we are required to design buildings, waste treatment facilities, public transportation systems, etc. that conform to a myriad of design standards, specifications, and codes. Although they may have more to deal with, civil engineers are not alone in having to deal with regulation of the performance of their designs; most professional engineers must verify that their designs meet some collection of performance regulations. The ability to properly use these codes and standards (i.e., to correctly identify applicable code provisions, interpret them, and apply them) takes experienced designers years to develop. Because of the many applicable codes that must be considered and obeyed by an engineer and the dynamic nature of these standards, computer-aided usage of design standards is an extremely important component of computer-aided engineering (CAE). Several researchers are working towards a standards representation and processing environment that will free the engineer from concern with the details of any particular standard by assisting him in designing for, and verifying, conformance with all applicable standard provisions [6, 3, 2, 4, 8]. Such an environment would support creative design but limit solutions to be within the bounds of acceptable practice spelled out in the codes. One might argue that such an environment would in fact overly limit creative design and hence be less desirable as a design environment. However, where applicable codes exist, we have an obligation to ensure that our meet the minimum levels of performance spelled out in those codes. It is for this type of design activity, i.e., that falling under the jurisdiction of an existing code, that this standards representation and processing environment is envisioned to support.

For over 20 years, researchers have been investigating ways to represent and automatically use the information contained in a design standard. Fenves, who was the first to propose the idea of formally representing design standards, represented the logic of the standards as a collection of decision tables, where each decision table was responsible for the evaluation of a data item within the standard [5]. Data items were simply defined as the variables, including the provisions themselves, to which the standard refers within its text. When addressing the issue of automated usage of standards, most researchers have treated this formal standard representation as a passive entity to be acted upon by a single, monolithic standards processor, in much the same way that most pure rule-based systems rely on a single inference engine. A more flexible method of representing and processing a standard, which is the subject of this paper, would be to provide each data item the capability of maintaining its own dependencies, of determining its own value, of retrieving the necessary data from a design description, etc., using an object-oriented approach. In other words, treat each data item identified in the standard as a self-contained object that contains all information particular to that data item and all methods for manipulating that information.

The purpose of this paper is to describe such an object-oriented standards representation and processing environment. The remainder of this paper describes: 1) the basic function and form of a design standard, 2) the general concept of object-oriented programming, and 3) the proposed object-oriented model of a design standard.

2. DESIGN STANDARDS

The basic function of a design standard is to state requirements that must be met in order to ensure that an adequate level of performance for an entity is provided. These requirements are derived from experience with successful and unsuccessful designs. As more and more knowledge is gleaned from design experience and experimental research, the definition of adequate performance, and thus the design standard, are further refined.

Most standards state minimum levels of acceptable performance and identify a collection of criteria that quantitatively define what acceptable performance means. Each criterion is a logical expression of some set of variables, or data items. For example, in Fig. 1. a portion of the AISC LRFD specification is given from which data items and and the criteria can be determined. The requirement is that there be adequate compressive strength for compression members; the criterion for determining that adequate compressive strength is provided is $P_u \leq \phi_c \ P_n$, where P_u is the factored compressive load on the member. This section is predominantly concerned with defining the data item F_{cr} .

The data items within a standard can be classified as follows:

1. *basic* – no explicit expression is provided in the standard that defines its value, hence, it's value is to be retrieved from the design being evaluated or from general knowledge of the domain (e.g., E, r, F_y , K, l);

2. *derived* – an explicit logical or mathematical definition for deriving its value is provided within the text of the standard (e.g., F_{cr} , P_n, λ_c); and

3. *requirement* – a special type of derived data item that identifies the criteria that must be satisfied and has one of the following status values: "satisfied", "violated", or "not applicable" (e.g., design-compressive-strength).

As can be seen from the above described example, precedence relationships exist between the data items within a design standard and the methods to use in determining the value of a data item are many times dependent on the value of other data items.

" The design strength of compression members whose elements have width-thickness ratios less than λ_r of section B5.1 is $\phi_c P_n$

$$\phi_c = 0.85$$
$$P_n = A_g F_{cr} \qquad \text{(E2-1)}$$

For $\lambda_c \leq 1.5$

$$F_{cr} = (0.658^{\lambda_c^2}) F_y \qquad \text{(E2-2)}$$

For $\lambda_c > 1.5$

$$F_{cr} = \left[\frac{0.877}{\lambda_c^2}\right] F_y \qquad \text{(E2-3)}$$

where

$$\lambda_c = \frac{Kl}{r\pi}\sqrt{\frac{F_y}{E}} \qquad \text{(E2-4)}$$

A_g = gross area of member, $in.^2$
K = effective length factor
l = unbraced length of member, in.
r = governing radius of gyration about plane of buckling, in.

For members whose elements do not meet the requirements of Sect. B5.1, see Appendix B5.3. "

Figure 1. — Excerpt for AISC LRFD [1] - Chapter E, Section E2

3. OBJECT-ORIENTED METHODOLOGY

The basic building block of an object-oriented representation is the *object* — a modular, self-contained collection of descriptive attributes and the procedural methods for manipulating those attributes. Representation in an object-oriented environment first requires the description (declaration of attributes and methods) of the general types of objects that populate the domain (class objects), and then requires the generation of instances of the class objects to describe the particular entity being modelled.

In object-oriented representations, everything is an object. Objects can represent concepts, physical objects, processes, etc. In all cases, the object possesses a set of attributes and methods. Attributes represent data about the object; methods represent processes that the object is capable of performing. Attributes and methods are usually both represented as *slots* within the object. Other objects can access these slots, but only by sending a message to the object that "owns" the data or method. In addition to having a value, the attributes of an object may also have self-descriptive information, such as permissible range or type. This

information is stored in *facets* that are associated with the slots. A special type of facet, called a procedural attachment or demon, watches a slot value and executes a method when that value is added, changed, or erased. This feature of object-oriented environments is especially suited for performing event-driven computation and will be extensively used in the object-oriented modelling and usage of standards.

Fig. 2. shows the typical structure and an example of a data-item object. In that example, the first four slots store declarative information about the data-item, such as its value or its ingredients. The "value", "ingredients", and "dependents" slot all have facets that describe what to do if a slot value is needed or erased.

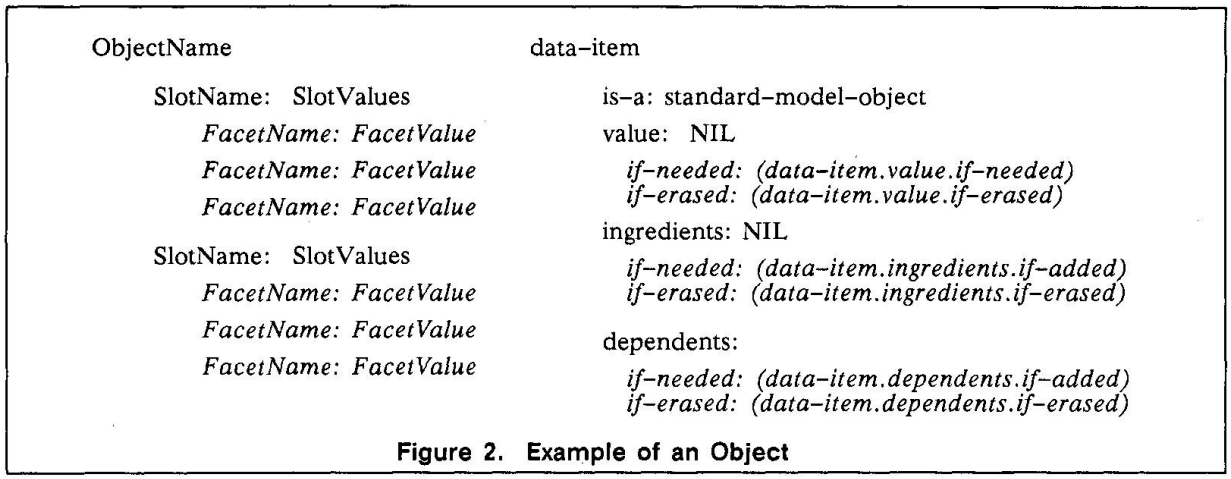


Figure 2. Example of an Object

A common practice in object-oriented programming is to develop templates for types of objects, commonly called *class* objects. These class objects usually possess attributes, attribute values, method names, or methods that are common to several more specific objects. If these more specific objects are themselves templates for other even more specific objects, they are called *subclass* objects. Objects that represent an specific instance of a class or subclass object are called *instances*. Instances are *children* of subclasses (or classes), subclasses are children of classes (or other subclasses), classes are *parents* of their subclasses and instances and subclasses are parents of their instances or other subclasses. These parent-child relations are important because in most object-oriented programming environments, children automatically *inherit* attributes and methods from their parents. For example, all instances of the object data-item (shown in Fig. 2.) will inherit the slot names "value", "ingredients" and "dependents", and their procedural attachments from the object data-item. Through inheritance, it is possible to represent information at an appropriate level of object generality and have all more specific instances of objects inherit that information, thus reducing redundancy and improving consistency.

Hence, the key ideas of object-oriented programming are that objects possess attributes and methods, can inherit attributes and methods from other objects, and communicate with each other (i.e., get data or execute an object's method) only by sending messages.

4. OBJECT-ORIENTED DESIGN STANDARD MODEL

As was stated previously, the purpose of this paper is to describe an object-oriented model of a design standard that facilitates automated standard conformance verification. In order to ba able to fully automate the verificalion of a design for conformance with applicable design standards the following are necessary:

1. an object-oriented model of the data items (both basic and derived) to which the standard refers and the logic for determining the value of each derived data item expressed in the standard;

2. an object-oriented description of the entities to which the standard applies, which represents the attributes of each entity within the scope of the standard and serves as a repository of knowledge about the entities not found in the design standards; and

3. a collection of mappings: 1) between the basic data items in the standard and the attributes of the design description, and 2) between design description objects and behavior limitations (see Section 4.3.) in the standard model.

All three parts of the model are required in order to ensure proper automated interpretation, not just the first. Most standards processors have provided the first part and a little of the second in the form of a hierarchy of classifiers [6]. Elam's and Lopez's work identified the need for, and implemented in limited form, all three parts [4]. This work presents an architecture, cast in an object-oriented framework, that includes a representation of standard logic (explicit knowledge contained in a standard), the underlying description of the entities within the scope of the standard (implicit knowledge in some standards), and the mappings between data items in the standard and the attributes of the design description.

4.1. Object-Oriented Design Standard Logic Model

4.1.1. Objects in the Standard Logic Model

As illustrated in Section 2., a design standard is a collection of logically interrelated data items. These data items and their logical interrelationships are represented using the following object classes, the hierarchy of which is shown in Fig. 3.

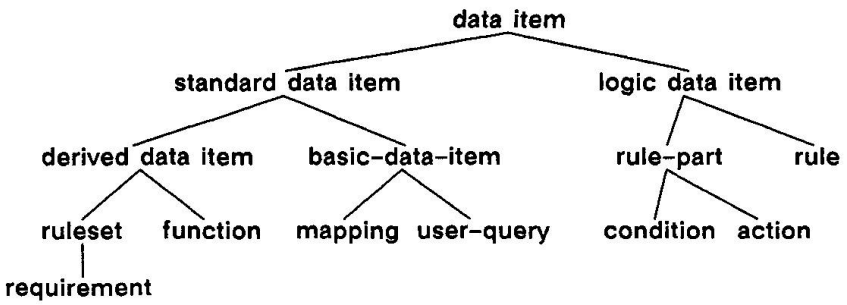


Figure 3. — Hierarchy of Object Classes for Standard Logic

Data item. This class of objects is the most general class in the hierarchy describing the general properties of data items used to represent a standard. A data item is defined to be an explicitly represented variable that has the following properties: a value (either computed or input by the user), a list of ingredient data items upon which their computed value is based, a method (ruleset, function, mapping, or prompt-string) for determining its value from the list of ingredients, and a list of data items that depend on its value. The hierarchy in Fig 3. shows two subclasses of data-item: standard data item and logic data item.

Standard Data Item. This class of objects represents what can be thought of as the "traditional data items" of standard representations — the data items referred to explicitly within the text of the standard. Standard data items are specialized according to the method in which their value is determined: derived or basic.

Derived Data Item. This class of objects represents the data items referred to explicitly, and given a method of evaluation, within the text of the standard. Derived data items are specialized according to the method in which their value is computed. Other subclasses of derived data items may be added later because of the flexibility of this object-oriented approach, but for now only functions and rulesets are defined.

Ruleset. This class of object represents a specific type of derived data item whose value is conditional and whose evaluation strategy is thus represented as a collection of rules. A ruleset is almost identical to a decision table in that all of its rules focus on the evaluation of a single data item. The ruleset offers a little more flexibility in that the inference strategy may be varied; this flexibility gained in using rulesets over decision tables was promoted by Elam and Lopez [4]. Like all data items, rulesets have ingredients. But, unlike past representations, the ingredients to a ruleset are the rules that make up the ruleset (i.e., logic data items), not standard data items.

Requirement. This class of objects is a special subclass of ruleset that describes a requirement with a set of criteria (or conditions) from which its value ("satisfied", "violated", or "not applicable") is computed. Requirements are also instances of behavior limitation objects, which are described in Section 4.3.

Function. This class of objects represents a second subclass of derived data item, whose evaluation method is a non-conditional function, similar to an action of a ruleset. Like all data items, functions have ingredients which are the data items appearing in the expression of the function.

Basic Data Item. This class of objects represents the data items referred to explicitly within the text of the standard, but not given a method of evaluation within that text. For these data items, it is assumed that the user will provide the needed value either from a design model or directly in the form of an answer to a query. Hence, basic data items are specialized according to the method in which their value is retrieved: mapping or user-query.

Mapping. This class of objects represents a subclass of basic data items. A mapping is a declarative description of where to look in, or how to compute a value from, the object-oriented design description. Mappings are described in more detail in Section 4.3.

User Query. This class of objects represents a second subclass of basic data items. A user query data item simply describes a data item for which it is known a priori that its value will have to be asked for from the user. Because of this a priori knowledge, the user query object contains a prompt-string to use in querying the user, type and range information for checking user input, and default values in case the user does not know the answer but wishes to continue.

Logic Data Item. — This class of objects, a subclass of the data item object, represents such items as conditions, actions or rules, that are used to describe the logical relationships between standard data items. Each logic data item possesses an evaluation method. For conditions and actions (rule-parts) , this evaluation method is in the form of an algebraic expression. For rules, this evaluation method is in the form of a list of condition-value pairs and an action to perform given those conditions match those expressed for the rule. Layers of standard data items are related through layers of logic data items. By having these logic data items explicitly represented, it is possible to maintain much more refined dependency relationships. Past representations would invalidate a data item if any of its ingredient data items changed, which may not have been necessary if the rule used to originally compute the dependent data item did not depend on that ingredient.

Rule Part. This class of objects represents conditions and actions, which have a symbolic expression for their description of evaluation strategy.

Condition. This class of objects has as its evaluation method an algebraic expression that evaluates to either T or F. The ingredients to a condition are defined to be the data items contained within the symbolic expression of the condition; the dependents of a condition are the rules that refer to that condition.

Action. This class of objects has as its evaluation method an algebraic expression with no restriction on its value. The ingredients to an action are defined to be the data items contained within the symbolic expression of the action; the dependents of an action are the rules that refer to that action.

Rule. This class of objects possesses an attribute for storing a pattern of condition-value pairs and an action to execute in the event that the condition-value pattern matches the actual condition-value situation. The ingredients to a rule are the conditions and action to which the rule refers; the dependent of a rule is the ruleset (a derived data item) to which the rule belongs.

4.1.2. Example Standard Logic Model

To illustrate the use of the above described objects in modeling and evaluating the logic of a design standard, Sect. E.2 from Chap. E of the American Institute of Steel Construction Load and Resistance Factor Design Specification [1] (see Fig. 1.) is modeled using the above described objects (see Fig. 5.).

To determine the value of DESIGN_COMPRESSIVE_STRENGTH, it must be sent a message to return its current value. The DESIGN_COMPRESSIVE_STRENGTH object, being an instance of a ruleset, responds to the message by sending messages to the rules in its RULES slot: DCS-1, DCS-2 and DCS-3. The ruleset first sends a message to rule DCS-1; if DCS-1 does not respond with a non-NIL value, the next rule is messaged. DCS-1, being a rule, responds to a message for its value by sending messages to each of its

identified conditions and then checks if the condition returns the value indicated for the rule. For example, DCS-1, in response to a message for its value, sends a message to the object LOCAL_BUCKLING_SATISFIED to return its value and if this object responds with a value "T", DCS-1 then sends a message to the object DCS_E2_SATISFIED to return its value.

When a condition is defined, i.e., its symbolic expression is filled in, this expression is parsed to determine the ingredients and is transformed into a LISP-evaluatable expression. Hence, when a condition is sent a message to return its value, it sends messages the identified ingredients and then evaluates the LISP-evaluatable expression with the returned ingredient values. Thus, when DCS_E2_SATISFIED is sent a message, it responds by sending messages to the ingredients Pu, PHI-C, and Pn.

Pn, being a function, responds by sending messages to its ingredients and evaluating its LISP-evaluatable expression, both of which were generated when the symbolic description was defined. Hence, when a function is messaged, it sends messages to its ingredients for their values and then evaluates the LISP-evaluatable expression with the returned ingredient values. When Pn is sent a message, it sends messages to Ag and Fcr to return their values. Fcr, being a ruleset, reacts to a message to return its value exactly as the DESIGN_COMPRESSIVE_STRENGTH ruleset did. This recursive process of messaging rules, conditions, actions and functions continues until the objects receiving messages are instances of the basic-data-item class, such as Ag, Kx, Lx, etc.

There are two types of basic data items, user-queries and mappings. When a user-query is messaged, it simply prints out its prompt string and checks the users response against type and range information stored in the query object. When a mapping is sent a message, it retrieves the information from the object-oriented design description (described in Section 4.3.). After the values of these basic data items are retrieved and backpropagated to the derived data items, their values can be computed. After the values for these derived data items have been backpropagated to logic data items, their values can be computed. And finally, after the values of these logic data items have been backpropagated to the requirements, their values can be computed.

4.1.3. The Benefits of an Object-Oriented Representation of Standard Logic

Because the decision table for so long has been the main way of representing the logic within a design standard, one must ask why this object-oriented model is any better. The benefits of using this object-oriented approach all basically relate to flexibility and are described as follows.

1. Every object maintains its own strategy for determining its value. In other words, there is no central definition of what a condition is, of what an action is, of what a rule is, etc. Although the predominant kind of function is the algebraic expression in terms of other data items, this does not have to be the only kind of function. For example, a subclass of function could be defined to be a neural network that has been trained to recognize a collection of data that defies mathematical description, which when given a set of ingredient values as input, returns a value for the function. Similarly, one could have more than one type of ruleset (fire-one rule, fire-all-rules, etc.), condition (symbolic, numeric, neural), and rule (AND rules, OR rules, etc). Such flexibility can only be achieved when the representation is object-oriented, where the object requesting a value need not know with what kind of object it is dealing.

2. The values and ingredients for individual conditions and actions can be maintained individually, whereas for a decision table all ingredients for all conditions and actions are lumped into the set of ingredients for the data item evaluated by the decision table. This lumping of ingredients causes data items to be unnecessarily nullified when an ingredient's value is changed. If the ingredient was not actually used in the evaluation of a data item, its change should not invalidate the dependent data item.

3. By also representing basic data items as objects, a place is provided to store data item specific prompt strings, range and type information, and mappings. The storing of this information with the basic data items makes for a much more organized and flexible description of a design standard.

4.2. Object-oriented Design Description

The design description is intended to perform many functions. First, it serves to identify the entities to which the standard requirements are intended to apply. Second, this model will identify, for each design object, a set of attributes from which the basic data items found within the standard can be computed. This will permit the generalized expression of the mappings between the data items of a standard and the attributes of these design entities. The design descriptions can then be mapped to a much larger, global model using the knowledge-based database interface mechanisms of KADBASE [7], or a similar data communication environment. In fact, because such a global model does not yet exist, it is most likely that the design descriptions developed for various standards will play a part in determining the information content of such a global model.

4.3. Mappings Between Standard Model And Design Description

There are two types of mappings between the standard data items and the design description objects: 1) the mappings between the design description objects and the various applicable requirements of the standard, and 2) the mappings between the basic data items in the standard model and the design description object attributes. The first set of mappings is essentially used to determine which of the standard requirements are applicable for the design entity in question and need to be checked. This mapping is simply represented as a slot in the design description objects which contains a list of applicable behavior limitation objects [6] (in the structural case - behavior limitation objects represent combinations of stress state and limit state). Each of the requirement data items in the standard model is an instance of the behavior limitation object to which it applies. Thus, the layer of behavior limitation objects is the medium of communication between a design description and a standard; these behavior limitation obejcts are the components of the classification system used in previous standards processing environments [6].

The second set of mappings go in the opposite direction to the previous set by linking data items to the appropriate slots of the objects in the design description. The concept of this type of mapping was proposed and implemented by Elam and Lopez [4]. However, their mappings were hardcoded queries into a specific database. The mapping concept envisioned here is similar to that of Elam and Lopez, but instead of expressing the mapping in terms of a query, it is expressed in the form of a set of attribute names (present within the object-oriented design description) and a symbolical expression for combining those ingredients into a value for the basic data item. For example, consider the basic data item rx, the radius of gyration about the x axis, shown in Fig. 4. When the value of rx is requested by some other derived or logic data item, the mapping object responds to that message in the following manner.

1. The mapping object sends a message to an object, called the "context object", to return the name of the design description object for which the standard is being checked.

2. The mapping object then sends a message to this design description object, requesting values for the attributes named in the "DESIGN-ATTRIBUTES" slot of the mapping object.

3. The mapping object then evaluates the symbolic expression, with the given attribute values, to determine the value of the mapping data item.

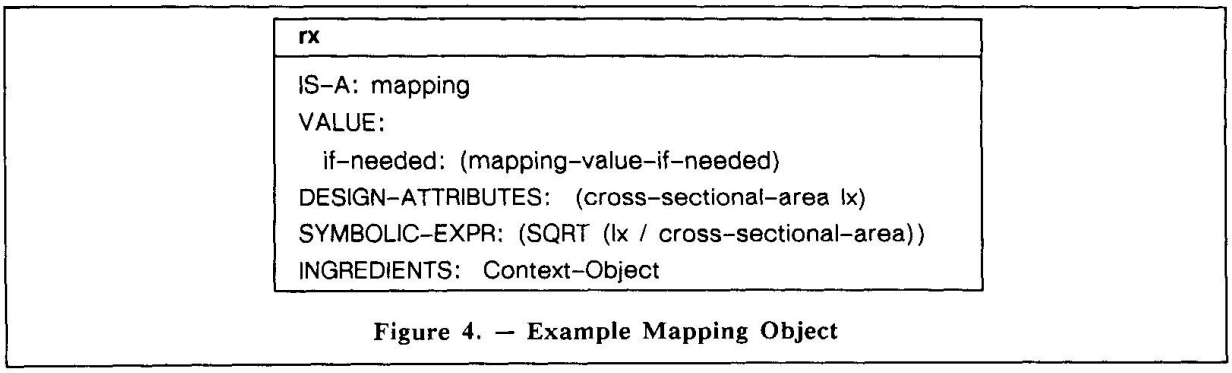
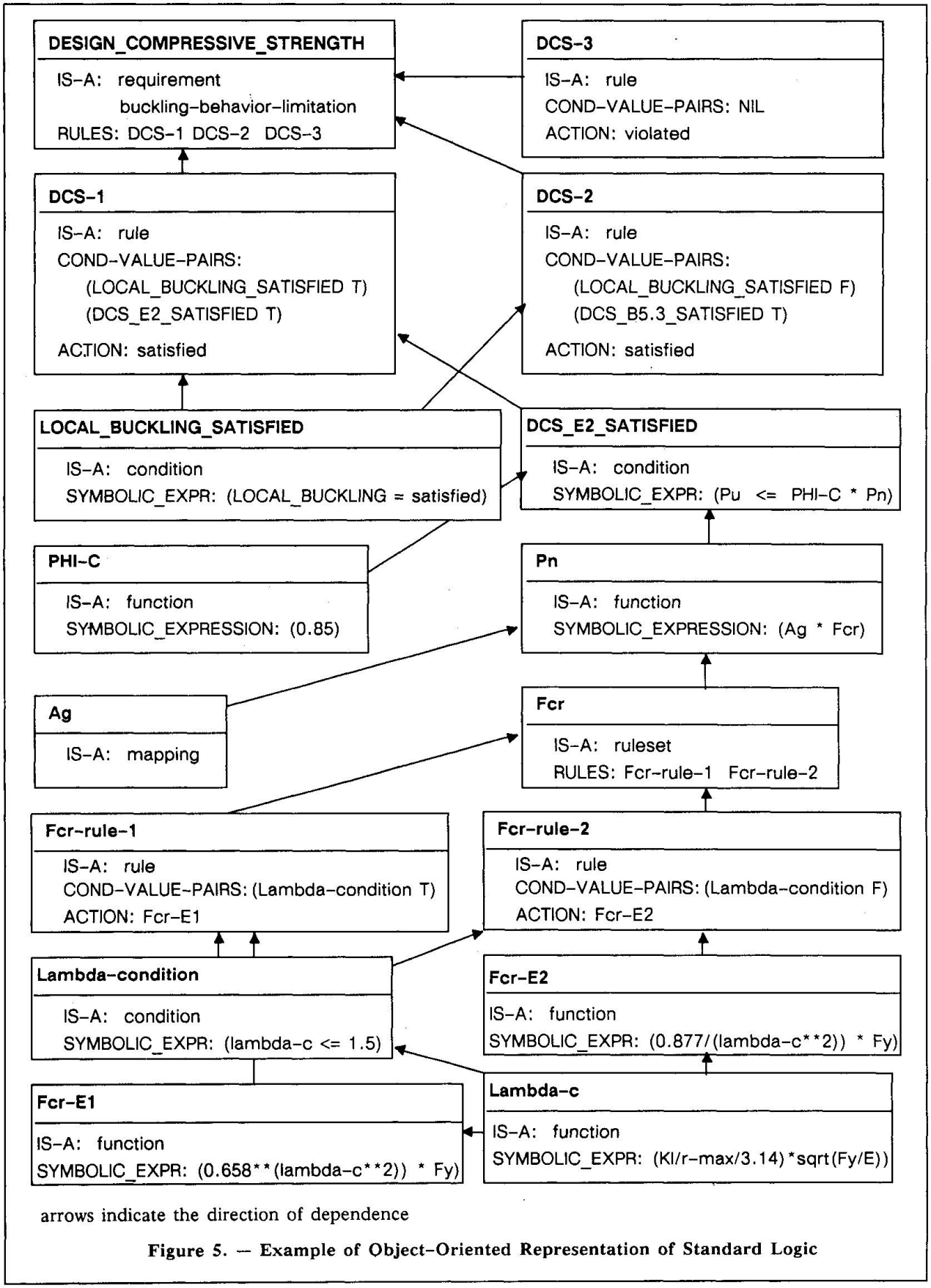


Figure 4. — Example Mapping Object



arrows indicate the direction of dependence

Figure 5. — Example of Object-Oriented Representation of Standard Logic

Note, that the expressiveness of the mappings is dependent on the attributes present within the design description. If the attributes needed to compute a basic data item are not present or available for reference, the default behavior is to query the user for the design description attributes. Also note that initially it has been assumed that all of the information needed for the determination of a basic data item can be found in the design description object that initiated the checking of the requirement. However, it is planned to extend this assumption as follows: the information needed is either present within the design description object or in some other object that is explicitly related to this object (i.e., by a part-of or connected-to relationship).

5. CONCLUSIONS

This paper describes an object-oriented representation of a design standard that provides more flexibility in the representation and usage of the information present within a design standard. All information within the design standard is represented as instances of predefined object classes. The benefits of having this declarative, object-oriented description of a design standard are: 1) it facilitates the automated checking of a design for conformance with a design standard, 2) it facilitates the ability to reason with and manipulate the requirements of a design standard during computer-aided design, and 3) it has the representational flexibility of an object-oriented approach. In addition, by having the design entities to which the standard applies represented as part of the model of a standard, it is possible to describe a set of mappings between the standard and this design description that will ensure proper interpretation of the basic data items in the standard, as well as the derived and requirement data items.

6. ACKNOWLEDGEMENTS

This material is based on work supported by the National Science Foundation under Grant No. DMC-8808132. The Government has certain rights to this material.

7. REFERENCES

1. *Load and Resistance Factor Design Specification for Structural Steel Buildings*, Amercan Institute of Steel Construction, Chicago, IL, 1986.

2. CRONEMBOLD, J. R. and K. H. LAW, "Automated Processing of Design Standards", *Journal of Computing in Civil Engineering*, Volume 2, Number 3, pages 255-273, July, 1988.

3. DYM, C. L., R. P. HENCHEY, E. A. DELIS and S. GONICK, "A Knowledge-Based System for Automated Architectural Code Checking", *Computer-Aided Design Journal*, Volume 20, Number 3, April, 1988, pp 137-145.

4. ELAM, S. L. and L. A. LOPEZ, "Knowledge Based Approach to Checking Designs for Conformance with Standards", Technical Report CESLRS No. 9, Department of Civil Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1988.

5. FENVES, S. J., "Tabular Decision Logic for Structural Design", *Journal of the Structural Division*, Volume 92, Number ST6, pages 473-490, June, 1966.

6. GARRETT, JR., J. H. and S. J. FENVES, "A Knowledge-Based Standard Processor for Structural Component Design", *Engineering with Computers*, Volume 2, Number 4, pages 219-238, 1987.

7. HOWARD, H. C., "KADBASE: An Expert System/Database Interface", in proceedings of The Fifth Conference on Computing in Civil Engineering, pp. 11-32, March 1988.

8. RASDORF, W. J. and T. E. WANG, "Generic Design Standards Processing in an Expert System Environment", *Journal of Computing in Civil Engineering*, Volume 2, Number 1, pages 68-87, January, 1988.