Building Information Modeling

Automated Code Checking and Compliance Processes

Nawari O. Nawari

CRC Press
Taylor & Francis Group

Building Information Modeling

Automated Code Checking and Compliance Processes

Building Information Modeling

Automated Code Checking and Compliance Processes

Nawari O. Nawari, Ph.D., P.E., F.ASCE
University of Florida, Gainesville

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper

International Standard Book Number-13: 978-1-4987-8533-4 (Hardback)
International Standard Book Number-13: 978-1-351-20099-8 (eBook)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Dedication

I dedicate this to my parents; O. Nawari: an engineer and architect, 1908–1976, and S. Orabi: 1918–2010.

Contents

Preface

The advancement of technology has uninterruptedly engaged the design profession over the past several centuries. Recently, the construction industry has started to engage information technologies more effectively to incorporate its design, construction, and operational processes. However, there are still manual processes, the duplication of business functions, and the sustained dependence on paper-based information management to record and exchange data among project participants, as well as the reviewing and verification of compliance with regulations. This book aims at the concept of automation in the building design process by utilizing building information modeling (BIM) to assist in computerizing and streamlining the communication and compliance verification of building design data against codes and regulations. Specifically, it represents the confluence of multilayered concepts and frameworks ranging from logic theories and cybernetics to BIM.

The book covers current and emerging trends in automating the processes of examining-construction documents against building codes and standards of practice. The role of BIM technologies in these methods and how this new technology is significantly transforming twenty-first-century practice activities in architecture, engineering, and construction (AEC) domains are thoroughly analyzed. The book also introduces the fundamentals of computerizing the process of verifying the compliances of building design documents against regulations and standards. These include domain knowledge representations, building model representations, and automated code-checking systems. An underlying goal for the material covered is to present the use of BIM technology as part of the computerized auditing process that can lead to a more comprehensive, intelligent, and integrated building design—a design where an optimized solution can be achieved in harmony with the current codes and standards of practice.

A new practical approach and a framework formulation for automating the code compliance–checking process is proposed and presented in this book. It provides a method to automatically or semiautomatically validate design documents against a specific building code or regulations based on data-driven design technologies. The framework enables synthesized streams of building codes and regulations data into a computable model that can interact with the building information model to yield results that are informative in a broad sense about compliances and co-generative in the design process. The suggested approach would read an object-based building model and audit the model against a selected set of object-based standard constraints. For building regulation clauses requiring subjective and qualitative performance, different methods need to be considered. On the one hand, the qualitative part of the building code requires feature extraction techniques of specific objective data objective concepts to enable full encoding. This includes the transformation of conditional and dependent building regulations into a set of rules using an object-based model. On the other hand, for the subjective and incomplete information in the building regulations, the framework proposes the feature extraction of all ambiguous information and uncertain data, and then employs partial encoding using fuzzy

logic, neural network algorithms, or combinations of these systems. This BIM-based approach for automating code conformance checking is one of the most powerful methods presently available that closely reflects real building code requirements and aims at accelerating the integration of the automated code conformance–auditing framework into the International and US National BIM Standard (NBIMS-US). Thus, the proposed approach contributes positively to advancing an open neutral automation environment for software developers, researchers, and AEC professionals. In other words, this book establishes the foundation for the Virtual Permitting (VP)_ process—the Era of VP.

Author

Dr. Nawari is an associate professor at the College of Design, Construction and Planning at the University of Florida, where he has served on the faculty of the School of Architecture. Dr. Nawari has more than 25 years of experience in design, teaching, and research, specializing in architectural structures and building information modeling (BIM). Currently, he teaches graduate and undergraduate architectural structures, structural modeling, and BIM courses at the University of Florida. Dr. Nawari has written and co-authored more than 150 publications and 3 books. He is a frequent contributor and speaker at American Society of Civil Engineers (ASCE) conferences. He has contributed to the engineering profession with several innovations during his career. Dr. Nawari's works open the door to new paradigms in teaching and designing building structures using the Structure and Architecture Synergy (SAS) framework. Also, his research in BIM standardization has led to major advancements in BIM standardization, particularly in the structural domain. Moreover, he has contributed significantly to the concept of encoding building rules and regulations and methods for automating building code conformance processes in BIM environments. He is a member of the BIM committee of the Structural Engineering Institute (SEI) and co-chairs the subcommittee on BIM in education. He is also a member of the NBIMS-US Committee and a fellow of the ASCE. Dr. Nawari currently serves as the assistant dean for graduate education at the College of Design, Construction and Planning, University of Florida. He is a board-certified professional engineer in the states of Florida and Ohio and has significant design and building experience.

1 Introduction

GENERAL

The advancement of technology has continuously engaged the design profession over the past several centuries. From the ancient Egyptians—who used models in the form of drawings and physical objects, as demonstrated in the plans of the tomb of Rameses IV and the drawing of the shrine from Ghorâb 4000 years ago—to the use of the mouse in the early 1970s, to the development of building information modeling (BIM) in the mid-1990s, technology has had a radical impact on building design and construction.

During the middle ages, models were used increasingly to verify the design and construction of cathedrals (Kostof, 1977) before building the full-scale facility. These models were an integral part of the design and decoration of building exteriors and interiors. The history of buildings finds its origins in the work of the Roman architect Vitruvius, who traced the origination of construction to the imitation or modeling of nature. He observed that, seeking shelter, humans learned lessons from swallows and bees, who built their own habitations. Then, humans started using natural materials to create forms based on shapes and proportions found in nature. An example of this is the "Vitruvian Man," which affirms that the figure of a man could be inscribed in both the circle and the square; this fundamental technological advancement represents the geometrical forms on which the design universe was ordered.

Later on, Romans created their structures on paper and then built physical models as replicas of the intended project before full-scale construction, to verify they would stand. Bad designs fell down; good designs stood (some of which are still around today). This practice of trial and error has come a long way over the past couple of thousand years, and the construction industry continues to seek tools that improve design before construction begins (Figure 1.1). Integrating BIM with computable building codes is a major step in this evolution.

Recently, the construction industry has started to engage information technologies more effectively to incorporate its design, construction, and operational processes. However, there are still manual processes, the duplication of business functions, and the sustained dependence on paper-based information management to record and exchange data among project participants, as well as the review and verification of compliance with regulations.

In efforts to resolve such issues, engineering ontology was introduced in the late nineties as a process that focuses on cognition-related activities to facilitate the creation, capture, exchange, conversion, and use of knowledge, with the ultimate goal of leveraging automation in engineering design and analysis decisions to achieve optimum solutions. In building design, there has been much research in the area of design knowledge reuse. In the past few decades, limited research efforts have

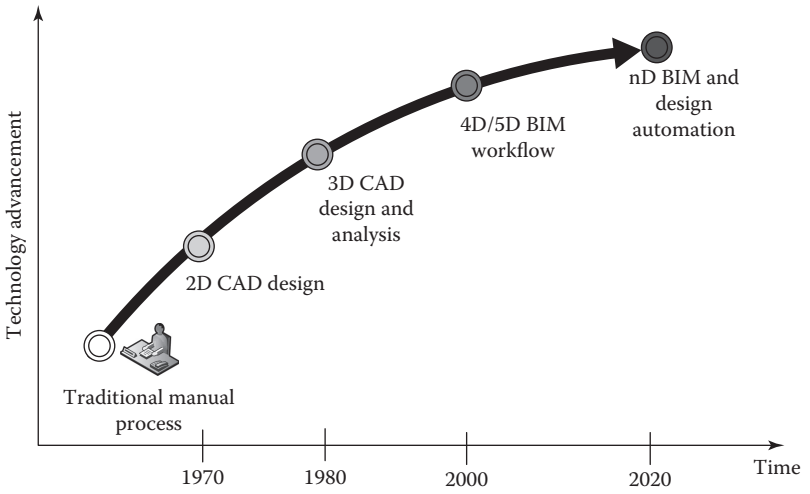


FIGURE 1.1 Evolution of building design tools and technology.

focused on handling the complexity of building code provisions and integrating different digital design tools to achieve automation. Most of the earlier research works were focused on the area of design knowledge reuse in building design, with most of the approaches previously tested originating from artificial intelligence (AI). In the seventies and eighties, a great number of rule-based expert systems were developed for the purpose of knowledge reprocessing. However, these systems were not entirely successful because of the difficulties in producing a formal representation of the information and keeping up with the frequent updating of knowledge. The knowledge first had to be acquired from building regulations and experienced designers and then generalized and transformed into rules. Although many of these systems are useful in solving the specific problem they are intended for, they are rarely used in practice.

The recent challenging global economic downturn accompanied by the continued growth in the complexity of building regulations and standards in a fragmented construction industry make the designing and delivering of a facility that meets the owner's objectives within budget and schedule a cumbersome task. The notion of computerization within the context of facility design in the twenty-first century offers key solutions to optimizing the building design process. By offering more accurate information in an open and asynchronous data format, computerization provides engineers, architects, and consultants with efficient and innovative methods to collaborate, investigate a large number of design alternatives, and validate design assumptions and requirements against code specifications in a virtual environment before construction to achieve optimum design objectives.

The concept of automation in the building design systems described in this book focuses on the mechanisms of checking building regulation compliance, which are defined by the relationship among various design and engineering information management systems and BIM, and how this computerization will assist in streamlining

the communication and dissemination of building design information among a breadth of stakeholders. Specifically, it represents the confluence of multilayered concepts ranging from logic theories and cybernetics to BIM.

In the architecture, engineering, and construction (AEC) industry, specifications and regulations are written by professionals to be read and applied by people. They typically take the form of written texts, tables, and equations. In general, these rules have lawful status. However, the cognitive and analytic ability of the human brain is dissimilar to anything implemented in computer systems. Thus, the automation of this process poses a real challenge to the AEC industry (Nawari and Alsaffar, 2015). For example, how can the interpretation of these rules into a computer-interpretable format be performed in a manner such that the implementation can be validated as consistent with the written regulations? Quite often, the process counts on the computer programmer's interpretation and translation of the written rules into computer code. In other cases, the logic of the human language statements is formally interpreted and then encoded into computer instructions.

Fortunately, new developments in AI research and BIM offer practical solutions to resolve these problems. AEC building standards and regulations commonly endeavor to organize, categorize, label, and define the rules, actions, and patterns of the building environment to attain efficiency, safety against any kind of failure, and overall economy. Nevertheless, their best-laid plans are overwhelmed by inevitable change, growth, innovation, progress, evolution, diversity, and entropy (Nawari, 2012b). Quite often, regulations can amend provisions and interpretive standards, which normally leads to massive volumes of semistructured documents that amend, complement, and potentially conflict with one another. These issues, which indicate complications for both young architects and engineers as well as experienced professionals, are also far more disorderly for the fragile traditional knowledge bases in computer systems. Notwithstanding that precise definitions and specifications are essential for encoding building regulations, many building code provisions aren't precisely defined and are often characterized by high subjectivity. Furthermore, some code provisions are characterized by continuous progressions and open-ended ranges of exceptions that make it difficult to give complete, exact definitions for any concepts that are learned through experience.

OVERVIEW OF AUTOMATED RULE-CHECKING SYSTEMS

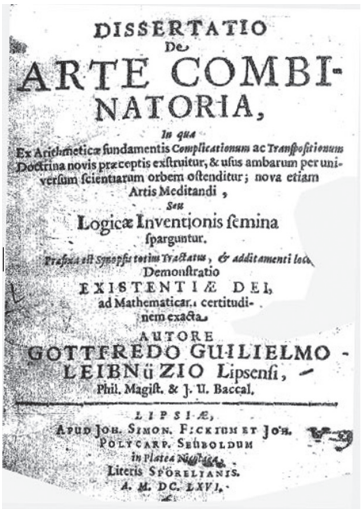
Automated rule verification systems are generally organized into four phases. Phase one is comprised of rule interpretation and development and the logical arranging of rules for their application. In the second phase, building model data generation is performed. This phase commences checking the necessary information required. Phase three is the execution phase, which carries out the actual rule compliance verification. The last phase is the reporting of the compliance-checking results.

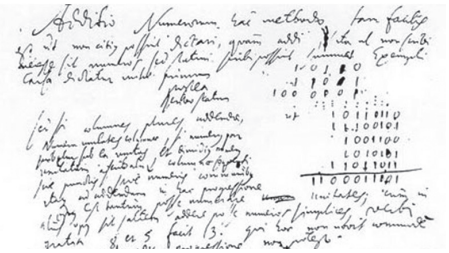
This issue of automating rules and regulations checking has interested many researchers and practitioners over the years. Historically, more than 2000 years ago, efforts were made to develop intelligent classification and verification systems, as depicted in Aristotle's categories and his system of syllogisms for reasoning about the categories. These were the most highly developed systems of logic and ontology

(Sowa, 2006). The syllogisms are rules of reasoning based on four sentence arrays, each of which relates one class in the subject to another category in the predicate (Nawari, 2012b). These rules are as follows:

- *Universal affirmative*: An example would be "Every truss is a frame."
- *Particular affirmative*: An example would be: "Some trusses are space frames."
- *Universal negative*: An example would be "No truss is a deep foundation."
- *Particular negative*: An example would be "Some space frames are not trusses."

In an effort to computerize the evaluation of Aristotle's syllogisms, in 1666, Leibniz was intrigued enough to try and develop the first computable model to appraise Aristotle's syllogisms. Leibniz analyzed the possibilities of the binary system to encode Aristotle's syllogisms. He demonstrated the four binary fundamental operations of calculation—addition, subtraction, multiplication, and division—expressing the conviction that one day in the future, machines would use this system to compute logical rules (Figure 1.2). Leibniz (1666) stated, "The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us verify by computation, without further argument, in order to see who is correct."


DISSERTATIO
De
ARTE COMBI-
NATORIA,
In qua
Ex Arithmeticæ fundamentis Complicationum ac Transpositionum
Doctrina novis præceptis exstruitur, & usus ambarum per uni-
versum scientiarum orbem ostenditur; nova etiam
Artis Meditandi,
seu
Logicæ Inventionis semina
sparguntur.
Præmissi ad Synopsin totius Tractatus, & additamenti loco,
Demonstratio
EXISTENTIÆ DEI,
ad Mathematicar. certitudi-
nem exacta.
AUTORE
GOTTFREDO GUILIELMO
LEIBNÜ ZIO Lipsensi,
Phil. Magist. & J. U. Baccal.
LIPSIÆ,
Apud Joh. Simon. Fickium et Jo'n.
Polycarp. Seuboldum
in Platea Nicolaitana,
Literis Sförelianis.
A. M. DC. LXVI.



(a) (b)

FIGURE 1.2 Leibniz's habilitation thesis in philosophy (1666). (a) Thesis cover page: "Dissertation on the Art of Combinations" and (b) Excerpt from Leibniz's thesis illustrating the binary system.

In regard to the construction industry, the first successful effort to automate design compliance was demonstrated by the work of Fenves (1966), when he investigated the application of decision tables to represent the American Institute of Steel Construction (AISC) standard specifications. He made the remark that decision tables, an *if–then* novel programming and program documentation technique, could be used to represent design standard provisions in a precise and unambiguous form. The concept was given a practical application in the 1969 AISC specification, represented as a set of interrelated decision tables. Later, other researchers tried to build on Fenves's work, such as Lopez et al., who implemented the Standards Interface for Computer Aided Design (SICAD) system (Lopez and Wright, 1985; Lopez et al., 1989). The SICAD system was a software prototype developed to demonstrate the checking of designed components as described in application program databases for conformance with design standards. The SICAD concepts were in production use in the AASHTO Bridge Design System (AASHTO 1998). Garrett developed the Standards Processing Expert (SPEX) system (Garrett and Fenves, 1987) using a standard-independent approach for sizing and proportioning structural member cross-sections. The system reasoned with the model of a design standard, represented using SICAD system representation, to generate a set of constraints on a set of basic data items that represent the attributes of a design to be determined.

Since then, there have been over 350 relevant research studies focusing on automating building code compliance, traversing over 30 years (Figure 1.3). Some of the most significant contributions are summarized in the following sections.

More focused research efforts on frameworks for the representation and processing of design standards for automated code conformance began two decades ago (Yabuki and Law, 1992; Kiliccote, 1996). During that time, building models and the methods for rule checking have been developed, but effective computable code systems have more recently begun to emerge. In the 1990s, the introduction of the Industry Foundation Classes (IFC) by buildingSMART International (formerly known as the International Alliance for Interoperability [IAI]) led to early research using this building model schema for building code compliance. The IFC specification is regularly maintained by buildingSMART International as its *data standard*. Since IFC4, it has been established as the international standard ISO 16739. Examples of these early efforts include the work by Han et al. (1998) and Vassileva (2000), who developed schema for a client–server approach. They later developed a simulated approach to the American Disability Act (ADA) wheelchair accessibility checking (Han et al., 1998, 2002). These research works set the stage for larger, more industrial-based efforts. In 1995, a research initiative was led by Singapore building officials, who started considering automating code compliance on two-dimensional (2D) drawings. In their next phase of development, they changed their approach and started the CORENET system, working with IFC building models in 1998. In the United States, similar works have been initiated under the Smart Code initiative. There are also several other research implementations of automated rule checking to assess accessibility for special populations (SMC, 2009) and for fire codes (Delis, 1995). The General Services Administration (GSA) and US courts have also supported the development of design rules checking for federal courthouses, which is

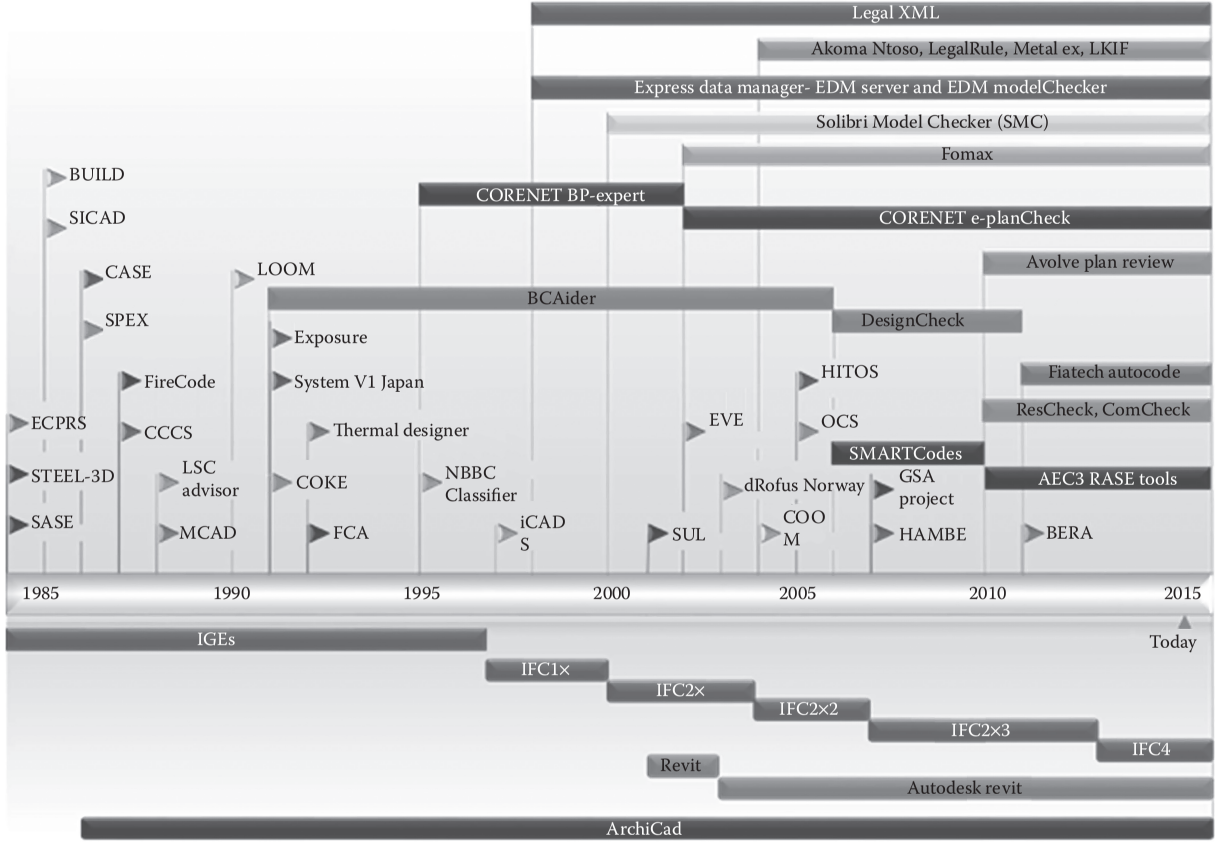


FIGURE 1.3 Timeline of related research to automated code checking. (Modified from Dimyadi and Amor, 2013.)

an early example of rule checking applied to automating design guidelines (GSA, 2007).

There are also a wide range of querying languages that have been proposed over the years for the automation of building regulations. These methods allow the extraction and manipulation of data in large models. These include, for instance, the Resource Description Framework (RDF), which is a directed, labeled graph data format for representing information on the Web, along with the SPARQL protocol, a well-established query language that has been officially standardized by the W3C (2005). RDF is the main building block of the Semantic Web effort, which enables the configuration of queries that traverse several repositories and thus effectively link information from different sources. In the automated code-checking context, such a mechanism would allow the ad hoc connection of submodels that reside in repositories owned and operated by the various stakeholders in a building project. Such an approach could furthermore be used as a starting point for the inclusion of software services that insert the results of specialized operations such as building regulation checking, performance calculations, or quantity takeoffs. Other more recent query languages include Building Environment Rule and Analysis (BERA) (2011) and BIMQL (2013). (See the section "Automated Code-Checking Systems" for more information.)

Other more recent efforts on computerizing building code rules are focused primarily on the concept of marking up regulatory texts to create a computable representation (Hjelseth and Nisbet, 2010a). Other research studies are centered chiefly on investigations of the automated or semiautomated extraction of information from regulatory texts into rules and other computable objects using AI techniques (Hjelseth and Nisbet, 2010b, 2011; Kiyavitskaya et al., 2006; Zhang and El-Gohary, 2011, 2012, 2013).

DOMAIN KNOWLEDGE REPRESENTATIONS

In recent years, many researchers have started to work on engineering information systems to provide an efficient means for knowledge domain representations of the construction industry. They have focused on many related issues such as the storage, retrieval, transfer, indexing, exchange, and utilization of data to enhance and streamline the design and construction process.

AEC codes and regulations are legal documents written and authorized by people to be understood and applied by professionals. They are hardly precise as formal logic. That elasticity of expression is essential for a system of knowledge acquisition. Yet professionals can read those documents and translate them into formal scientific notations and software applications (Figure 1.4). They can excerpt any type of information they need, reason about it, and apply it at various levels of precision. The means by which these extractions and applications are carried out is the area that researchers and professionals have tried to improve by introducing automated or semiautomated methods for many years. Most of the earliest efforts focused on transforming the knowledge domain from natural language into a formalized language such as first-order logic (FOL).

FIGURE 1.4 Domain knowledge representation.

In FOL, a predicate is a well-defined term (or function) that can be evaluated as TRUE, FALSE, or UNKNOWN (if terms are not defined). Also, predicate logic deals with quantification, whether a statement applies to ALL instances where a condition arises or it applies to at least one of the instances, in which case, the condition EXISTS. There are well-developed general techniques for translating logical assertions into executable statements, most importantly the Prolog computer language. However, the application of these methods in the AEC regulations is cumbersome and has many limitations. For example, the domain of buildings and the interpretation of where rules apply and how many instances of the rule need to be applied are major issues. Furthermore, many subjective provisions cannot be translated into FLO. One of the early applications of FLO was the implementation of decision tables in 1969 when representing the AISC specifications. The decision logic tables approach lends itself well to a technical standard with many objective data, such as the AISC specifications. This application was used as a design tool for steel structures for at least 15 years (Fenves et al., 1969). Additional efforts in this field resulted in the development of the Standards Analysis, Synthesis, and Expression (SASE) model by the National Institute of Standards and Technology (NIST; formerly known as the US National Bureau of Standards) in 1984, which represents one of the most significant early standards representation systems. SASE was employed to manage the creation and maintenance of decision tables and the structure of standards (Fenves et al., 1995; Lopez et al., 1989).

Other efforts focused on using expert systems or AI methods to encode regulatory data for use in building design (Eastman et al., 2009a,b; Frye et al., 1992; Mugridge et al., 1996; Rosenman et al., 1986). These systems were only useful if the underlying knowledge base was kept up to date with the updated regulatory provisions. Despite the intrinsic ineffectiveness and the dependence on manual updates, the investigations into the automated or semiautomated extraction of information from regulatory texts into rules and other computable objects using AI have continued until recently (Hjelseth, 2010; Kiyavitskaya et al., 2006; Zhang and El-Gohary, 2011, 2012).

Further approaches to computerized building code checking that are being investigated include markup document modeling and the use of hypertext to represent regulatory provisions (Turk and Vanier, 1995; Vanier, 1989). The concept of marking up regulatory texts to create a computable representation has been revisited in many research works (Bolioli et al., 2002; Lau et al., 2004; Hjelseth and Nisbet, 2011). For example, Lau et al. (2004) proposed the Extensible Markup Language (XML) as a unified format to represent regulations because of XML's capability of handling semistructured data such as legal documents (Figure 1.5).

As indicated previously, creating a proper computable representation of building codes and standards that derives data from legal sources and is kept updated is still far from ideal. Thus, in the absence of a practical computable representation of standards and regulations for the AEC industry, the quest for a better permanent solution continues.

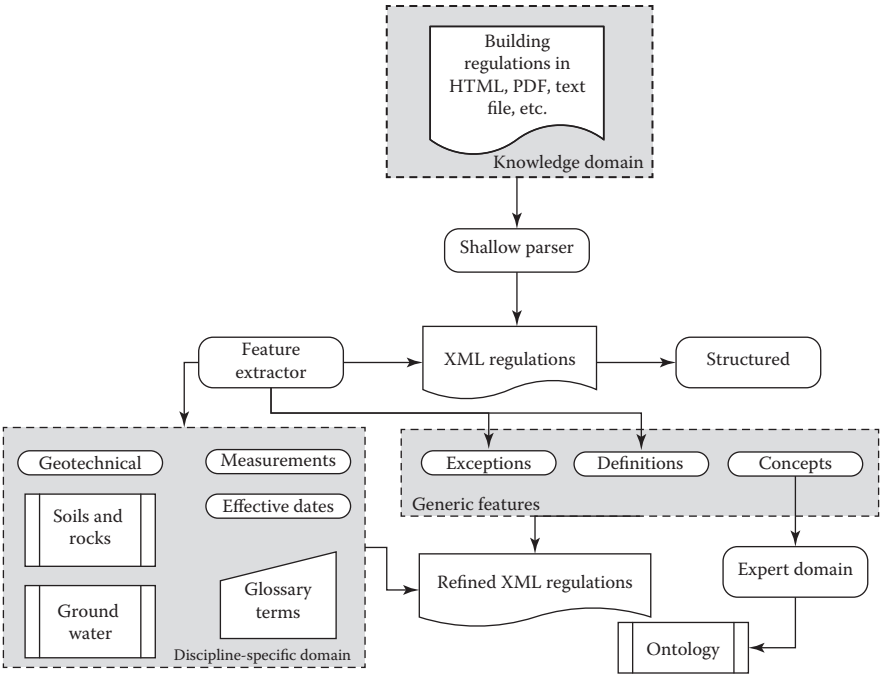


FIGURE 1.5 Development of regulations in XML. (From Lau, G. and Law, K., *Proceedings of 10th International Conference on Computing in Civil and Building Engineering* (p. 11), Weimar, Germany: Stanford University, 2004.)

BUILDING MODEL REPRESENTATION

Two-dimensional CAD systems have been used to represent building components since the early 1970s. Later on, these systems were developed into three-dimensional (3D) representation tools through systems such as RUCAP, Sonata, and REFLEX, as well as ArchiCAD and AutoCAD (Eastman et al., 2011). During the early 1990s, building model representations had started to shift into the object-based BIM paradigm (Hakim and Garret, 1993). Since then, the BIM process has fundamentally changed the role of computation in the AEC industry. Rather than using a computer to assist in producing a series of drawings that together describe a building, it introduced the concept of utilizing the computer to develop a single, unified database representation of the entire facility so comprehensive that it can generate all the necessary construction documentation (Nawari, 2015). BIM is a unified modeling approach used across the AEC industry.

The emergence of the BIM concept, along with the IFC open data format specification for building components, has provided a standard method and a generally agreed-upon protocol for the computable representation of a building. IFC2x4, or IFC 4, is the latest model specification, currently being accepted as the international standard ISO 16739 (Liebich, 2010). The IFC now represents the neutral standard data format for the building model. This part of automated code design verification is

the most advanced part of the automation process. Some areas of the AEC industry still need to be added to the IFC schema of the building model to enable comprehensive automatic code checking. These areas include, for example, foundations, subsurface conditions, and infrastructure systems.

AUTOMATED CODE-CHECKING SYSTEMS

After the successful implementation of AISC specifications by Fenves et al. (1969) as a series of logic decision tables, a number of researches have been encouraged to pursue the topic further. Examples include an advanced 3D graphical CAD system known as STEEL-3D for the design of steel frames to AISC specifications (Pesquera et al., 1984), a software tool developed at Carnegie Mellon University for the design of reinforced concrete beams (Noland and Bedell, 1985), an automated compliance-checking system developed at the University of Austin (Jaeger and Harelik, 1985), and computerized building standards research at VTT Finland (Kähkönen and Björk, 1987).

Major leaps in the area of automated code verifications took place after the advent of the IFC open data format for BIM in the late nineties (Han et al., 1998, 2002). Examples of these efforts include the development of the Express Data Manager (EDM) suite (now incorporating EDMmodelChecker), Solibri Model Checker (SMC), the Fornax plan checking tool, the Avolve plans review, the Design Data System (DDS), and so on. In 1995, the Building Construction Authority (BCA) of Singapore initiated the Construction and Real Estate Network (CORENET) electronic consent submission system, incorporating an in-house developed Building Plans (BP) Expert System to check 2D plans for compliance. The system was upgraded in 2002 to CORENET e-Plan Check, replacing the 2D BP Expert System with the 3D IFC data model (Khemlani, 2005).

Another automation system of the nineties that has survived the test of time is Design++. It has been developed into a knowledge-based design automation tool in combination with BIM. Design++ has been incorporated into a number of commercial products, including Bluethink's House Designer. Apart from giving advice to designers based on the set of requirements and rules, this system can also provide a set of generative rules for creating objects automatically (Huuskonen and Kaarela, 1995). In this system, like most of the expert systems of the nineties, all of the rule sets are hard coded into the application and can only be accessed within the application.

One of the latest efforts reported is the collaboration project between ICC, Solibri, Fiatech, and a few other software companies to develop AUTOCodes. This is currently a prototype system that promises integrated compliance-checking capability for US building model codes (Fiatech, 2012).

All of the approaches discussed so far, including current commercial systems, appear to have one factor in common. They all use an independent regulatory data representation either directly or via other dependent systems, and the exemplification is hard coded into the system and is subject to manual updates by the software developers. For example, CORENET e-Plan is using the Fornax library in conjunction with EDM, and has regulations and additional rules hard coded in EXPRESS.

Other recent automated code-checking systems are based on query languages. These include, for example, BERA (2011) and BIMQL (2013). The BERA language is a domain-specific modeling language that tries to deal with building information models in a natural approach in order to define and analyze rules even in the early design stages. The application of the BERA language aims to provide efficiency in defining, analyzing, and checking rules in the built environment (Figure 1.6). It is based on the BERA Object Model (BOM), which is an abstraction of the universe of discourse. BOM is a human-centered abstraction of the knowledge domain and it is one of the key concepts to the building environment rule checking and analysis (Lee, 2011). The BOM is derived from the IFC-oriented data model and represents a low-level implementation within the BERA as its backend. The BERA Language Tool is an integrated development environment for the BERA language. By using the BERA Language Tool, one can evaluate building models, focusing on both design analysis and rule checking of the purposes of building circulation and spatial programming (Lee, 2011).

One of the main limitations of BERA is the fact that it is a strictly domain-specific language. All the references to attributes and properties through an (Entity). (attribute/property) pattern (e.g., "Space.GUID") are hard coded into the grammar of BERA. Updates need to be performed manually for any additional subdomains.

BIMQL is an extendable, open, domain-specific query language for BIM. It can be used to interact with building information models in an automated code-checking

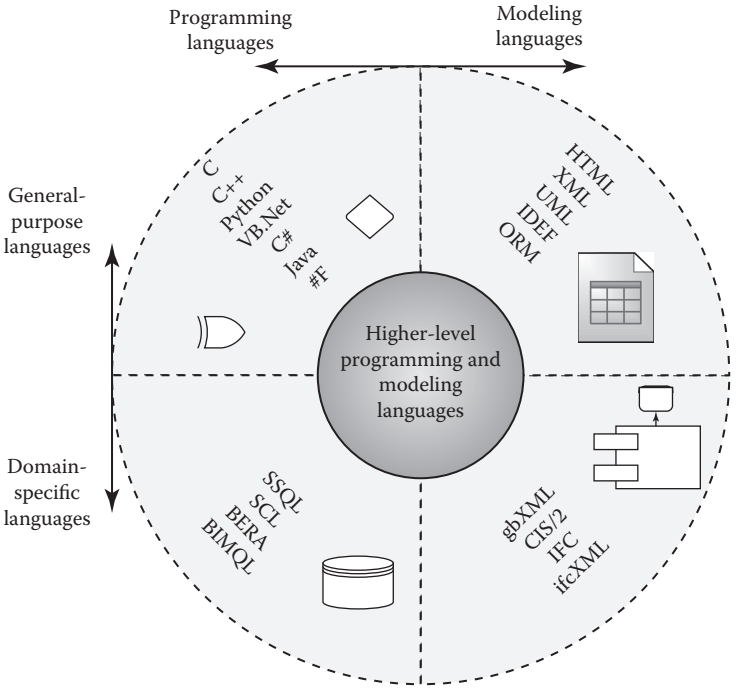


FIGURE 1.6 High-level computer languages.

environment. It is suggested as a flexible, ad hoc way for information exchange similar to SPARQL (2005) endpoints. The language has been designed and implemented on top of the bimserver.org platform, even though it is generic enough to be adapted in other implementations of IFC-based modeling and development tools. BIMQL queries are formulated spontaneously or composed from libraries in information requests communicated via network protocols. The current implementation and integration into the bimserver.org framework allows remote BIMQL queries via SOAP (Curbera et al., 2002), REST (Fielding and Taylor, 2002), and Google Protocol Buffers. The bimserver.org Java classes are generally used to store BIM models in the database and manipulate objects already stored there. Each class contains *getters* and *setters*, which are methods used to manipulate variables (Mazairac and Beetz, 2013). Figure 1.7 depicts an overview of the bimserver.org framework and the BIMQL plug-in.

Even though the language in its current state of specification and implementation is useful for submodel extraction scenarios, it has a number of limitations. For example, the query for model components by their commonly used terms in the AEC industry is currently limited to singular signifiers explicitly mapped in the buildingSMART data dictionary. An additional constraint of the present BIMQL language is the need to request all obligatory information aspects manually by defining individual query variables ($wall and $openingCoverInWall) or to collect this information by sequential queries or other forms of postprocessing (Mazairac and Beetz, 2013). Moreover, the language lacks the simplification needed to search for certain specific dependencies among building component entities. For instance, it is now very difficult with BIMQL to query all doors or windows related to one particular wall.

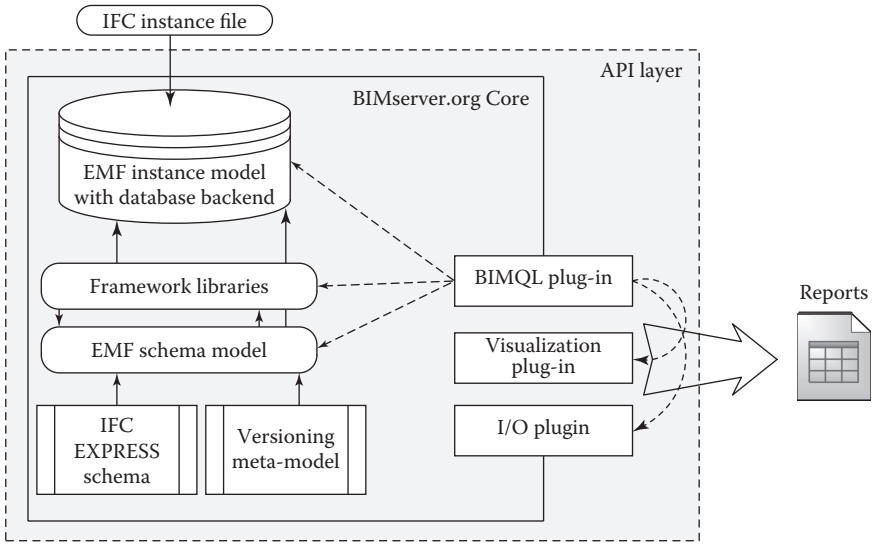


FIGURE 1.7 System overview of the bimserver.org framework, the BIMQL plugin. (Modified from Mazairac, W. and Beetz, J., *Advanced Engineering Informatics*, 27, 444–456, 2013.)

REFERENCES

AASHTO (1998). *AASHTO Guide for Design of Pavement Structures*, 4th Edition. American Association of State Highway and Transportation Officials.

Bolioli, A., Dini, L., Mercatali, P., and Romano, F. (2002). For the automated mark-up of Italian legislative texts in XML. In T. J. M. Bench-Capon, A. Daskalopulu, and R. Winkels (Ed.), *Legal Knowledge and Information Systems: Jurix 2002: The Fifteenth Annual Conference* (pp. 21–30). Amsterdam, the Netherlands: IOS Press. Retrieved from http://jurix.nl/pdf/j02-03.pdf.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. (2002). Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI. *Internet Computing, IEEE*, 6(2), 86–93. doi:10.1109/4236.991449.

Delis, E. and Delis, A. (1995). Automatic fire-code checking using expert-system technology. *Journal of Computing in Civil Engineering*, 9(2), 141–156. doi:10.1061/(ASCE)0887-3801(1995)9:2(141).

Dimyadi, J. and Amor, R. (2013). Automated building code compliance checking: Where is it at? In S. Kajewski, K. Manley, and K. Hampson (Ed.), *Proceedings of the 19th CIB World Building Congress* (pp. 172–185). Brisbane, Australia: Construction and Society. Retrieved from https://researchspace.auckland.ac.nz/docs/uoa-docs/rights.htm.

Eastman, C., Lee, J., Jeong, Y., and Lee, J.-K. (2009a). Automatic rule-based checking of building designs. *Automation in Construction*, 18(8), 1011–1033. doi:10.1016/j.autcon.2009.07.002.

Eastman, C., Lee, J.-M., Jeong, Y.-S., and Lee, J.-K. (2009b). Automatic rule-based checking of building designs. (M. Skibniewski, Ed.) *Journal of Automation in Construction*, 18, 1011–1033. Retrieved from https://www.researchgate.net/publication/222620888_Automatic_rule-based_checking_of_building_designs_Autom_Constr.

Eastman, C., Teicholz, P., Sacks, R., and Liston, K. (2011). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors* (2nd edn). Wiley. doi:10.1002/9780470261309.

Fenves, S. J., (1966). *Tabular decision logic for structural design*. J. Structural Engn 9(92), 473–490.

Fenves, S. J., Garrett, J. H., Kiliccote, H., Law, K. H., and Reed, K. A. (1995). Computer representations of design standards and building codes: U.S. perspective. *The International Journal of Construction Information Technology*, 3(1), 13–34. Retrieved from http://fire.nist.gov/bfrlpubs/build95/PDF/b95012.pdf.

Fenves, S., Goel, S., and Gaylord, E. (1969). *Decision Table Formulation of the 1969 AISC Specification*. Urbana, IL: University of Illinois. Retrieved from http://babel.hathitrust.org/cgi/pt?id=wu.89038865689;view=1up;seq=9.

Fiatech (2012). AutoCodes Project: Phase 1; Proof-of-concept final report. Construction Industry Institute, Cockrell School of Engineering, The University of Texas at Austin. Austin, TX: Fiatech Regulatory Streamlining Committee. Retrieved from http://www.fiatech.org/images/stories/techprojects/project_deliverables/Updated_project_deliverables/AutoCodesPOCFINALREPORT.pdf.

Fielding, R. and Taylor, R. (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2), 115–150.

Frye, M., Olynick, D., and Pinkney, R. (1992). Development of an expert system for the fire protection requirements of the national building code of Canada. *Proceedings of CIB W78 Conference* (pp. 215–226). Montreal, Canada. Retrieved from https://www.irbnet.de/daten/iconda/CIB12664.pdf

Garrett, J. H., Jr. and Fenves, S. J. (1987). A knowledge-based standard processor for structural component design, *Engineering with Computers*, 2(4), 219–238.

GSA (2007). U.S. courts design guide, Administrative Office of the U.S. Courts, Space and Facilities Division, GSA. Retrieved from http://www.gsa.gov/Portal/gsa/ep/contentView.do?P=PME&contentId=15102 &contentType=GSA_DOCUMENT.

Hakim, M. and Garret, J. (1993). A description logic approach for representing engineering design standards. *Engineering with Computers*, 9(2), 108–124. doi:10.1007/BF01199049.

Han, C., Kunz, J., and Law, K. (1998). A hybrid prescriptive/performance based approach to automated building code checking. *International Computing Congress* (pp. 537–548). Boston, MA: ASCE. Retrieved from http://eil.stanford.edu/publications/chuck_han/9810%20ICC.pdf.

Han, C., Law, K., Latombe, J., and Kunz, J. (2002). A performance-based approach to wheelchair accessible route analysis. *Advanced Engineering Informatics*, 16(1), 53–71. doi:10.1016/S1474-0346(01)00003-9.

Hjelseth, E. (2012). Converting performance based regulations into computable rules in BIM based model checking software. In G. Gudnason and R. Scherer (Ed.), *The 9th European Conference of Product and Process Modelling, eWork and eBusiness in Architecture*, *Engineering* and *Construction*, *ECPPM Conference* (pp. 461–469). Reykjavik, Iceland: CRC Press. doi:10.1201/b12516-73.

Hjelseth, E. and Nisbet, N. (2010a). Exploring semantic based model checking. *The Proceedings of the 2010 27th CIB W78 International Conference* (p. 11). Cairo, Egypt: CIB W78. Retrieved from https://www.academia.edu/873826/EXPLORING_SEMANTIC_BASED_MODEL_CHECKING.

Hjelseth, E. and Nisbet, N. (2010b). Overview of concepts for model checking. *The Proceedings of the 2010 27th CIB W78 International Conference* (p. 8). Cairo, Egypt: CIB W78. Retrieved from https://www.academia.edu/873824/Overview_of_concepts_for_model_checking.

Hjelseth, E. and Nisbet, N. (2011). Capturing normative constraints by use of the semantic mark-up (RASE) methodology. *Proceedings of CIB W78-W102 International Conference* (pp. 1–10). Sophia Antipolis, France: CSTB. Retrieved from http://2011-cibw078-w102.cstb.fr/papers/Paper-45.pdf.

Huuskonen, P. and Kaarela, K. (1995). Explaining plant design knowledge through means–end modelling. In Y. Anzai, K. Ogawa, and H. Mori (Ed.), *Symbiosis of Human and Artifact Future Computing and Design for Human-Computer Interaction: Proceedings of the Sixth International Conference on Human-Computer Interaction (HCI International '95)*, pp. 417–422. Tokyo, Japan: Elsevier. doi:10.1016/S0921-2647(06)80252-2.

Jaeger, S. and Harelik, L. (1985). Automation of the building code compliance. National Bureau of Standards Special Report. US Department of Commerce.

Kahkonen, K. and Bjork, B.-C. (Eds.). (1991). Computers and building regulations. *VTT SYMPOSIUM 125* (pp. 1–250). Espoo, Finland: Laboratory of Urban Planning and Building Design. Retrieved from http://www.gbv.de/dms/tib-ub-hannover/127958460.pdf.

Kähkönen, K., and Björk, B.-C. (1987). Computerization of Building Standards. Espoo, Finland: Technical Research Centre of Finland.

Khemlani, L. (2005). CORENET e-PlanCheck: Singapore's automated code checking. *AECbytes: Building the Future*. Retrieved from: http://www.novacitynets.com/pdf/aecbytes_20052610.pdf.

Kiliccote, H. (1996). *A standards processing framework*. PhD thesis, Dept. of Civ. And Envir. Engrg., Carnegie Mellon University, Pittsburgh, PA.

Kiyavitskaya, N., Zeni, N., Mich, L., Cordy, J. and Mylopoulos, J. (2006). Text mining through semi automatic semantic annotation. *6th International Conference, PAKM 2006 Proceedings*. 4333, pp. 143–154. Vienna, Austria: Springer. doi:10.1007/11944935_13.

Kostof, S. (1977). *The Architect: Chapters in the History of the Profession*. University of California Press, Oakland, CA, 1977.

Lau, G. and Law, K. (2004). An information infrastructure for comparing accessibility regulations and related information from multiple sources. *The Proceedings of 10th International Conference on Computing in Civil and Building Engineering* (p. 11). Weimar, Germany: Stanford University. Retrieved from http://eig.stanford.edu/publications/gloria_lau/icccbe.pdf.

Lee, J-K. (2011). Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program. Ph.D. thesis, College of Architecture, Georgia Tech.

Liebich, T. (2010). Unveiling IFC2x4: The next generation of OPENBIM. *Proceedings of the CIB W78 2010: 27th International Conference* (pp. 124–131). Cairo, Egypt. Retrieved from http://itc.scix.net/data/works/att/w78-2010-124.pdf.

Lopez, L. A. and Wright, R. N. (1985). *Mapping Principles for the Standards Interface for Computer Aided Design* (NBSIR 85-3115). Gaithersburg, MD: National Bureau of Standards.

Lopez, L., Elam, S., and Reed, K. (1989). Software concept for checking engineering designs for conformance with codes and standards. *Engineering with Computers*, 5(2), 63–78. doi:10.1007/BF01199070.

Mazairac, W. and Beetz, J. (2013). BIMQL: An open query language for building information models. *Advanced Engineering Informatics*, 27 (2013), 444–456.

Merry, A. and Spearpoint, M. (2008). The Building Act 2004 and the New Zealand Fire Service. *Proceeding of 7th International Conference on Performance-Based Codes and Fire Safety Design Methods* (pp. 47–58). Auckland, New Zealand: Society of Fire Protection Engineers (SFPE). Retrieved from http://ir.canterbury.ac.nz/bitstream/handle/10092/2011/12610896_The%20Building%20Act%202004%20and%20the%20New%20Zealand%20Fire%20Service.pdf?sequence=1&isAllowed=y.

Moulin, B. (1992). Automated generation of rules from the national building code text. *Proceedings of CIB W78 Conference* (pp. 343–358). Montreal, Canada: CIB W78. Retrieved from http://www.irbnet.de/daten/iconda/CIB12680.pdf.

Mugridge, W., Hosking, J., and Amor, R. (1996). Adding a Code Conformance Tool to an Integrated Building Design Environment. Auckland, New Zealand: Auckland Uniservices. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.731&rep=rep1&type=pdf.

Nawari, N. O., Alsaffar, A. (2015). *Practical Approaches for Computable Building Codes.* Proc. of the 32nd CIB W78 Conference 2015, 27th-29th 2015, 569-576. Eindhoven, The Netherlands.

Nawari, N. O. and Alsaffar, A. (2015). Methods for computable building codes. *Journal of Civil Engineering and Architecture*, 3, 163–171. doi:10.13189/cea.2015.030601.

Nawari, N. (2012a). The challenge of computerizing building codes in a BIM environment. In R. Issa and I. Flood (Ed.), *International Conference on Computing in Civil Engineering* (pp. 285–292). Clearwater Beach, FL: American Society of Civil Engineers (ASCE). doi:10.1061/9780784412343.0036.

Nawari, N. (2012b). Automated code checking in BIM environment. In V. Telichenko, A. Volkov, and I. Bilchuk (Ed.), *The Proceedings of the 14th International Conference on Computing in Civil and Building Engineering* (pp. 104–106). Moscow, Russia: ICCCBE. Retrieved from http://www.gbv.de/dms/weimar/toc/726596442_toc.pdf.

Nawari, N. (2012c). Automating codes conformance. *Journal of Architectural Engineering*, 18(4), 315–323. Retrieved from http://ascelibrary.org/doi/abs/10.1061/(ASCE)AE.1943-5568.0000049.

Noland, J. L., and Bedell, R. (1985). *Automated Checking of Simply-Supported Prismatic Reinforced Concrete Beams for Compliance with Code Requirements*. National Bureau of Standards Special Report. 71–82.

Pesquera, C., Hanna, S., and Abel, J. (1984). Advanced graphical CAD system for 3D steel frames. In *Computer Aided Design in Civil Engineering* (pp. 83–91). New York: American Society of Civil Engineers. Retrieved from http://cedb.asce.org/cgi/WWWdisplay.cgi?41831.

SMC, (2009). *Automated code checking for accessibility*. Solibri. http://www.solibri.com/press-releases/solibri-model-checker-v.4.2-accessibility.html

Sowa, J. F. (2006) The challenge of knowledge soup. In J. Ramadas and S. Chunawala (Ed.), *Research Trends in Science, Technology and Mathematics Education*. Mumbai, India: Homi Bhabha Centre. Retrieved from http://www.jfsowa.com/pubs/challenge.pdf.

Turk, Z. and Vanier, D. (1995). Tools and models for the electronic delivery of building codes and standards. *Proceedings of CIB W78 Conference* (pp. 20–30). Stanford, CA: CIB W78. Retrieved from http://itc.scix.net/data/works/att/w78-1995-20.content.pdf.

Rosenman, M.A. , Gero, J.S. , and Oxman, J. (1986). *An expert system for design codes and design rules*. B. Sriram, R. Adey (Eds.), Applications of artificial intelligence in engineering problems, Vol II, Springer-Verlag.

Vanier, D. (September, 1989). Computerized building regulations. *Proceedings of the International Conference on Municipal Code Administration Building Safety and the Computer* (pp. 43–62). Winnipeg, Manitoba: National Research Council Canada. Retrieved from https://www.researchgate.net/publication/44076935_Computerization_of_building_regulations.

Yabuki, N. and Law, K. H. (1992). An integrated framework for design standards processing. Technical Report 67, *Center for Integrated Facility Engineering*, Stanford University, Stanford, CA.

Zhang, J. and El-Gohary, N. (2011). Automated information extraction from construction-related regulatory documents for automated compliance checking. *Proceedings of the 28th International Conference of CIB W78* (pp. 1–10). Sophia Antipolis, France: CIB W78. Retrieved from http://itc.scix.net/cgi-bin/works/Show?w78-2011-Paper-99.

Zhang, J. and El-Gohary, N. (2012). Extraction of construction regulatory requirements from textual documents using natural language processing techniques. In R. Issa and I. Flood (Ed.), *International Conference on Computing in Civil Engineering* (pp. 453–460). Clearwater Beach, FL: ASCE American Society of Civil Engineers. doi:10.1061/9780784412343.0057.

Zhang, J. and El-Gohary, N. (2013). Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking. *J. Comput. Civ. Eng.*, 2016, 30(2): 04015014-1–04015014-14, ASCE. doi:10.1061/(ASCE)CP.1943-5487.0000346.

2 Domain Knowledge Representations

BACKGROUND

An automated code compliance–checking system is a computational procedure for addressing the manual regulation verification problem in a finite number of computable steps. It involves deduction, induction, abstraction, generalization, and structured logic. It is the systematic extraction of logical rules from textual documents and the development of a generic execution plan. Furthermore, it develops algorithmic strategies to search for repetitive patterns, universal principles, interchangeable modules, and inductive links. The rational power of such systems lies in their ability to allow for the machine interpretation of building regulations, to infer results where data is incomplete, and to extend certain limits of the human intellect. Thus, these automated compliance-checking systems suggest rationality, consistency, coherency, organization, and systemization.

The first step in the automated process of compliance checking of building codes is the computable representation of the context and content of building regulations. This involves an interpretation process where the semantic structure of each regulation is translated into rules or parametric models using certain formal languages. These are then interrogated and acted on by bespoke software. The next step deals with the need to link these representations and relate to the building information modeling (BIM) data being examined.

The automated compliance-checking process aims to improve the actions of the manual design review by rationalizing the information of a predictable outcome. It involves the action of deducing and verifying specific details automatically, which generally can be very time-consuming when carried out manually on the initial data set of properties and conditions. A computable model serves as the means through which an explorative compliance-checking process can be undertaken.

Traversing the second half of the twentieth century, various formal models of language were proposed to deal with building codes and regulations. These models have been useful for processing some aspects of the knowledge domain, but none of them have been adequate for all issues of the subject or even for full coverage of just a single aspect. Some of these major computable models are depicted in Table 2.1.

The development of computable building code is a key factor in improving the current design practice by abridging the communication to code provisions and conformance auditing. Representing building codes and standards in a machine-readable format that accepts and understands the specific characteristics of the knowledge domain plays a vital role in the automation of the building code conformance verification process. The computable digital schema of the building rules and specifications permits automated provisions

TABLE 2.1
Comparisons between Existing Main Computable Models of the Building Codes

Country	Target Domain	Checking Platform	Computable Model	References
Singapore (CORENET)	Building code	FORNAX	Predicate logic	Khemiani (2011)
Norway (Statsbygg)	Accessibility	Solibri Model Checker	Decision tables	Sjogren (2007)
Australia (DesignCheck)	Disabilities (AS1428.1)	EDM	Ruble-based language	Lan Ding et al. (2006)
Portugal (LicA)	Water system	LicA	XML-based language	Monteiro (2013)
Canada (ACCBEP)	Building envelope	Rule Engine	XML-based language	Tan et al. (2010)
Korea (GTPPM)	Fire resistance	Checking Engine	Direct hard-coding	Jeong and Lee (2010)

compliance checking without changing a building design, but rather assesses a design on the principle of the compliance of parametric objects, their relations, or their attributes. It does involve rule-based systems to a suggested design, and produces results in the format of "SUCCESS," "FAILURE," "WARNING," or "UNIDENTIFIED" for conditions where the mandatory data is inadequate or missing.

It is evident that computable building regulations center on data readiness and rules development. Each of these factors has its own characteristics and restriction aspects. The main constellation of difficulties stems from the nature of building regulations and standards. For example, building codes are not self-contained documents and often reference many other sources. This signifies that most of the regulations in a building code or standard refer to knowledge that all professionals should be familiar with. Unfortunately, such data is not always represented in a formal expression. Moreover, comprehending a design standard requires some knowledge about the design field under consideration. In AEC professional disciplines, fundamental scientific knowledge (the basic knowledge that engineers and architects acquire in their education) is expected from the users of an engineering design standard (Nawari and Alsaffar, 2015). Additionally, knowledge and heuristics are required to decide when to inspect another referenced standard and when to advance based on presumed compliance. In summary, the features that characterize these building provisions contain

- Subjectivity issues
- Inconsistent usage of terminologies
- Complexity of code structuring, exceptions, and the various interrelationships

These properties make the auditing of a building design for conformance with these building regulations a complex and time-consuming activity, and it is quite often error prone and reliant on the professional's experience, judgment, and skills. They also, by their nature, do make the transition to automated compliance checking a cumbersome process. Thus, the development of a computable model of building regulations must be semantically rich and object oriented so that it can be appropriate for automated compliance checking. This means leveraging object-oriented BIM data coupled with the Industry Foundation Classes (IFCs) as an interoperability standard can provide key solutions.

HUMAN LANGUAGES

Human languages are easy to learn by children; they can express any thought that any adult might conceive, and they are adapted to the limitations of human breathing rates and short-term memory (Sowa, 2007). *Human language* implies that it is a language that any cognitively normal infant is able to acquire and whose initial development has been through use rather than prescription. Seven-year-old children are almost as proficient at speaking and understanding as their parents. This shows that with a limited vocabulary, they enjoy the countless extensibility of expressions and lengths of phrases. Together, they indicate that many words in a human language will have an open-ended number of senses, and thus vagueness and ambiguity are unavoidable. Some of the main characteristics of human languages include (Figure 2.1)

- *Phonology*: This refers to how words are related to sounds.
- *Morphology*: This means how words are built from more primitive morphemes (e.g., how "friendly" comes from "friend"). Morphology deals with the different inflections of a word and the forms it can take: a noun can be singular or plural, a verb can have different tenses, and so forth.

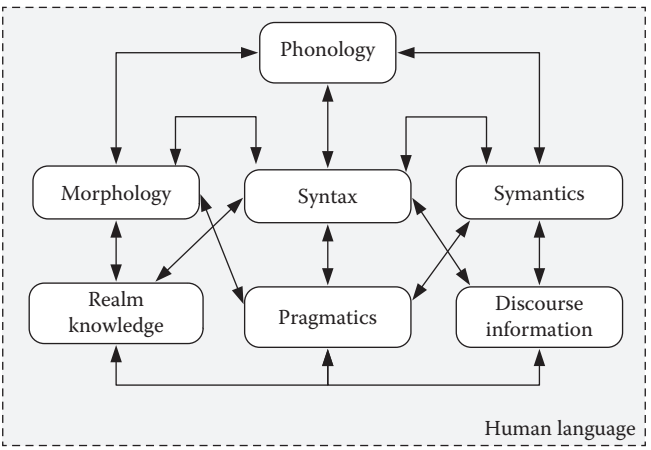


FIGURE 2.1 Characteristics of human language.

- *Syntax*: This refers to how sequences of words form correct sentences. It is knowledge of the rules of grammar.
- *Semantics*: This denotes how words have "meaning"—how words have reference (denotation) and associated concepts (connotations).
- *Pragmatics*: This refers to how sentences are used in different situations and how use affects the interpretation of the sentence, which involves the intentions and context of the conversation.
- *Discourse information*: This represents how preceding sentences determine the meaning of a sentence, as in the case of the referent of a pronoun.
- *Realm knowledge*: This include perceptions, emotions, beliefs, and general knowledge about the issues in a conversation.

Sounds and sound patterns, the basic units of meaning, such as words, and the rules to combine them to form new sentences constitute the grammar of a language. Languages evolve and diversify over time, and the history of their evolution can be reconstructed by comparing modern languages to determine which traits their ancestral languages must have had in order for the later developmental stages to occur. Every language's grammar is equally complex and logical and capable of producing an infinite set of sentences to express any thought. Thus, it is very difficult in human language to speak of a single meaning for a sentence; rather, there is a probability of possible meanings. Furthermore, human vocabularies are very large and constantly change with time. Therefore, it is impossible for a human to understand every text or dialog in his or her native language. Thus, it is also unrealistic to expect a computer to do so.

Humans have a remarkable ability to learn and to extend their understanding without explicit training. Fundamental to human understanding is the ability to learn and use language in extensible ways.

In essence, the difficulties in dealing with human languages do not necessarily arise from the way the human brain works or the way that natural languages express information. Whitehead (1937) noted that it is the outcome of leftover questions lurking in "the penumbral background" and the trouble of recognizing which ones are relevant to the "focus of experience" (Swoa, 2007).

- *Overgeneralizations*: The *live load* in building structures is uniformly distributed in floor areas. But what is the case when the live load is always in a specific location of the floor? What if animals of different weight are going to occupy the floor?
- *Abnormal conditions*: If you have a car, you can drive from New York to Boston. But what if the battery is dead, your license has expired, or there is a major snowstorm?
- *Incomplete definitions*: *Panelized assemblies* are defined in the National Green Building Standard (ICC 700-2008) as factory-assembled wall panels, roof trusses, and/or other components installed on-site. Does this also mean columns and beams? What about windows and doors?
- *Conflicting defaults*: Trusses are pinned and connected; moment frames are not. But what about Virendeel trusses, which can have both pinned connections and moment connections?

- *Unanticipated applications*: Structural elements are described in structural engineering books. But is a door a part of the structural elements? Windows? What about curtain walls?

These exceptions and borderline cases result from the nature of the universe, not necessarily from defects in natural languages. Part of the difficulty stems from the discrepancy resulting from a mismatch of the continuous world with the discrete words of a natural language. All languages consist of discrete symbols organized in discrete syntactic patterns; the real world, however, contains an endless variety of things, events, forms, substances, gradations, changes, and continuous flows with imperceptible transitions from one to another. It would be very cumbersome for any language based on discrete words or symbols to capture the full complexity of continuous real-world systems.

Through history, several philosophers (e.g., Charles Sanders Peirce and Ludwig Wittgenstein) have discerned that vagueness and ambiguity are important components of human languages and have determined that these are not defects in a natural language but vital characteristics that enable it to express a variety of concepts and all the aspects of objects that humans need to describe. This not only indicates the ineffectiveness of any effort to develop a precisely defined ontology of everything, but it also provides two valuable alternatives: (1) formal classification, such as a thesaurus or terminology, and (2) an open-ended collection of formal theories about narrowly delimited subjects. It also raises the question of how and whether these resources might be utilized as a link between natural languages and formally defined logics and computer programming languages.

The overarching goal of formal languages is to represent as adequately as possible the existing problem domain in order to better approximate the objective functions. Some of the major formal languages that have been developed over the years indicate the tireless efforts by researchers to develop a formal language to capture the full complexity of continuous real-world systems. These include statistics, syntactics, binary and fuzzy logic, lexical semantics, statistics, neural networks, and genetic algorithms. In the 1950s, Claude E. Shannon's information theory and other statistical methods were popular in both linguistics and psychology, but the speed and storage capacity of the early computers were not adequate to process the volumes of data required. By the end of the century, the vastly increased computer power made them competitive with other methods for many purposes. Their strength is in pattern discovery methods, but their weakness is in the lack of a semantic interpretation that can be mapped to the real world or to other computational methods.

Most of these formal languages seek to represent as adequately as possible the given problem domain knowledge in order to better approximate the goal function. These languages strive to develop processes of mapping the knowledge domain space into the solution space by using heuristic rules and data. Statistical representations can be effective when statistical data are available and the underlying type of target function is known. Artificial intelligence (AI) methods such as symbolic rule-based systems can be used effectively when the problem domain knowledge is in the form of well-defined, rigid rules, when no adaptation is possible or difficult to implement. Neural networks are applicable when the domain knowledge includes

data without having any information as to what type of objective function might be; they can be used to learn heuristic rules after training with the given data. The fuzzy logic approach is applicable to model the domain knowledge when it includes heuristic rules with vague, ill-defined, or approximate conditions. Figure 2.2 illustrates the main areas of application of these formal languages.

Each of these modeling languages is based on a particular technology: mathematical statistics, grammar rules, dictionary formats, fuzzy logic, or networks of neurons. Each of them disregards those characteristics of a language for which the technology is ill-suited. For people, however, language is effortlessly integrated with every aspect of life, and they don't blunder over borders between different technologies. The chief strength of natural language is its flexibility and power to express any sublanguage ranging from composing music or writing software code to stock-market statistics and probability computations.

For over 2000 years, efforts were made to create a formal language system as represented in Aristotle's categories, and his system of syllogisms for reasoning about the categories were the most highly developed system of logic and ontology. The syllogisms are rules of reasoning based on four sentence patterns, each of which relates one class in the subject to another category in the predicate (Nawari, 2012).

- *Universal affirmative*: Every truss is a frame.
- *Particular affirmative*: Some trusses are space frames.
- *Universal negative*: No truss is a deep foundation.
- *Particular negative*: Some space frames are not trusses.

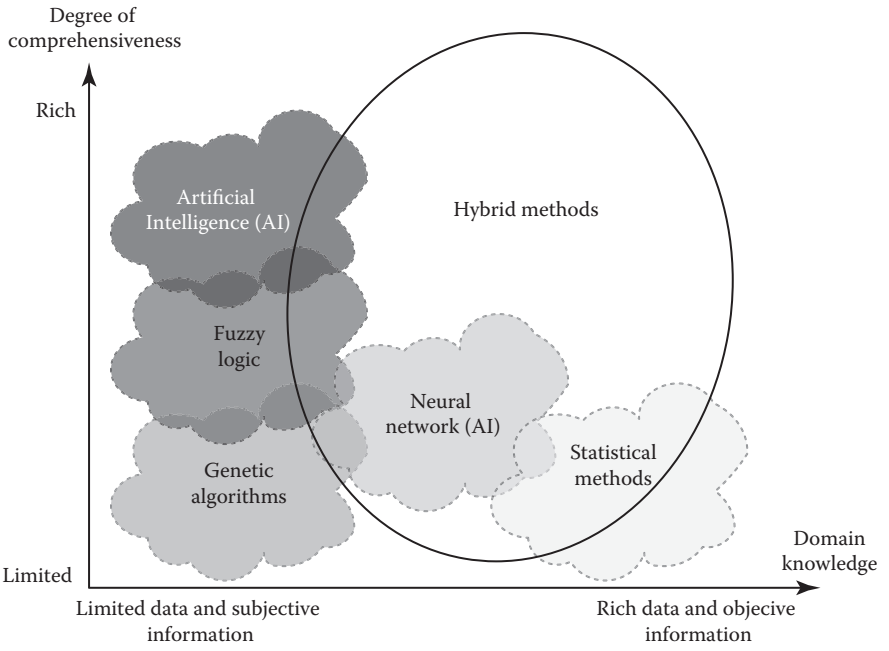


FIGURE 2.2 Application of formal languages.

In 1666, Leibniz was fascinated with trying to create the first computable formal language model to automate Aristotle's syllogisms: "The only method to remedy our reasonings is to make them as concrete as those of the Mathematicians, so that we can find our errors at a glance, and when there are disagreements among people, one can merely say: Let us compute, without further argument, in order to see who is correct."

Recently, a number of researchers have investigated the application of the ontology-based approach (Yurchyshyna et al., 2009) and Semantic Web information as a possible formal language framework (Pauwels et al., 2009) for computerizing building regulations and standards. The first research approach works on formalizing conformance requirements conducted under the following methods (Yurchyshyna et al., 2009): (1) knowledge extraction from the texts of conformance requirements into formal languages (e.g., XML, RDF), (2) the formalization of conformance requirements by capitalizing the domain knowledge, (3) the semantic mapping of regulations to industry-specific ontologies, and (4) the formalization of conformance requirements in the context of the compliance-checking problem. On the other hand, the Semantic Web method centers on improving the IFC model by utilizing descriptive language based on a logic theory and rules such as the one found in the Semantic Web domain.

AEC building codes and standards are lawful documents, composed and accepted by experts, and finally consumed by professionals. These texts are generally in natural language formats, typically in the form of English text, mathematical formulas, tables, and other provisions with legal status. They are barely precise as formal languages. That elasticity of the text is crucial for a system of knowledge acquisition, yet engineers and architects can consume those documents and translate them into formal scientific representations and software applications. They can extract any type of data they want, reason about it, and apply it at various phases of execution. The methods by which these extractions and applications are performed is an area that researchers and professionals have tried to automate or semiautomate for many decades (Nawari, 2015).

Until recently, most of the previous efforts on modeling the formal language representation of building regulations have only focused on the syntax and grammar of rules. However, understanding the meaning of the code provisions is also important, which requires experts' knowledge and experience to interpret the semantics of the regulation rules. For instance, CORENET e-PlanCheck (Singapore Civil Defense, 2002), which utilizes a logic-based interpretation approach, in transforming the provisions from natural language into more precise formal language. During the interpretation procedure, implied assumptions and expectations are discovered, which could help to finish the comprehension of what requires to be examined. However, this process may lead to underestimations of the complexity of the building regulations involved in the execution of the compliance verification process.

It is obvious that the manual code compliance–auditing process produces a great deal of inconsistencies, because human assessments always fill in vagueness, incorporating experience and intuition. Ontology and BIM standardization are vital factors in resolving these issues and support the transition from human efforts to more

machine-oriented automatic processes, to create consistent, accurate, and measurable conditions and constraints for each rule compliance.

The most effective method to achieve the computerized execution of regulations and specifications in building informatics is to implement modeling languages that have the ability to create computer-interpretable regulation and rules—that is, rules that a machine can read and interpret automatically (Figure 2.3). Currently, investigators are studying different modeling techniques for the generation of a formal

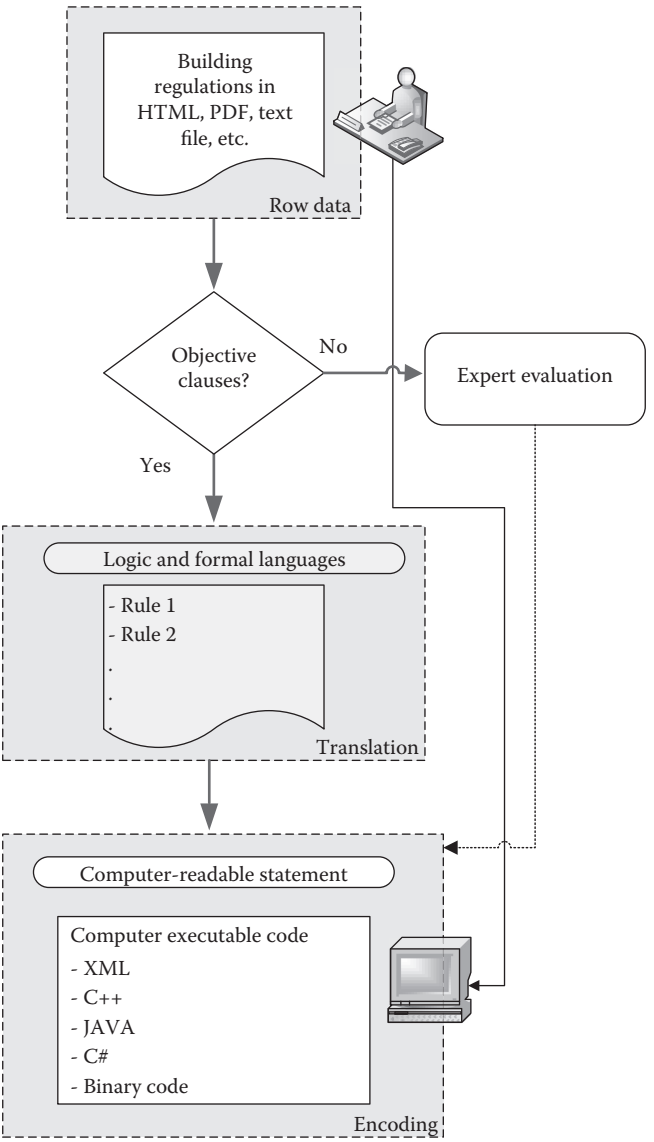


FIGURE 2.3 Developing computer-interpretable building codes.

language—that is, computer-executable rules from regulations written in human language. The following is a summary of the two major modeling methodologies for automating building code compliance checking.

ARTIFICIAL INTELLIGENCE METHODS

The objective of AI has always been to simulate, in some form or other, human intelligence, knowledge, and perception. This simulation has focused on two major areas: the first one deals with learning relationships from a set of given data (also known as the *biologically inspired AI approach*), and the second area focuses on encapsulating and reusing knowledge that humans have already acquired.

The first biologically inspired AI system was the *perceptron method*, introduced in 1957 by Frank Rosenblatt, when he envisioned copying the behavior, as it was then understood, of individual brain cells known as *neurons*. At the same time, other researchers, particularly Marvin Minsky (1970), were working on the first AI systems encapsulating human knowledge as rules. The competition between these two approaches (biologically inspired AI and rule-based AI) began when Minsky and Pappert wrote a paper pointing out the limitations of the neuron-based AI approach and that discussion has continued until today.

Rule-based AI systems were therefore at the forefront of the research for many years. In 1986, David E. Rumelhart and James L. McClelland published their book *Parallel Distributed Programming*, which demonstrated that neural networks could learn the class of problems the perceptron couldn't. Furthermore, they showed that neural networks could learn any arbitrary function that didn't disappear off to infinity at some value and could perform better than biologically inspired AI systems.

Most of the original formulations of AI systems are based on Minsky's work and represent knowledge as a mixture of facts and rules. The facts were pairs of data items and their values. The rules operated on facts and allowed the system to infer new facts from the given sets of data. The current approach works very similarly, and if one changes anything, such as a fact taking on a new value, then everything in the system could potentially change. Conventional AI systems use the Rete algorithm like a net to reveal from the given data any rules that rely on the changed data item. These rules are then fired, and any data items that are changed are added back to the data set. Again, the net is used to search for rules using these new data items, and the process is continued until no data items change.

The AI methods of modeling human languages utilize various natural language processing (NLP) algorithms to drive meaning from the provided natural language text. NLP as a discipline has been developing for many years. It began in 1960 as a subfield of AI and linguistics, with the purpose of studying problems in the automatic generation and understanding of human language. NLP algorithms generally center on the area of human–computer interaction. Numerous challenges in NLP methods include natural language understanding and extracting rules from building regulation documents. They seek to allow computers to derive meaning from textual formats such as building regulations and to generate logical rules for further processing. They rely on the predicted probability distribution of the language expressions being extracted. A model of the probability distribution of an *n*-letters sequence is

normally referred to as an *n*-gram model. For example, $P(a_{i:N})$ is the probability of a sequence of N characters a_1 through a_N.

The *n*-gram model is normally defined as the Markov chain of order $(n-1)$. In this Markov chain, the probability of a_i characters depends only on the immediate preceding characters (Russel and Norvig, 2010).

$$P\left(a_i|a_{1:i-1}\right) = P\left(a_i|a_{i-2:i-1}\right) \tag{1}$$

One can define the probability of a sequence of characters $P(a_{1:N})$ using the trigram model, by first factoring with the chain rule and then using the Markov assumption.

$$P\left(a_{i:N}\right) = \prod_{i=1}^{N} P\left(a_i|a_{1:i-1}\right) = \prod_{i=1}^{N} P\left(a_i|a_{i-2:i-1}\right) \tag{2}$$

In a typical language, a trigram character model with 100 characters, $P\left(a_i|a_{i-2:i-1}\right)$ has 1 million entries and can be accurately estimated by counting character sequences in a body of text with over 10 million characters (Russel and Norvig, 2010).

The process of detecting *n*-grams consists in identifying words that are usually tied together (compound words, proper nouns, etc.) to be able to process them as a single conceptual unit. This is usually performed by estimating the probability of two words that are often together to make up a single term (compound). These techniques attempt to identify compound terms such as *building codes* or *bending moment*. Furthermore, the statistical processes identify a list of empty words in a terms list (prepositions, determiners, pronouns, etc.) that are generally considered to have little semantic value and are eliminated when found in a textual document, leaving them out of the terms index to be analyzed. Deleting all of these terms avoids document noise problems and saves on resources.

Other methods of the NLP approach focus on text classification and categorization. They excerpt information and metadata generating a list of technical critical tasks. These comprise, for example (Nawari, 2015): (1) text analytics, (2) content monetization, (3) the automatic classification of content for regulatory compliance checking, and (4) text mining. Content abstraction and classification is a central area of NLP. These approaches can be founded on the syntactic and/or semantic properties of the chief content. They generally employ the concept of *entities*. An entity can be described as the outcome of the extraction of proper nouns from text, such as people, geographic places, and building elements such as slabs, floors, beams, walls, windows, doors, and so on. Each extracted named entity is sorted, tagged, and assigned a sentiment score, which yields meaning and context to each entity.

The common method utilizing NLP algorithms begins with generating ontology to capture domain knowledge to improve the interpretability and understanding of domain-specific content. The NLP approaches can be the rule-based method or the machine learning–based approach (supervised and unsupervised learning). Rule-based NLP utilizes manually coded rules for meaning extraction and processing. These rules are then iteratively built and refined to enhance the precision of semantics processing (Nawari, 2015). On the other hand, the machine learning–based NLP

approach uses algorithms for training text-processing models based on the content of a given training text. In general, rule-based NLP methods have a tendency to give better text-processing performance (in terms of precision and recall) but do need more human involvement.

Given the complexities of the building codes, most of the past efforts have aimed at investigating the use of semantics for the annotation of building regulations with domain-specific ontologies and then exploited the NLP methods in extracting certain rules. Current research investigations utilize the intermediate processing phase before the final extraction of rules from the source text (e.g., Zhang et al., 2013). This phase generally results in a three-tuple semantic element in the form of <Subject, Attribute, Value>. This sematic element is characterized by (1) an ontology concept, (2) an ontology relation, and (3) a deontic operator indicator.

The method development often has many phases before the final encoding results of the regulatory text. These encompass the preprocessing phase, feature generation, the analysis of the extracted information, extraction rules, and the concluding implementation of the encoding results. In the preprocessing phase, the original content undergoes tokenization, parameterization, sentence splitting, dehyphenation, and morphological analysis. In the second phase, a set of semantic and syntactic structures that define the content are established based on the domain-specific ontology (Zhang et al., 2013). In the parameterization phase, the complexity of the text is minimized once the relevant terms have been identified. This consists of quantifying the document's characteristics (i.e., the terms). It also includes assigning a weight to each one of the relevant terms associated with a document. A term's weight is usually calculated as a function of its appearance frequency in the document, indicating the importance of this term in the document's content description. An example demonstrating these phases is shown in Table 2.2.

Some of the drawbacks of these data-modeling efforts include the prerequisites for human interpretation in order to make them (1) accessible electronically, (2) structured and understandable by machines, (3) represented in a standard format, (4) interoperable, (5) explicit, and (6) transparent (Nawari and Alsaffar, 2015). Moreover, code regulations are complex and often highly subjective in nature; thus, these modeling techniques do need building regulation officers to be involved in the process to ensure the correct interpretations of the encoding.

Nawari (2012) suggested the application of first-order logic (FOL) as a modeling language for encoding building code text. The method is devoted to the knowledge extraction method that seeks a direct transformation of text regulations into formal languages. The approach has the benefit of simplicity but has the limitation of manual effort as well as the necessary technical knowledge to translate the regulatory texts into a set of rules.

FOL is a formal logical language that denotes areas of discourses over which various quantifiers range. Similar to human languages, it describes the universe in terms of

1. Objects: persons, trees, floors, beams, columns, trusses, cables, and so on.
2. Relations: wide, rectangle, oval, greater than, part of, arises from, and so on.
3. Functions: excessive deformation, maximum shear force, and so on.

TABLE 2.2
Encoding of Part of the International Building Code (IBC 2006) Using NLP Method

Information tuple Extracted from Text Sentences		Horn Clause Logic Representation
Subject	**Airspace**	∀ *(a,i,r,s) ((airspace(a) ^ insulation(i) ^ roof_sheathing (r) ^ between (a, i, r) has (a, s) → O (Greater_Than_Or_Equal (1, inch))))*
Subject restriction	Relation (between, insulation, roof_sheathing)	
Compliance checking attribute	NA	
Deontic operator indicator	Obligation	
Quantitative relation	Provide	
Comparative relation	Greater_Than_Or_Equal	
Quantity value	1	
Quantity unit/reference	Inch	
Quantity restriction	NA	

Source: Modified from Zhang et al., 2013

There are two chief mechanisms known in FOL modeling: the syntax, which governs which groups of symbols are allowable terms, and the semantics, which control the meanings and sense behind these terms (Nawari, 2012). Unlike natural languages such as English, the FOL language is completely formal, so that it can be readily determined whether a certain formulation is accurate. There are two main types of permitted terminologies: terms, which indicate objects, and formulas, which signify predicates that can be true or false. The terms and formulas of FOL are strings of symbols that together constitute the script of the language. These symbols of the language alphabet are grouped into logical symbols, which continually have the same connotation, and nonlogical symbols, whose meaning differs by context explanation. A sample illustrating the application of this technique is shown in Table 2.3.

The FLO method would work effectively in the case of condition clauses, as depicted in Table 2.3. Frequently, condition clauses and content clauses are condensed into one sentence in a typical building regulation code. In such cases, the provisional clauses should be checked before checking content clauses. Under these circumstances, one clause is commonly comprised of sentences having a subject, a predicate (verb), and an object, which is referred to as the S+P+O pattern. These declarative sentences can be readily parsed into FLO rule-based schema (Table 2.3).

MARKUP LANGUAGE METHODS

Presently, one can find that most building codes and standard documents are obtainable in some kind of Hypertext Markup Language (HTML) or Portable Document Format (PDF) in addition to hard copy. To enable the knowledge representation for electronic processing, a number of researchers have suggested diverse types

TABLE 2.3
Example of Using FOL Method [1]

Building Code	Provision	Encoding
ASCE 7-10 Standard for minimum design loads for buildings and other structure has the following provision (3.2.1) for computing lateral pressure.	"In the design of structures below grade, provision shall be made for the lateral pressure of adjacent soil. If soil loads are not given in a soil investigation report approved by the authority having jurisdiction, then the soil loads specified in Table 3.2-1 shall be used as the minimum design lateral loads. Due allowance shall be made for possible surcharge from fixed or moving loads. When a portion or the whole of the adjacent soil is below a free-water surface, computations shall be based on the weight of the soil diminished by buoyancy, plus full hydrostatic pressure. The lateral pressure shall be increased if soils with expansion potential are present at the site as determined by a geotechnical investigation."	$\forall x\ Structure(x) \wedge BelowGrade(x) \longrightarrow Required(x, \text{lateral pressure}) \forall x \neg SoilReport(x) \longrightarrow Required(x, \text{Table 3.2.1}) \forall x\ Structure(x) \wedge BelowGrade(x) \wedge BlelowWaterSurface(x) \longrightarrow (Required(x, \text{lateral pressure submerged}) \wedge Required(x, \text{hydrostatic pressure})) \forall x\ Structure(x) \wedge BelowGrade(x) \wedge ExpansiveSoil(x) \longrightarrow Required(x, \text{increase lateral pressure})$

of markup languages to formalize building regulatory rules and guidelines. For instance, Lau and Law (2004) suggested the Extensible Markup Language (XML) as an integrated format to represent regulations, since XML has the capability to model semistructured data such as legal documents. The XML language, in fact, has dual characteristics—namely, as a markup language and as a Web standard—which enables it to form the common ground for action both "at the source" (i.e., regulations) and "downstream" (i.e., relating to the digital-processing capacity).

One of the earliest efforts in using markup language in modeling building regulation was proposed by Omari and Roy (1993). Their markup modeling language is called Dialogue Language (DL) and had been developed to interpret Life Safety Codes (LSC) for Australia in an expert system. It assumes a consistent interpretation to represent the code provisions as well as the interactions between users and the expert systems. DL provides structures that organize the hierarchical patterns of regulations. These structures contain eight primary items (Table 2.4). For example, the *comment* item is used to explain the semantic meaning of the clause text. It generally deals with explanations of the object violation the noncompliance of the building design to the codes. DL offers a simple schema structure that allows one to

TABLE 2.4
Components of the Dialogue Language (DL)

Component Name	Description
Dialog_ID	An identifier that references a dialog.
Parent_ID	An identifier, which points to the dialog from which the current dialog was referenced.
Clause	The actual text of the provision from the code which is embodied within the current dialog.
Condition	The DL interpretation of the conditions which must be satisfied for this dialog to be applicable.
Code_Violaiton	The ID of the object that is shown in the error message.
Action	The DL interpretation of the actions to be carried out if the conditions for application of the dialog are met.
Comment	An explanatory message that describes the reasons for the application of this dialog. This field is primarily dedicated to indicating to the user why a particular dialog has failed, in more simple terms than are normally available from the original code provisions. This text file is attached to the frame identified by the code violation field to indicate the nonconformance of the building model elements of the BCA.
Dependency	A list of property value identifiers that is used during the verification of a dialog. As such, they provide a reference by which the user can determine the exact property that is not in compliance.

Source: Modified from Omari and Roy, 1993.

manage the hierarchical building codes and their interactions with building models to generate meaningful reports for designers. Figure 2.4 depicts an example of the schema of a building code using the DL modeling language (Omari and Roy, 1993).

Further efforts were made by Lau et al. (2004), who proposed the XML as a unified format to represent regulations because of XML's capability of handling semistructured data. To partially automate the translational process, Tau and Law (2004) proposed a shallow parser as the first step to combine different formats of building rules into XML. The representation of the regulatory information, specifically its hierarchical and referential structures, are reconvened in XML schema. For other types of data that are not nonstructured, a feature extraction mechanism has been proposed (Figure 2.5).

Lau and Law (2004) developed parse trees using a context-free grammar and a semantic modeling technique that is capable of tagging regulation provisions with the list of references they encompass. For example, an XML reference tag is shown in Figure 2.6, where Section 4.7.4 of the ADA guidelines cites Section 4.5 once. If properly extracted and linked, these references provide users extra but vital information to better understand building code regulations.

Over the past few years, several Legal XML standards have been suggested to define and represent legal textual information with XML-based rules (Lupo et al., 2007; Bore et al., 2008; Sartor et al., 2011; Palmirani et al., 2011). Other efforts have

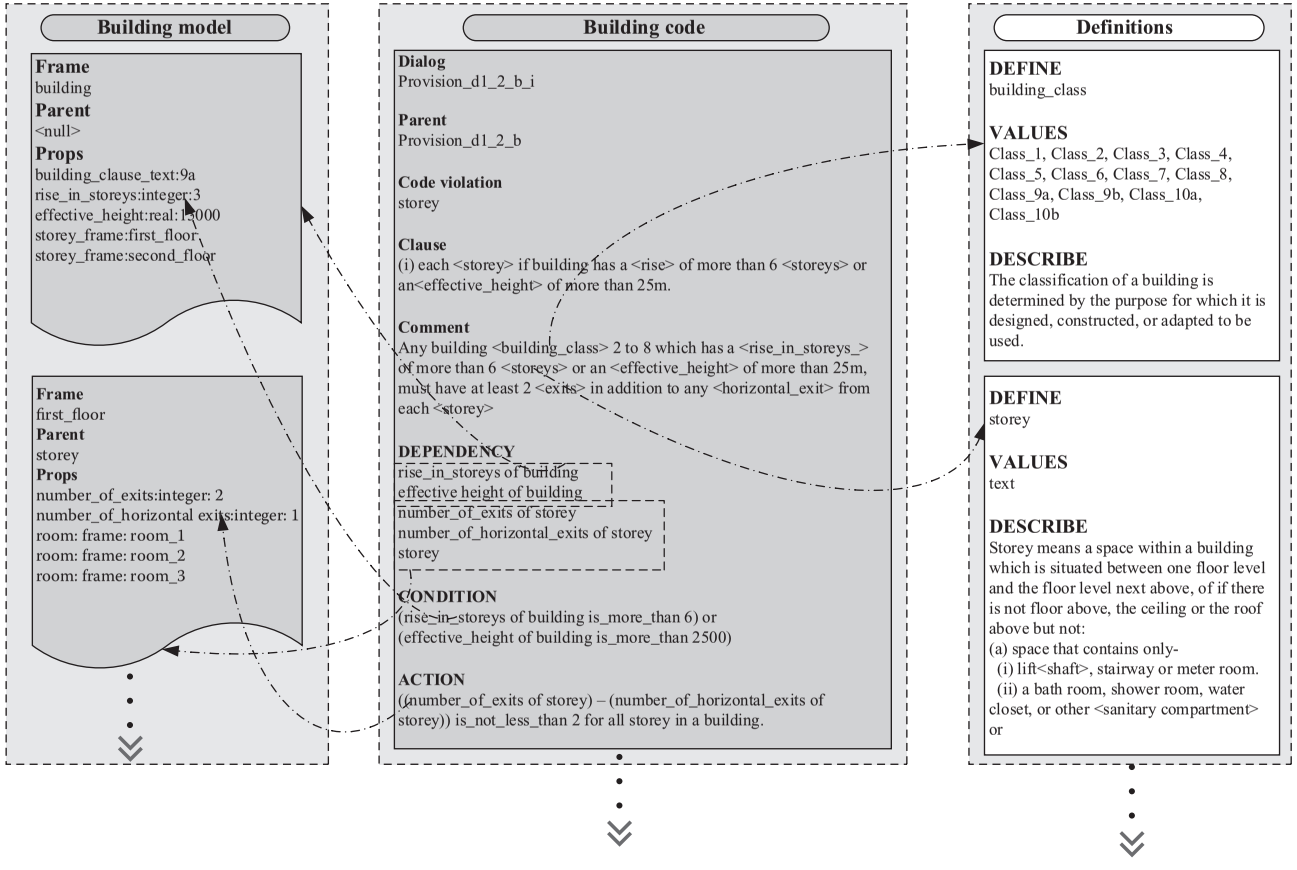


FIGURE 2.4 Illustration of the DL modeling language (modified from Omari and Roy, 1993).

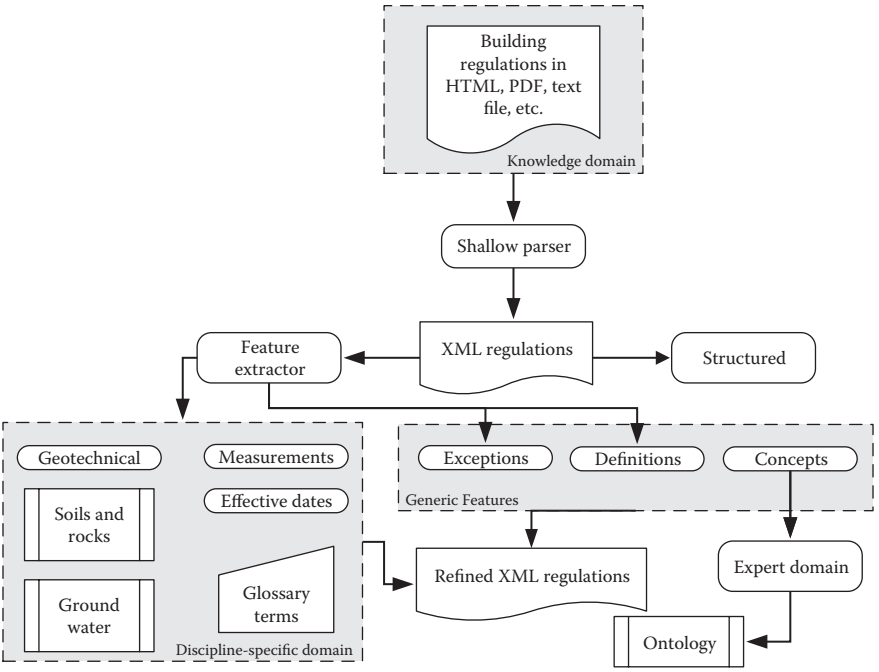


FIGURE 2.5 Representing building regulations in XML (modified from Lau and Law, 2004).

```
<regulation id="adaag" name="ADA Accessibility Guidelines" type="Federal"> ...
  <regElementid="adaag.4" name="Accessible Elements and Spaces..."> ...
    <regElementid="adaag.4.7" name="Curb Ramps"> ...
      <regElementid="adaag.4.7.4" name="Surface">
        <regText> Surfaces of curb ramps shall comply with 4.5. </regText>
        <referenceid="adaag.4.5" num="1" />
      </regElement> ...
    </regElement> ...
  </regElement> ...
</regulation>
```

FIGURE 2.6 Sample of XML format for building code regulations.

focused more on the Semantic Web—specifically, legal ontology research combined with semantic norm extraction based on NLP to model legal concepts and provisions (Mazzei et al., 2009; Francesconi et al., 2010; Palmirani et al., 2010). Most of these efforts have utilized expressive XML annotation combined with Semantic Web technologies to be able to meet the unique characteristics of the legal rules and norms domain. Examples of these efforts have resulted in the RuleML (Boley et al., 2001; Wagner et al., 2004; RuleML, 2017) and LegalRuleML (Palmirani et al., 2011) modeling languages. The goal of LegalRuleML is to represent the specifics of the legal rules and norms with a comprehensive, articulated, and meaningful markup language.

An example illustrating the modeling of legal provision using these approaches is the representation of the US Code covering the infringement of copyright, Title 17, Chapter 6 (Figure 2.7).

In the case of the violation of rule 602b depicted in Figure 2.7a, the penalty statement can be found in Title 17, Chapter 5, 504 (c)(1).

The rule that connects the violation of the rule 602b, and the reparation with the penalty according to rule 504c-1, when the subject has paid the award of statutory damages to the copyright owner, can be modeled as depicted in Figure 2.8b.

§ 602 (b) In a case where the making of the copies or phonorecords would have constituted an infringement of copyright if this title had been applicable, their importation is prohibited.

(a)

```
<Implies id="rule602b">
  <then>
     <prohibition>
        <Atom id="rule602b-prh1-atm1">
           <Rel>importation is prohibited</Rel>
            <Var>z</Var>
        </Atom>
     </prohibition>
  </then>
  <if>
     <And>
        <Atom id="rule602-if-atm1">
           <Rel>copies or phonorecords</Rel>
           <Var>z</Var>
        </Atom>
       <Atom id="impl602-1-if-atm2">
          <Rel>without the authority of the owner of copyright </Rel>
          <Var>x</Var>
       </Atom>
     </And>
  </if>
</Implies>
```

(b)

FIGURE 2.7 (a) Title 17, Chapter 6, Section 602b of the US Code related to the infringement of copyright (US Congress, 2017), (b) An example of LegalRuleML modeling the provision shown in Figure 2.7 (Palmirani et al., 2011).

§ 504. Remedies for infringement: damages and profits (c) statutory damages.

(1) Except as provided by clause (2) of this subsection, the copyright owner may elect, at any time before final judgment is rendered, to recover, instead of actual damages and profits, an award of statutory damages for all infringements involved in the action, with respect to any one work, for which any one infringer is liable individually, or for which any two or more infringers are liable jointly and severally, in a sum of not less than $750 or more than $30,000 as the court considers just. For the purposes of this subsection, all the parts of a compilation or derivative work constitute one work.

(a)

```
<Atom id="atm504">
   <penalty id="atm504-pnl1">
      <obligation id="obl2" subject="z" beneficiary="y" timesBlock="#t2">
         <Atom id="atm504-pnl1-atm1">
            <Rel>award of statutory damages to</Rel>
            <Var>z</Var>
            <Dat>min $750 </Data>
            <Data>max $30,000 </Data>
         </Atom>
      </obligation>
   </penalty>
</Atom>
```

(b)

FIGURE 2.8 (a) Title 17, Chapter 5, Section 504 of the US Code related to the infringement of the copyright (US Congress, 2017), (b) An example of LegalRuleML modeling the provision shown in Figure 2.9 (Palmirani et al., 2011).

FORMAL LANGUAGES

Further examples of research work on the conversion of regulation texts into formal languages is the automated markup of Italian legislative texts in XML (Biagioli et al., 2004). The automatic transformation of legal documents is attained with a content-based clustering of regulations and the labeling of Italian law texts. The significance of various expressions that can recognize a regulation text is highlighted, such as terms appearing in the head notes, heading section, case name, and so on. These terms are then taken into consideration when dealing with information extraction methods for legal case retrieval and are formalized as markup values in the XML representation of a legal text.

Hjelseth et al. (2010, 2011) proposed a framework for a formal language that can represent building rules from a direct semantic understanding of the text by utilizing four semantic markup operators: *Requirement*, *Applies*, *Select*, and *Exception*. The suggested semantic-based method is able to deal with a wide range of purposes such as validating and guiding systems and adaptive and content-based model checking. This system still needs a manual identification of the formal rules by an AEC professional familiar with domain skills and the implementation of applicable software code based on common predefined measures. The suggested markup language model is founded on a few operators—namely, *Select* (*S*), *Applies* (*A*), *Requirement* (*R*), and *Exception* (*E*). Applied to provisional text, the user may highlight any phrase that means "more scope" as *S*, "less scope" as *A*, "shall," "must," and so on as *R* (plus alternative requirements),

and "unless" and so on as an *E* (as well as compound exceptions). The constructs *R*, *A*, *S*, and *E* are commonly qualified to have a topic, a property, a comparator, and a target value. The topic and property are preferably derived from a restricted dictionary comprised of objects described within the code provisions and general practice. The resulting output may be numeric, whereupon the comparators will cover "greater," "lesser," "equal," and their opposites. When the value is expressive, then only the "equal" or "not equal" comparators are pertinent. If the value indicates a set of objects, then the comparator can be any of the set comparison operators such as "includes" or "excludes" (Hjelseth et al., 2011).

Additionally, the suggested method is proposed to be connected to the IFC constraint subschema. This subschema can be instantiated as an object qualified as a specification constraint. An example demonstrating the implementation of this method in the case of the International Building Code ICC IECC 2006 moisture control specification is depicted in Table 2.5.

This suggested approach has the potential to be applied successfully in the case of fully normative regulations documents that are anticipated to result in pass or fail outcomes. However, the approach has limitations to deal with other types of code regulations text—specifically, the deep hierarchies and substantial cross-referencing among provisions in typical code regulations texts.

SEMANTIC WEB APPROACH

The internet (World Wide Web) was initially intended as a content service whose documents were to be exhibited by Web browsers and was only meaningful to people rather than to machines (Cardoso, 2007). The data and meaning expressed on Web pages are challenging for a computer to extract and understand, thus preventing further automated information processing (Berners-Lee, 2001). The Semantic Web approach is suggested to solve this issue by giving order and meaning to the unstructured information available on the internet by adding contextual information (i.e., metadata) to existing information. This semantic approach to Web markup is a key concept

TABLE 2.5
Markup Specification ICC IECC 2006 502.5 Moisture Control [16].

Code Clause Source	**ICC IECC 2006 502.5 Moisture control**
Rule Description	All {*A/*} framed {\A}{*S/*} walls, floors {*S*}, and {S/} ceilings {\S} {E/} not ventilated {*E*} to allow moisture to escape shall be provided with an {*R/*} approved vapor retarder {*R*} having {*R/*} a permanence rating of 1 perm {*R*} (5.7 × 10 –11 kg/Pa s m^2) or less, when tested in accordance with the desiccant method using Procedure A of ASTM E 96. The vapor retarder shall be {*R/*} installed on the warm-in-winter side {*R*} of the insulation. Exceptions: {*E/*} Buildings located in Climate Zones 1 through 3 {*E*}. In constructions where {*E/*} moisture {*E*} or its {*E/*} freezing {*E*} will not damage the materials. Where there are other approved means to avoid {*E/*} condensation {*E*} in unventilated framed wall, floor, roof, and ceiling cavities.

underlying efficient Web coding, information processing, universal usability, search engine visibility, and maximum display flexibility. Web content can be accessed using Web browsers, mobile computing devices of all kinds, and screen readers. The context (or semantic) information stored in the tags will be available to software applications reading the content and will enable them to identify the difference between similar data. Thus, the Semantic Web is built on XML's ability to define customized tagging schemes and the Resource Description Framework (RDF)'s flexible approach to representing data. RDF is defined as a data model for objects (*resources*) and relations between them, which provides simple semantics for the data. It has also a graphical representation. An RDF graph is constructed by using a logical AND operator on a series of logical statements containing concepts or objects in the world and their relations. These statements are often referred to as *RDF triples*, comprising a subject, a predicate, and an object (Figure 2.9). In addition, each concept and relation has a Unique Resource Identifier (URI) assigned to it, thereby making the RDF graph explicitly labeled. Every concept described in an RDF graph, whether this is an object, subject, or predicate, is uniquely defined through this URI. Furthermore, the resulting RDF graph can be converted into a textual representation that follows a specific syntax. Syntaxes used for RDF graphs are RDF/XML, N-Triples (Grant et al., 2004), Turtle (Beckette et al., 2011), SPARQL (Prud'hommeaux et al., 2008), and Notation-3 (N3 logic notation) (Berners-Lee et al., 2011).

Another important language associated with the Sematic Web approach is the Web Ontology Language (OWL), which is aimed for use by applications that are required to process the content of information instead of just displaying information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. An ontology is represented as a set of concepts within a domain and the description of the relationships between the concepts (Akinci, 2008). Ontologies work like an encyclopedia for computers, connecting heterogeneous metadata information, explaining all the definitions and listing all the synonyms, therefore providing a global structure to the data on the Web and allowing the data to be understood and shared across multiple applications and communities (Zhan and Issa, 2011).

Based on the concepts and modeling languages defined previously, the Semantic Web Rule Language (SWRL; www.w3.org/Submission/SWRL) is introduced. SWRL is defined as a language that combines other sublanguages such as OWL Web Ontology and the Rule Markup Language (RuleML; www.ruleml.org). RuleML is a markup language developed to express a family of Web rules in XML for deduction, rewriting, and reaction, as well as further inferential, transformational, and behavioral tasks. It is defined by the Rule Markup Initiative (www.ruleml.org), an open

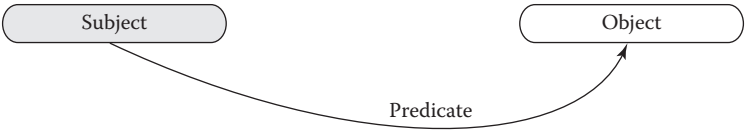


FIGURE 2.9 The "triple" form of an RDF statement: subject–predicate–object.

network of individuals and groups from both industry and academia that was formed to develop a canonical Web language for rules using XML markup and transformations to and from other rule standards/systems. It is based on a modular, hierarchical specification for different types of rules comprising facts, queries, derivation rules, integrity constraints (consistency maintenance rules), and production rules, as well as the ability of transformations to and from other rule standards/systems.

One of the earliest efforts to implement the Semantic Web technologies to building regulation compliance checking is demonstrated by the work of Pauwels et al. (2011). Their work aimed at accommodating acoustic regulation compliance checking for BIM models. Their concept is based on the notion of the *semantic network*. This semantic network describes concepts through a directed, labeled graph (Figure 2.10). Each node in this graph represents a concept or object in the world and each arc in this graph represents the logical relation between two of these concepts or objects (Pauwels et al., 2011). The graph depicted in Figure 2.10 illustrates an example of a combination of logic-based declarative sentences, each sentence consisting of two nodes and a relational arc. The semantics of a concept is described through the graph associated with that concept. In addition to the ability to describe facts in a semantic network, the Semantic Web domain also has the capability to define rules using a rule language that is based on logic theory. Since both the information content description language and the rule languages are systematically based on logic theory, they can provide the possibility to implement a logic-based rule-checking system declaratively. Using semantic rule language should enable a declarative approach explicitly based on logic for the implementation of a rule-checking environment (Pauwels et al., 2011).

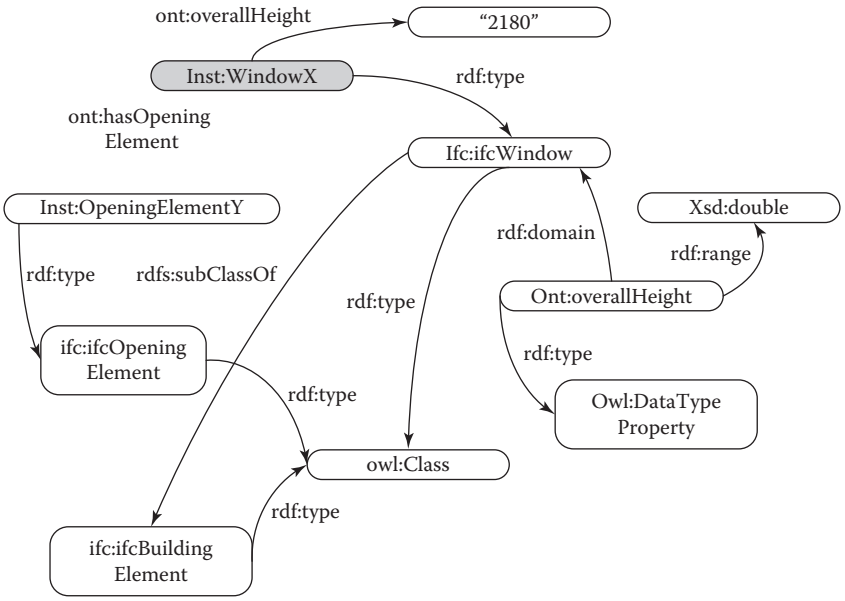


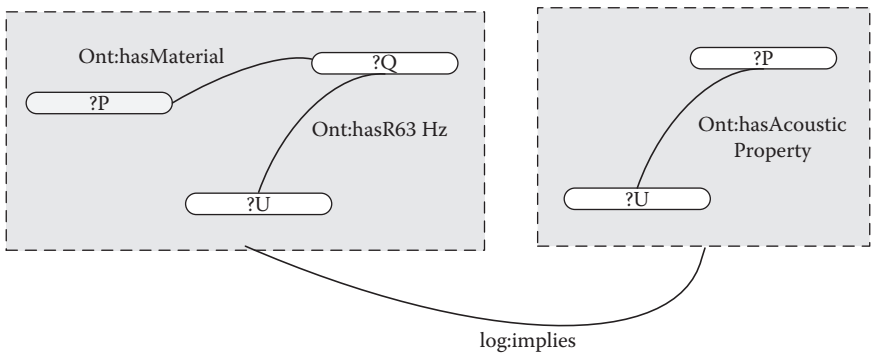
FIGURE 2.10 Semantic network using a directed, labeled graph describing a window in an opening element in a building (modified from Pauwels et al., 2011).

```
inst:WindowX              rdf:type          ifc:IfcWindow.
ifc:IfcWindow             rdf:type          owl:Class.
ifc:IfcWindow r           dfs:subClassOf    ifc:IfcBuildingElement
ifc:IfcBuildingElement    rdf:type          owl:Class
ont:overallHeight         rdf:type          owl:DatatypeProperty
ont:overallHeight         rdfs:domain       ifc:IfcWindow.
ont:overallHeight         rdfs:range        xsd:double
```

FIGURE 2.11 An OWL ontology specifying an IfcWindow class (modified from Pauwels et al., 2011)

The network can be given an improved semantic structure using RDF vocabularies or ontologies. The most basic elements to describe such ontologies are available in the RDF-S vocabulary (Brickley et al., 2014). RDF-S, for instance, permits the description of classes, subclasses, comments, data types, and so on. This description is performed through RDF statements that display concepts from the RDF-S vocabulary. For example, for the network shown in Figure 2.10, the ontology specified for the *ifc:IfcWindow* class is a subclass of the *ifc:IfcBuildingElement* class.

Moreover, Pauwels et al. (2011) employed N3-Logic to describe custom rules for the acoustic performance regulations of the European Code EN1235. A rule in N3-Logic generally encompasses hypothetical "formulae." Every hypothetical formula thereby uses curly brackets in its syntax to describe a subgraph—that is, a logical conjunction of the statements between the curly brackets (Berners-Lee et al., 2008) (Figure 2.12).




```
{
        ?P   Ont:hasMaterial     ?Q
        ?Q   Ont:hasR63 Hz       ?U
}
log: implies
{
        ?P   Ont:hasAcousticProperty    ?U
}
```

FIGURE 2.12 Description of the N3-Logic rule in its normative N3 representation (modified from Pauwels et al., 2011).

Similar to the syntax of an RDF triple statement, a rule may describe that IF one hypothetical formula is true, THEN the other hypothetical formula is also true. Furthermore, an N3-Logic rule references the concepts required in the rule using the URIs of these concepts, optimally abbreviated with prefixes. Figure 2.13 depicts an example of a rule in N3-Logic. This rule specifies that IF an entity can be found that has an *ont:hasMaterial* property AND that property has an *ont:hasR63Hz* property, THEN that entity has an *ont:hasAcousticProperty* property with the same URI as (i.e., identical to) the *ont:hasR63Hz* property.

Figure 2.13 shows a rule from the European Code EN12354-3:2000 rule set, which focuses on acoustic rules, specifying how the *EN12354: directSoundPowerRatio_4000Hz* property can be derived from a building model. The EN12354 prefix refers to the base URI of the rule ontology for this standard. The segment of the European standard is manually interpreted and translated into a rule ontology in OWL describing the concepts used within the standard, a rule set in N3-Logic describing the logic in the standard, and a conversion rule set describing the references required for the building information graph. The example is limited and simply aimed at detailing a proof of concept and is not an entirely executed and extensively functional application.

PROPOSED METHODS

This section aims at introducing practical approaches for computerizing building codes. The detailed description of this suggested approach is the subject of Chapter 5.

One of the main objectives of these practical techniques of transforming building regulations is to offer a computable model with clear syntax and semantics that can be utilized to represent and reason about building rules and provisions. Additionally, the model should be compatible with the general requirements of digital content providers. An object-based schema of building codes should express the least amount

```
{
        ?BE     rdf:type EN12354:BoundaryElement.
        ?BE     EN12354:elementSurfaceArea ?a.
        ?BE     EN12354:partOfRoomBoundary ?RB.
        ?RB     rdf:type EN12354:RoomBoundary.
        ?RB     EN12354:facadeSurfaceArea ?atot.
        ?BE     EN12354:soundReductionIndex_4000Hz ?R_4000.
        ?a      math:notLessThan 1.
        (?a ?atot) math:quotient ?value_i.
        (?R_4000 -10) math:quotient ?value_j.
        (10 ?value_j) math:exponentiation ?value_k.
        (?value_i ?value_k) math:product ?value_l
}
log:implies
{
        ?BE     EN12354:directSoundPowerRatio_4000Hz ?value_l
}
```

FIGURE 2.13 A segment of the European Code EN12354-3:2000 being modeled using the Semantic Web approach (modified from Pauwels et al., 2011).

of data needed to allow the automatic auditing of conformance, and consequently, services attain integrated practice goals when using the BIM workflow in the AEC industry.

In this proposed method, the provisions of any given building code are classified into four main categories, as shown in Figure 2.14. Specifically, these are conditional clauses, content clauses, ambiguous clauses, and dependent clauses.

Conditional clauses are the sections that are valid to interpret directly from the textual document into a set of formal rules. Examples of these are very common, and typical features include rules with specific values such as those given in Tables 2.3 and 2.4.

Contents clauses are the sections that cannot be transformed into TRUE or FALSE expressions. These clauses are normally utilized for descriptions and definitions, such as the definition of firewall, fire rate, smoke evacuation, high-rise building, and so on.

Ambiguous clauses are the subjective provisions. They normally include words such as *approximately*, *about*, *relatively*, *close to*, *far from*, *maybe*, and so on. An example of these is the footnote of the design lateral soil pressure for the clause given in Table 2.2.

> For *relatively* rigid walls, as when braced by floors, the design lateral soil load shall be increased for sand and gravel type soils to 60 psf (9.43 kN/m2) per foot (meter) of depth. Basement walls extending not more than 8 ft (2.44 m) below grade and supporting light floor systems are not considered as being *relatively* rigid walls. (Nawari and Alsaffar, 2015).

Dependent clauses indicate that one section is reliant on one or more other clauses. This means some provisions are only suitable for a particular condition when other

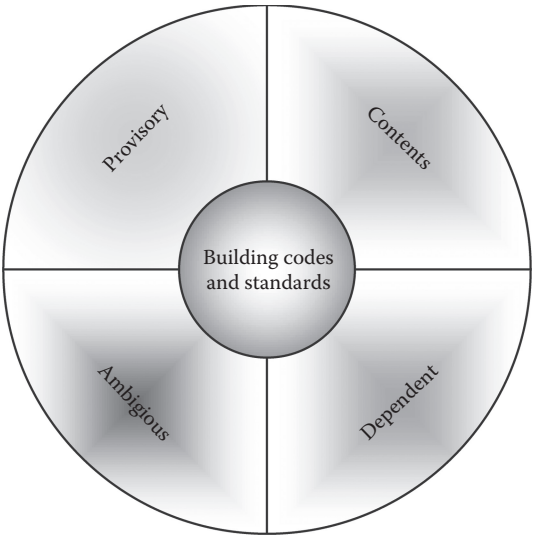


FIGURE 2.14 Major classification categories of building regulations clauses.

provisions are fully satisfied. These are often difficult to transmute into sets of formal rules and often require manual checking for conformance.

The method proposes that knowledge representations of the building codes and standards can be established by considering a number of development levels (Nawari, 2015):

1. High-order level I: Provisions and clauses of any given building regulations are classified into four main categories, as illustrated in Figure 2.14.
2. High-order level II: Requires the development of a Model View Definition (MVD), which will lead to IFC schema.
3. Higher-order level III: Requires the feature extraction of all specific data objective concepts leading to full encoding.
4. Lower-order level: Necessitates the feature extraction of all ambiguous information and uncertain data, then the employment of partial encoding using fuzzy logic or full manual checking.

The main characteristics of the proposed approach for automated and semiautomated building regulation checking include

1. *Minimality*: This means that the modeling language used for representing the building code provisions provides only a small set of required language constructs—that is, maintaining the same meaning of the language constructs recognized by the industry.
2. *Referential transparency*: This refers to the fact that the same modeling language structure constantly expresses the same semantics irrespective of the context in which it is used.
3. *Orthogonality*: The suggested framework addresses how a limited number of constructs can be combined in a comparatively small number of ways to get the anticipated outcomes. That means this framework is more orthogonal than other existing approaches. This makes it easier to learn, read, and implement.
4. *Pattern-based approach*: The framework emphasizes a modeling language that brings together a number of patterns in organizing the structural parts and the grammar in one cohesive model. It aims to reproduce all the knowledge needed to create quality items in automating building regulations. It gives the power to create an automated or semiautomated compliance-checking system for building regulations that is easy to author, process, use, and maintain.

Figure 2.15 demonstrates the suggested approach model for AEC regulations and standards. The method depends chiefly on the XML standard as it provides many benefits (Nawari, 2012):

- Organizing the structural parts, the grammar, and the constraints for representing the building codes using XML structural patterns.

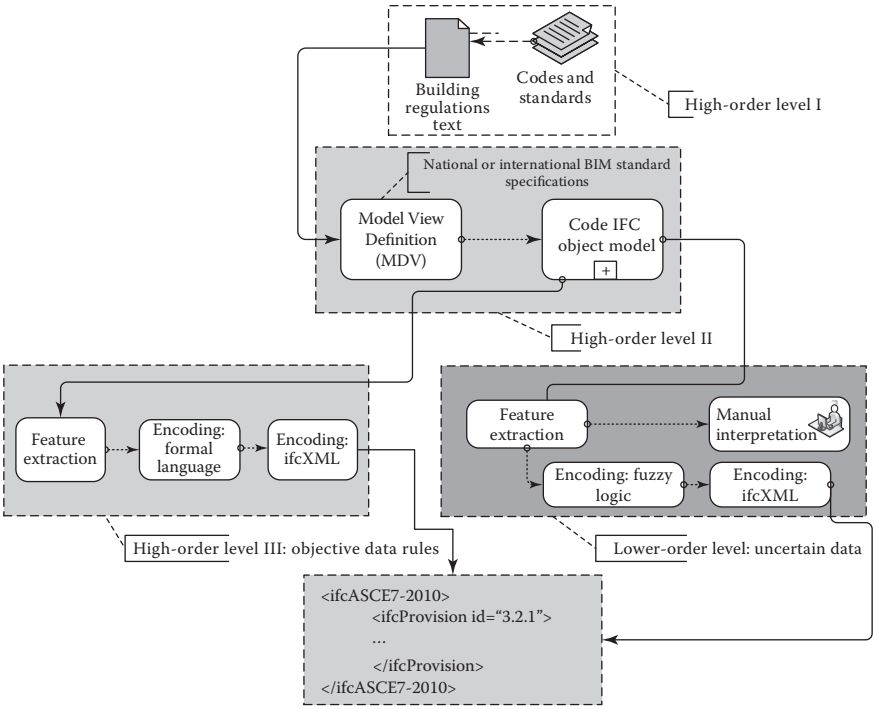


FIGURE 2.15 Suggested approach for computable building code model.

- The ability to capture the common meaning of AEC regulations terms as understood in the industry, formalize the connections among the various concepts and their representation in the IFC data model, and provide an XML-based abstract syntax.
- An in-memory XML programming interface that facilitates communication with XML from within the various software programming languages.
- The powerful extensibility of the query architecture can provide implementations that work over both XML and traditional SQL data stores.
- The query operators over XML use an efficient, easy-to-use, in-memory XML facility to provide XPath/XQuery functionality in the host programming language. This integration provides strong typing over relational data objects while retaining the expressive power of the relational database model and the performance of query evaluation directly in the underlying data store.
- All standard query operators can be replaced with user-defined implementations that provide additional services such as remote evaluation, query translation, and optimization.

The feature extraction is normally produced with the support of the ontological representation of all pieces of information taking part in the building code text along with the MVD and the resulting IFC subschema. According to the proposed

```
<ifcASCE7-2010>
    <ifcProvision id="3.2.1">
    ...
    </ifcProvision>
</ifcASCE7-2010>
```

FIGURE 2.16 Part of the ifcXML schema for ASCE 7-2010.

encoding approach, features from building rules and regulations can eventually be transmuted according to the ifcXML tag-based context. Conceivable values for these tags are the provision numbers, rules, concepts, and properties of the features being extracted (Figure 2.16). The ifcXML schema is in effect a standardized computable rule format, executed by any software application capable of reading an XML data file.

The suggested method seeks to develop a formal procedure for the encoding of building regulations in a way that is close to human language and readily verifiable, with numerous syntactic and semantic features that have the ability to deal with objective as well as subjective building code data. The transformation of the uncertain data into rules can be handled partially by utilizing fuzzy logic. Fuzzy logic has been applied successfully in many fields to support the processing of information that is uncertain or may not be entirely defined. Finally, there is always the likelihood that certain parts of building codes and regulations do still need manual checks and reviews for compliance.

REFERENCES

Abrantes, V. (2010). Automated code-checking as a driver of BIM adoption. In *World Congress on Housing*. Santander, Spain.

Babič, N.Č., Podbreznik, P., and Rebolj, D. (2010). Integrating resource production and construction using BIM. *Automation in Construction*, 19(5), 539–543.

Beckett, D. and Berners-Lee, T. (2011). Turtle: Terse RDF triple language. W3C Team Submission, March 28, 2011. http://www.w3.org/TeamSubmission/turtle (accessed June 7, 2017).

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Science American*, 284(5), 34.

Berners-Lee, T. and Connoly, D. (2011). Notation 3 (N3): A readable RDF syntax. W3C Team Submission, March 28, 2011. http://www.w3.org/TeamSubmission/n3 (accessed June 7, 2017).

Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., and Hendler J. (2008). N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, 8(3) (2008), 249–269.

Biagioli, C., Francesconi, E., Spinosa, P., and Taddei, M. (2004). XML documents within a legal domain: Standards and tools for the Italian legislative environment. In *Proceedings of IAPR Workshop on Document Analysis Systems*, pp. 413–424, 2004.

Boley, H., Tabet, S., and Wagner, G. (2001). Design rationale for RuleML: A markup language for Semantic Web rules. In I.F. Cruz, S. Decker, J. Euzenat, and D.L. McGuinness (eds), *Proceedings of SWWS 2001, the First Semantic Web Working Symposium*, pp. 381–401 (2001).

Boer, A., Radboud, W., and Vitali, F. (2008). MetaLex XML and the Legal Knowledge Interchange Format. In P. Casanovas, G. Sartor, N. Casellas, and R. Rubino (eds.), *Computable Models of the Law*. LNCS (LNAI), vol. 4884, pp. 21–41. Heidelberg: Springer (2008).

Brickley, D. and Guha, R.V. (2014). RDF Vocabulary Description Language 1.0: RDF Schema: W3C Recommendation, February 25, 2014. http://www.w3.org/TR/rdf-schema (accessed June 7, 2017).

Building Professionals Board (2011). Guide to the building approvals process. Building Professional Board.

Building Professionals Board (2012). Case study: Structural engineering design and construction defects (class 2–9 buildings). Sydney, Australia: Building Professional Board.

buildingSMART (2010). Model: Industry Foundation Classes (IFC). BuildingSMART International, http://www.buildingsmart-tech.org/specifications/ifc-releases (accessed 11/20/2010).

buildingSMART Australasia (2012). National Building Information Modelling Initiative, Vol. 1, Strategy: A strategy for the focussed adoption of building information modelling and related digital technologies and processes for the Australian built environment sector. Report to the Department of Industry, Innovation, Science, Research and Tertiary Education, Sydney, Australia.

Cardoso, J. (2007). *Semantic Web Services: Theory, Tools and Applications*. IG IGlobal, Hershey, PA.

Ding, L. et al. (2004). Automated code checking. CRC CI International Conference, Gold Coast, Australia, 2004.

Ding, L. et al. (2006). Automating code checking for building designs: DesignCheck. *Clients Driving Innovation: Moving Ideas into Practice*, pp. 1–16.

Eastman, C. et al. (2009). Automatic rule-based checking of building designs, *Automation in Construction*, 18(8), 1011–1033.

Francesconi, E., Montemagni, S., Peters, W., and Tiscornia, D. (2010). *Semantic Processing of Legal Texts: Where the Language of Law Meets the Law of Language*. Heidelberg: Springer (2010).

Grant, J. and Beckett, D. (2004). RDF test cases: W3C recommendation, February 10 2004. http://www.w3.org/TR/rdf-testcases (accessed June 7, 2017).

Greenwood, D. et al. (2010). Automated compliance checking using building information models. The Construction, Building and Real Estate Research Conference of the Royal Institution of Chartered Surveyors, Paris, France, September 2–3.

Grilo, A. and Jardim-Goncalves, R. (2010). Value proposition on interoperability of BIM and collaborative working environments. *Automation in Construction*, 19(5), 522–530.

Gu, N., Williams, A., and Gül, L.F. (2007). Designing and learning in 3D virtual worlds. In *Proceedings of the Iadis International Conference on Cognition and Exploratory Learning in the Digital Age* (Celda 2007), Algarve, Portugal, pp. 227–234.

Hjelseth, E. and Nisbet, N. (2010). Overview of concepts for model checking. CIB W78 2010, 27th International Conference–Applications of IT in the AEC Industry. Cairo, 16–18 November 2010.

Hjelseth, E. and Nisbet, N. (2011). Capturing normative constraints by use of the semantic mark-up (RASE) methodology. In *Proceedings of CIB W78-W102 Conference*, Sophia Antipolis, France, pp. 1–10.

International Code Council (formerly BOCA, ICBO and SBCCI). (2006). International building code, 4051, pp. 60478–65795.

Jeong, J. and Lee, G. (2010). Requirements for automated code checking for fire resistance and egress rule using BIM. In *ICCEMICCPM* 2009, pp. 316–322.

Khemlani, L. (2011). CORENET e-PlanCheck: Singapore's automated code checking system. AECbytes. http://www.aecbytes.com/feature/2005/CORENETePlanCheck.html (accessed 7/16/2011).

Lau, G.T. and Law, K. (2004). An information infrastructure for comparing accessibility regulations and related information from multiple sources. In *Proceedings of 10th International Conference on Computing in Civil and Building Engineering*, Weimar, Germany, June 2–4, 2004.

Leibniz, G. W. (1666). *Dissertatio de arte combinatoria*. Leibnizens mathematische Schriften 5, Georg Olms, Hildesheim.

Lupo, C., Vitali, F., Francesconi, E., Palmirani, M., Winkels, R., de Maat, E., Boer, A., and Mascellani, P. (2007). General XML format(s) for legal sources: Estrella European Project IST-2004-027655. Deliverable 3.1, Faculty of Law. University of Amsterdam, the Netherlands (2007).

Martins, J.P. and Monteiro, A. (2013). LicA: A BIM based automated code-checking application for water distribution systems. *Automation in Construction*, 29, 12–23.

Mazzei, A., Radicioni, D.P., and Brighi, R. (2009). NLP-based extraction of modificatory provisions semantics. In *ICAIL 2009*, pp. 50–57. New York: ACM Press (2009).

Nawari, N. (2012). Automating codes conformance. *Journal of Architectural Engineering*, 18(4), 315–323. Retrieved from http://ascelibrary.org/doi/abs/10.1061/(ASCE)AE.1943-5568.0000049.

Nawari, N.O. and Alsaffar, A. (2015). Understanding computable building codes. *Journal of Civil Engineering and Architecture*, 3(6), 163–172.

Noh, S.-Y. and Gadia, S.K. (2006). A comparison of two approaches to utilizing XML in parametric databases for temporal data. *Information and Software Technology*, 48(9), pp. 807–819.

Omari, A., and Roy, G. G. (1993). *A representational scheme for design code information in an expert systems approach to building design*. Computing Systems in Engineering Vol. 4, Nos. 2–3, 253–269, 1993.

Palmirani, M. and Brighi, R. (2010). Model regularity of legal language in active modifications. In P. Casanovas, U. Pagallo, G. Sartor, and G. Ajani (eds.), *AICOL-II/JURIX 2009*, pp. 54–73, LNCS, vol. 6237. Heidelberg: Springer (2010).

Palmirani, M., Governatori, G., Rotolo, A., Tabet, S., Boley, H., and Paschke, A. (2011). LegalRuleML: XML-based rules and norms. *Proceedings of 5th International Symposium, RuleML 2011*, Fort Lauderdale, FL, November 3–5, 2011 [W3C98], pp. 289–312.

Pauwels, P., Van Deursen, D., Verstraeten, R., De Roo, J., De Meyer, R., Van de Walle, R., and Van Campenhout, J. (2011). A semantic rule checking environment for building performance checking. *Automation in Construction*, 20(5), 506–518.

Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C Recommendation, January 15, 2008. http://www.w3.org/TR/rdf-sparql-query (accessed June 7, 2017).

RuleML. The Rule Markup Initiative. http://www.ruleml.org (accessed June 3, 2017).

Russel, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*, Third Edition. Prentice Hall, Upper Saddle River, NJ.

Sartor, G., Palmirani, M., Francesconi, E., and Biasiotti, M. (2011). Legislative XML on Semantic Web. Heidelberg: Springer (2011).

US Congress (2017). Public and private laws. https://www.gpo.gov/fdsys/browse/collection.action?collectionCode=PLAW&browsePath=115%2FPUBLIC%2F%5B*-*%5D&isCollapsed=false&leafLevelBrowse=false&isDocumentResults=true&ycord=0.

Wagner, G., Antoniou, G., Tabet, S., and Boley, H. (2004). The abstract syntax of RuleML: Towards a general web rule language framework. In *Proceedings of Web Intelligence 2004*, pp. 628–631. IEEE Computer Society (2004).

Whitehead, A. N., (1937). Essays in Science and Philosophy. Philosophical Library, New York, NY.

Zhang, J. and El-Gohary, N. (2013). Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking. *Journal of Computing in Civil Engineering, ASCE*, 1943–5487, 2013.

Zhang, L. and Issa, R.R.A. (2011). IFC-based construction industry ontology and Semantic Web Services framework. In *Proceedings of the International Workshop on Computing in Civil Engineering 2011*, June 19–22, 2011, Miami, FL, pp. 657–664, ASCE.

3 Building Information Modeling and Code Checking

INTRODUCTION

Building information modeling (BIM) is a simulation prototyping technology and workflow that is being recognized as a revolutionary development currently reshaping the architecture, engineering, and construction (AEC) industry (Nawari and Kuenstle, 2015). The technology component of BIM aids industry stakeholders in visualizing and examining what is to be fabricated or constructed in a virtual environment and detects any potential design, construction, functioning, or maintenance conflicts and errors. The workflow element enables collaboration and promotes integration across AEC disciplines. As a result, BIM is effectively changing the role of computation in building design and construction by generating a database of building objects used for all phases of a project, from conceptual design to construction and beyond. Researchers have cited that the collaborative use of BIM in the AEC industry results in completing projects with more resources, higher quality, and greater customer satisfaction (Nawari and Kuenstle, 2015).

According to the US National BIM Standard (NBIMS-US), BIM is defined as follows:

> BIM is a digital representation of physical and functional characteristics of a facility. A building information model is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition. (BuildingSMART, 2015)

Hence, BIM's overall scope is far-reaching, covering many aspects of the intelligent digital representation of building objects. BIM-authoring tools are platforms that produce detailed information and digital representations and intelligent virtual models. They also develop and aggregate information that was created initially as separate tasks with noncomputable information systems such as traditional computer-aided drafting (CAD) or paper-centric processes. Moreover, BIM is a collaborative process that covers business drivers, automated process capabilities, and open information standards used for information sustainability and fidelity. In summary, BIM offers facility life cycle management for all relevant data exchanges, workflows, and procedures that are based on reproducible, verifiable, transparent, and sustainable information used throughout the building life cycle (Figure 3.1).

In AEC practice, the BIM workflow has a more widespread application today and will continue to be the case in the future. This is due to the fact that BIM refers

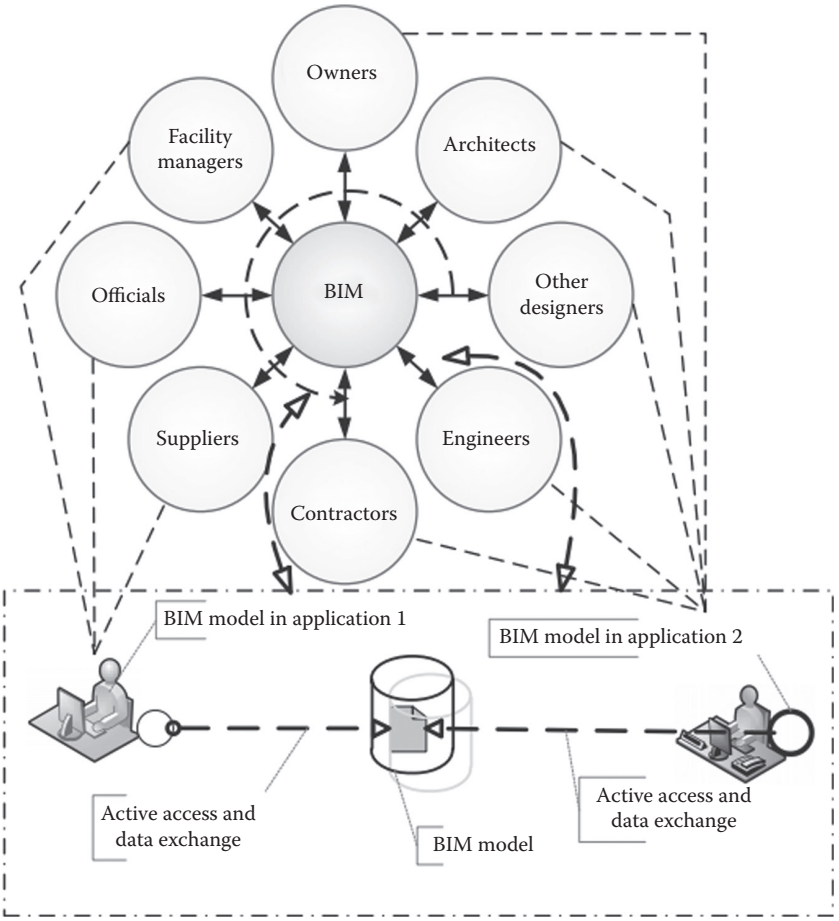


FIGURE 3.1 BIM concept and process.

not only to the physical and functional characteristics of a facility but also how to manage all relevant information throughout its life cycle. This contains, for instance, links to resources, processes, written documentation, and additional information that contribute to the success of a project or that can be combined with other digital tools. The principle of the continuous, centralized, and object-based management and coordination of a project information system is an essential change from the traditional isolated drafting files approach of CAD systems, for example. Clearly, BIM has had a significant impact on the AEC industry. It benefits clients, designers, engineers, educators, craftsmen, construction companies, and facility managers.

BIM's ability to attach an extensible set of *properties* to building objects means that it is potentially a far more convincing instrument in communicating building designs in terms of getting approval from the rule-checking authorities. In other words, BIM enables a new environment where codes and building regulations can mostly be verified automatically through appropriate software applications.

BIM models are in general semantically rich. However, the full benefits of BIM will materialize only through the sharing of information contained within the models across organizations, trades, and information management systems and software. The Industry Foundation Classes (IFC) data model, developed and maintained by buildingSMART International, is the key to facilitating this interoperability in a cost-effective way and without relying on any particular product or software-specific file formats. IFC adds an open-standard language for exchanges of information between different BIM-authoring tools while maintaining the meaning of different objects of information in the exchange. The IFC data model is registered as an international standard by the International Organization for Standardization (ISO) as ISO/IS 16739. Thus, IFC plays an important role in code compliance checking in terms of standardization, unambiguity, consistency, and the comprehensiveness of descriptions of building designs.

RELATIONSHIP BETWEEN BIM AND AUTOMATED CODE CHECKING

In the BIM workflow, all design activities are model driven. Thus, designer, contractor, client, and vendor systems cannot interoperate unless their data formats are identical. This problem was recognized as soon as the first database systems were interconnected in the 1970s. The primary requirement in the application of automated code compliance–auditing systems is that object-based building models must have the necessary information to allow for the automated checking of building regulations compliance. BIM objects being created normally have various parametric data and properties. For example, an object that represents a structural column possesses types and properties such as steel, wood, or concrete, sizes, and so on. Hence, the requirements of a building model adequate for automated code conformance verification are more stringent than normal 3D or 2D drafting requirements. Architects, engineers, and contractors creating building models that will be utilized for code conformance verification must prepare them so that the models

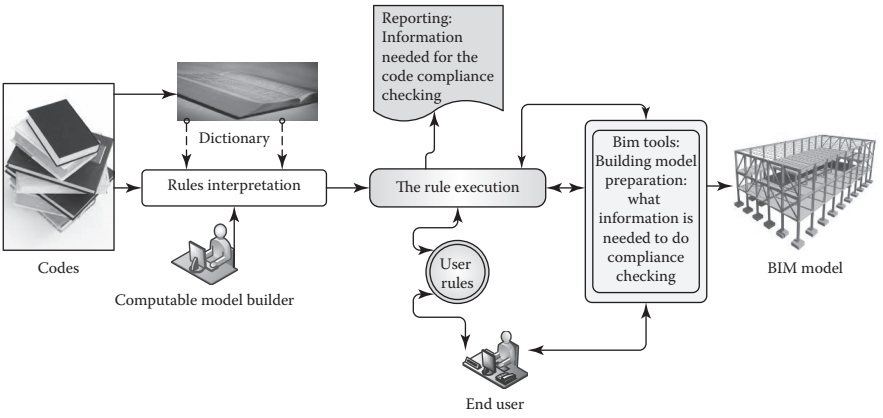


FIGURE 3.2 Relationship between BIM model and automated code compliance checking.

deliver the information needed in a well-defined, agreed-upon schema and level of detail (Figure 3.2).

For example, the US General Service Administration (GSA) BIM guide for energy performance (GSA, 2009) provides initial illustrations of modeling requirements for simple rule checking (Figure 3.3). This information must then be properly encoded in IFC by the software developers to allow the proper translation and testing of the design program or the rule-checking software-authoring tools.

In summary, BIM has great potential to drive the development of automated or semiautomated code compliance–checking systems for building regulations, hence improving the efficiency and accuracy of the checking processes for designers as well as for government and local agencies. The next section reviews the key features of BIM-enabled code-checking systems and identifies the information content requirements by BIM-enabled automated checking applications.

BUILDING MODEL CONTENT

Since code compliance is mechanical work with a large workload, this task is normally time-consuming and error prone if it is not automated to improve efficiency and economy. The primary requirement in the application of any automated code compliance auditing is that object-based building models must have the necessary information to allow for comprehensive building regulations compliance checking. This will allow the automated code compliance–checking system to examine whether all the details of the building objects are in accordance with the relevant articles in the proper building codes. Unfortunately, most of the BIM models currently created by typical BIM-authoring tools such as REVIT and ARCHICAD to date do not normally include the level of detail required for automating building code or other types of rule checking.

In order to specify the information needed to perform code compliance checking, Model View Definitions (MVDs) must be developed. MVDs derive the data required for a specific type of code compliance auditing and establish a BIM model to enable more efficient automatic compliance checking. Information Delivery Manual (IDM) is the foundation of model view definition, so it is a critical step for the BIM model preparation phase. These standards are defined briefly below (Figure 3.4):

1. IDM is proposed to enable the identification and documentation of information exchange processes and requirements. IDM is now an official ISO standard (ISO 29481-1 and ISO 29481-2). ISO 29481-1 specifies the methodology and format for the development of an IDM, while ISO 29481-2 specifies a methodology and format for describing “coordination acts” between actors in a building construction project during all life cycle stages (ISO, 2012). The aim is to enable interoperability between software applications, to promote digital collaboration between actors in the building construction process, and to provide a foundation for accurate, reliable, consistent, and high-quality information exchange.

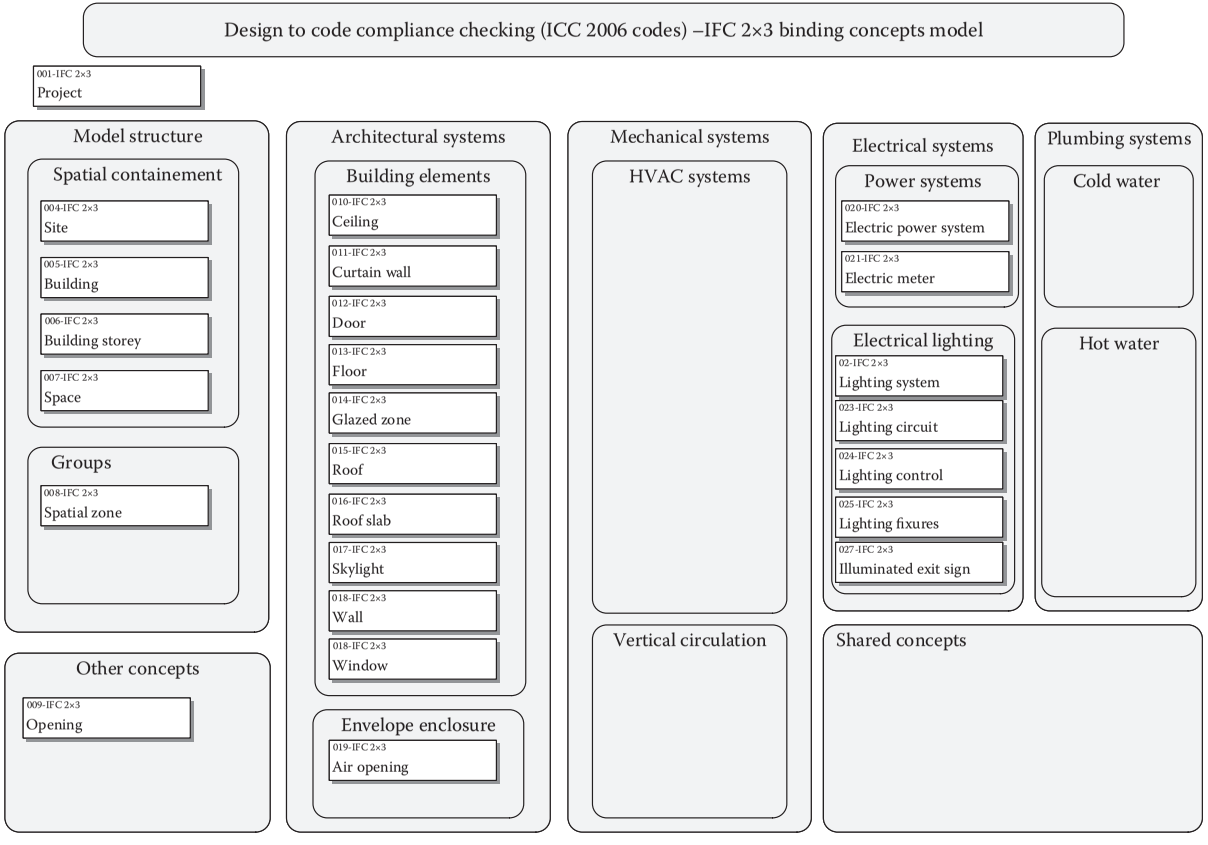


FIGURE 3.3 MVD for the International Energy Conservation Code. (Modified from IECC, International Energy Conservation Code, ICC, 2006.)

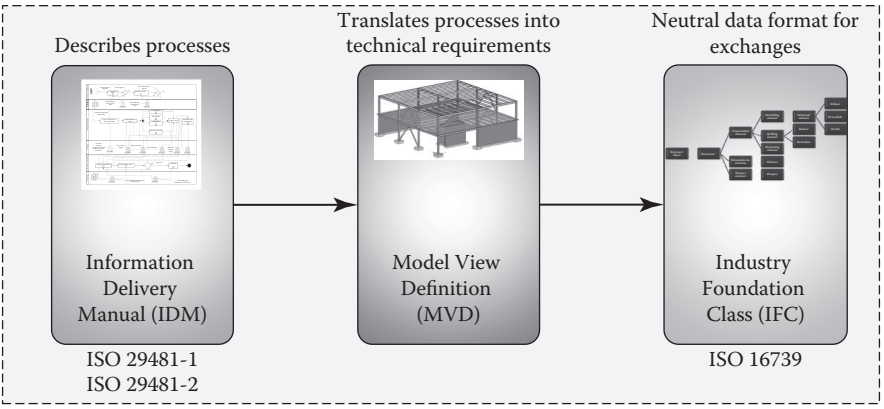


FIGURE 3.4 Standards for BIM content development.

2. Conceptually, the MVD is the process that integrates exchange requirements (ERs) coming from many IDM processes to the most logical model views that will be supported by software applications.
3. The implementation of these components will specify the schema and format for data to be exchanged using a specific version of the IFC standard to create and sustain a BIM application.

The evaluation of the BIM model for adequacy according to automated code compliance–checking specifications is performed by checking the IFC file exported from the BIM software to ensure that the IFC file includes all data structures specified in the MVD. The MVD is generated based on the exchange requirements specified in the automated code compliance–auditing IDM.

The development of the required model views goes hand in hand with the preparation of code conformance–checking functions. Some rule checking involves implicit properties, such as the ratio of glazing to floor area in rooms, the narrowest width in a passageway, and accessibility for the disabled in restrooms. Code conformance checking can be constructed on different types of model views in response to these requirements (Nawari, 2012a). An example of these model views for checking some sections of code has been developed by the International Code Council (IECC, 2006) and is shown in Figure 3.3. This model view is intended to facilitate BIM-based automated building code compliance checking. The scope for this MVD version includes provisions from the International Energy Conservation Code (IECC, 2006).

IDMs and MVDs are effective tools for defining the required information content in BIM models for automating building regulation compliance checking. These steps remove the ambiguity around whether the required information is captured properly or has not been established in the BIM model for code-checking processing. Once the BIM model is developed according to the standardized steps above, the required information for code compliance checking has been established.

IFC DATA MODEL

IFC provides comprehensive specifications for information about any construction project. In other words, they represent a set of internationally standardized object definitions. According to buildingSMART International (2015), IFC is defined as "an open specification for Building Information Modeling (BIM) data that is exchanged and shared among the various participants in a building construction or facility management project." The fundamental concept of IFC exchanges is that they are *object-based*. These objects have a geometry that is a 3D description and properties such as their element name, size, location, finishes, and cost. Some objects are "real" like beams or columns; some objects are abstract, like construction cost, calculation, quantities, and so on. Using IFC, the object's information can be exchanged between different software programs from diverse members of the AEC industry and the many related subsidiary participants. The IFC specification is developed and maintained by buildingSMART International (previously known as the International Alliance for Interoperability [IAI]). In a nutshell, IFC captures information from (1) all types of organization involved in the project (architects, engineers, constructors, facility managers, etc.), (2) all stages in the project life cycle, including initial requirements, design, construction, maintenance, and operation. An example of IFC hierarchal schema is depicted in Figure 3.5.

The underlying concepts of classes and objects in IFC are illustrated in Figure 3.6. The column class has various attributes that define the type of column, shape, size, material, and so on. Objects of the class will specify values for the various attributes.

An attribute of a class may be optional, a relationship to another class, or an aggregate (e.g., a column may have different geometric shape representations), as well as a combination of optional and aggregate. Furthermore, attributes can be classified into explicit, derived, and inverse attributes. Explicit attributes are those with direct visible values. Derived attributes get their values from an expression. Inverse attributes do not add "information" to a class but only a name and constrain an explicit attribute to a class from the other end (Figure 3.7).

The IFC schema representation is given in two main formats: (1) iconic or graphical notation used to establish the IFC schema, known as EXPRESS-G, (2) formal

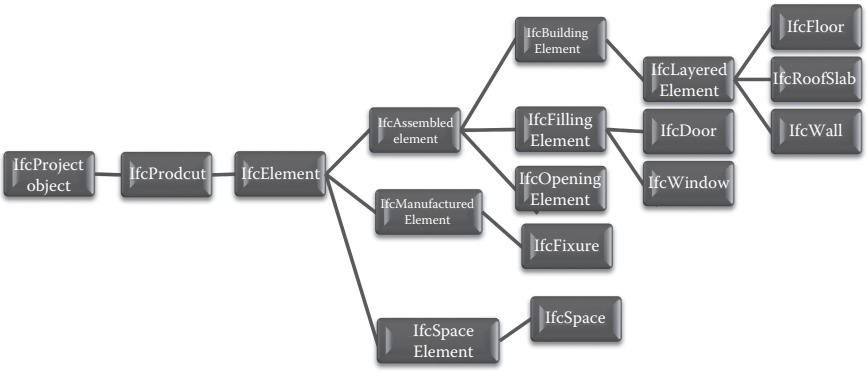


FIGURE 3.5 Example showing simplified part of the IFC classes hierarchy.

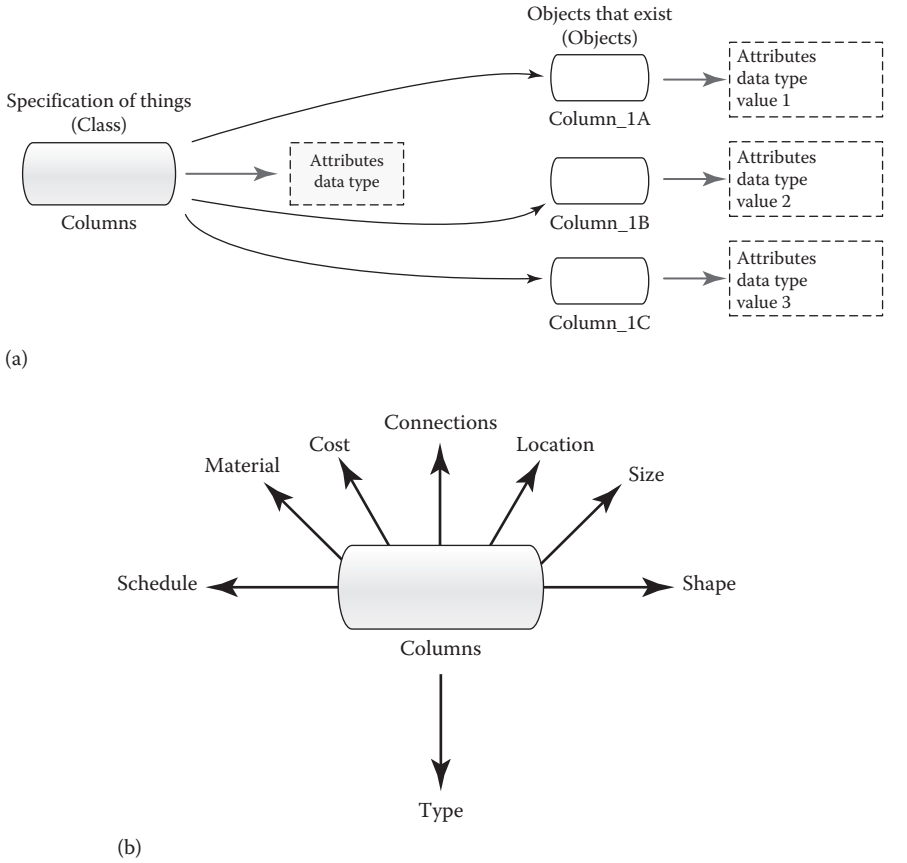


FIGURE 3.6 (a) Classes and objects and (b) definition of attributes of a class.

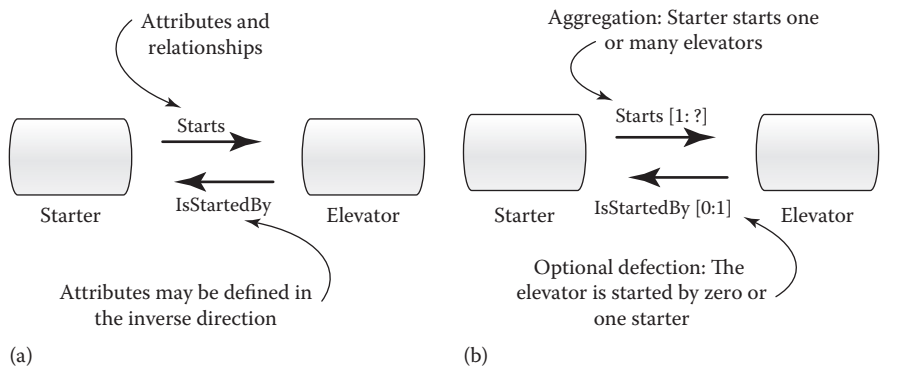


FIGURE 3.7 Illustration of the inverse attributes.

specification, which utilizes the international standard data definition language known as EXPRESS, a standard data modeling language for product data. It is important to note that EXPRESS is not a programming language. EXPRESS is formalized as part of the ISO Standard Exchange of Product (STEP) model (ISO 10303). EXPRESS is used to define data models for large-scale industrial applications, including manufacturing, engineering, defense, the oil industry, processing plants, and so on. In general, EXPRESS is defined as a conceptual schema language which provides for the specification of classes belonging to a defined domain, the information or attributes pertaining to those classes (wall, beam, column, etc.), and the constraints on those classes (unique, exclusive, etc.) It is also used to describe the relations that exist between classes and the numerical constraints applying to such relations. The language displays these as entity and type definitions, relationships, and cardinality. An alternative formal specification is also available in a version of Extensible Markup Language (XML) known as ifcXML. In the iconic schema, classes and objects are specified using graphical icons, such as a rectangle for a class or a line with an end marker for attributes (Figure 3.8). The end marker normally connects to a data type (which could be another class). Optional attributes are shown by a dotted line. Figure 3.8 depicts an example of the iconic representation of IFC schema. In this example, the *IfcBuilding* class has three obligatory attributes and one optional attribute. The obligatory attributes are *BuildingAddress*, *ElevationOfRefHeight*, and *Decomposes* and *DecomposedBy*. The optional attribute is *Expression*.

Iconic representation of the schema is then converted to a formal language, such as EXPRESS (Figure 3.9). This language is understood and verified by software applications. Its main purpose is for data exchange and database creation.

The EXPRESS language is utilized to define classes. Within the class definition, all the attributes and relationships that characterize it are declared. A class is declared by the keyword *ENTITY* and terminated by the keyword *END_ENTITY*. An entity declaration creates a class and gives it a name. The declaration is terminated by *END_ENTITY*. The EXPRESS language uses the semicolon character (;) to terminate an expression. Whitespace and carriage returns are ignored in interpreting or checking the validity of an EXPRESS schema, but they can be used to

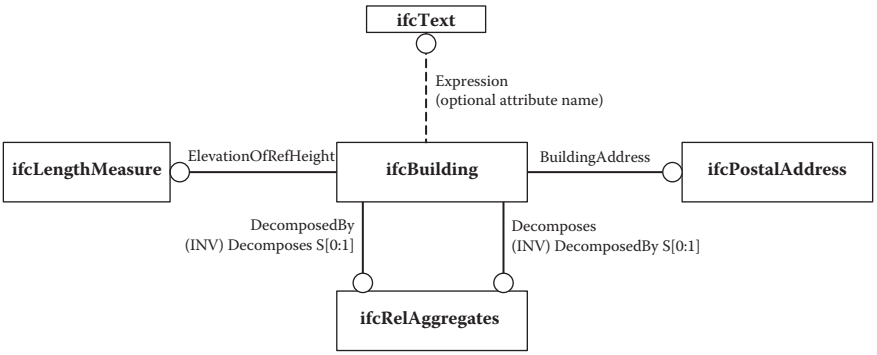


FIGURE 3.8 An example of IFC iconic schema.

```
1    ENTITY IfcBuilding;
2        ElevationOfRefHeight: IfcLenghtMeasure;
3        BuildingAddress: IfcPostalAddress;
4        Decomposes: IfcRelAggregates;
5        DecomposedBy: IfcRelAggregates;
6    WHERE
7        WR1: Decomposes:<>: DecomposedBy;
8    END_ENTITY;
```

FIGURE 3.9 IFC schema representation using EXPRESS language.

improve the layout of the schema so that it is human readable. An attribute represented by a relationship to another class is shown by the name of the relationship and the name of the class with which the relationship exists (in the direction of the relationship). Thus, the *IfcBuilding* with an *ElevationOfRefHeight* is declared as an *IfcLengthMeasure* class (Figure 3.9). Furthermore, EXPRESS allows for the organization of a class into subtypes. This expresses a parent–child relationship in which each subclass (referred to as a *subtype*) encompasses more specific detail than its parent superclass (referred to as *supertype*). For instance, *IfcFloor*, *IfcRoofSlab*, and *IfcWall* are subtypes of the *IfcLayeredElement* class (Figure 3.5).

IFC is proposed to be a high-level data model that is intended to be independent of any software implementations; that is to stay, it should be strictly neutral. It offers a consistent data structure for storing building information, but does not impose, or even allow, any particular method of implementing it into a software application. It is entirely up to the software-authoring services to decide on the method of implementation.

The EXPRESS data schema that represents IFC data can be encapsulated into digital files for the purpose of data exchanges that are file-based interactions. Alternatively, the IFC data schema can be represented in an object-oriented database and can be updated and shared remotely using any cloud computing platforms. In current practice, most BIM-authoring software tools offer end user interfaces with IFC in the "Save As" or "Export" dialogue of the software, where the IFC standard might be listed as one of the choices for saving or exporting the model data, in addition to proprietary data formats.

The overall schema of the IFC data model is divided into four layers (Figure 3.10): Domain layer, Interoperability layer, Core layer, and Resource layer (buildingSMART

International MSG, 2014). The Core layer provides a means of expansion of IFC based on the concept of classification encompassing the object for the existing target, the relationship between the objects or between the objects and the characteristic, and property. The layers have stringent referencing hierarchies; the general rule is that referencing can only take place downward in the hierarchy (Figure 3.10). This means that data in the resource layer must be independent and reference no classes above it. The other layers, however, can all reference data from the resource layer as well as all other layers below them. References within the same layer are permitted only for the Resource layer. The Resource layer hosts the resource structure that encompasses elementary definitions planned for defining objects in the top layers.

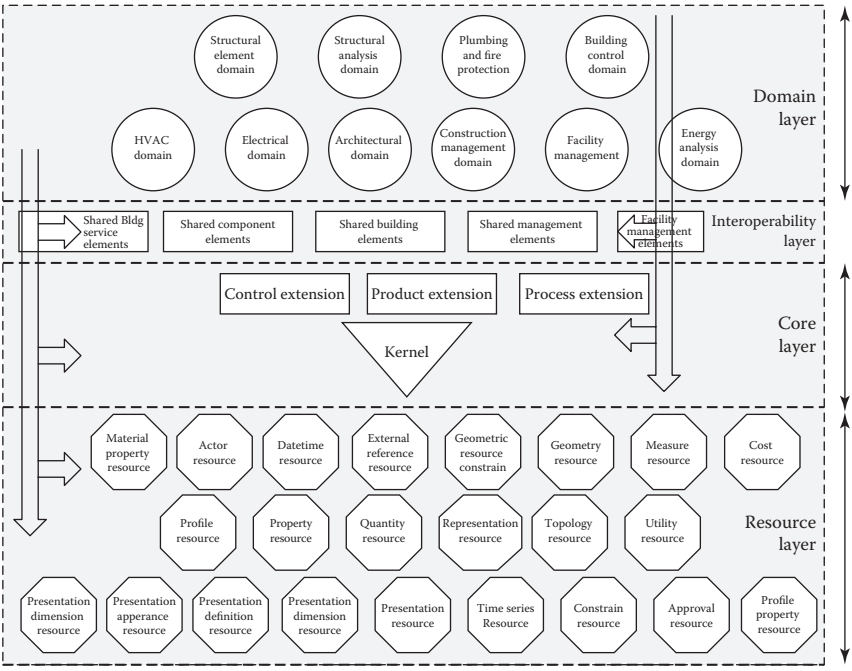


FIGURE 3.10 Structure of the IFC data model. (Modified from IAI, *IFC Technical Guide: Enabling Interoperability in the AEC/FM Industry*, 2000.)

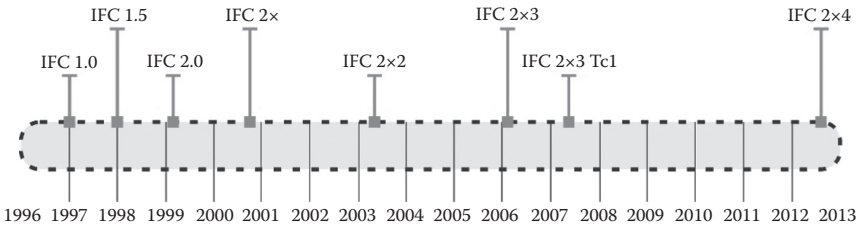


FIGURE 3.11 The IFC version timeline.

The Core layer involves the Kernel and Extension modules. The Kernel controls the model structure and disintegration, providing basic concepts concerning objects, associations, type definitions, attributes, and roles. Core extensions are concentrations of classes defined in the Kernel. The Interoperability layer offers the interface for domain models, hence giving an exchange mechanism for allowing interoperability between different disciplines. The Domain layer covers domain models for workflows in specific AEC areas or types of applications, such as architecture, structural engineering, and mechanical systems (IAI, 1999a,b, 2000).

IFC is presently considered one of the most appropriate schemas for improving information exchange and interoperability in the construction industry. Different versions of IFC have been released since its inception in 1997 (Figure 3.11). New applications have been developed, capable of parsing IFC-based models and interpreting and reusing the available information.

FROM IDM AND MVD TO IFC

The path to create an IFC data model begins with the development of IDM and MVD (Figure 3.4). This section provides a brief overview of the necessary steps, illustrating how exchange requirements in the IDM and the implementer's agreements in the MVD can be mapped onto the IFC schema.

- Establish the IDM, which seeks to provide the integrated reference for the process and data required by BIM by identifying the discrete processes undertaken within building construction, the information required for their execution, and the results of that activity (Nawari, 2012b). The IDM generally consists of two core parts: the first part is the process map, which defines the end user processes and information exchange between use cases. The second component is the list of exchange requirements (ERs). For example, the main groups of ERs for tensile structures can be recognized as the information exchanges required between various designers such as architects and engineers, architects and contractors, engineers and suppliers, engineers and contractors, and contractors and suppliers. The development of IDM starts with the identification of the data exchange–functional requirements and process scenarios for exchanges between those parties utilizing the *use case* concept (Nawari, 2012a). To create an IDM for such exchanges, a process map showing the specifics of data exchange requirements is given in Figure 3.12 (Nawari, 2014). The exchange model requirement offers a linkage between process and data. The span of the exchange essentials is the exchange of information about structural components and systems. The exchange models thus defined comprise a wide array of ERs to support the coordination of tensile structural analysis and design requirements with the building form and spatial order. For example, the exchange model AS_EM01 in Figure 3.12 denotes the data exchange requirements between designer and engineer during the conceptual design and may contain the following information (Nawari, 2014): (1) cables, fabric membranes, and geometrical properties of the structural forms (types of cable-layering

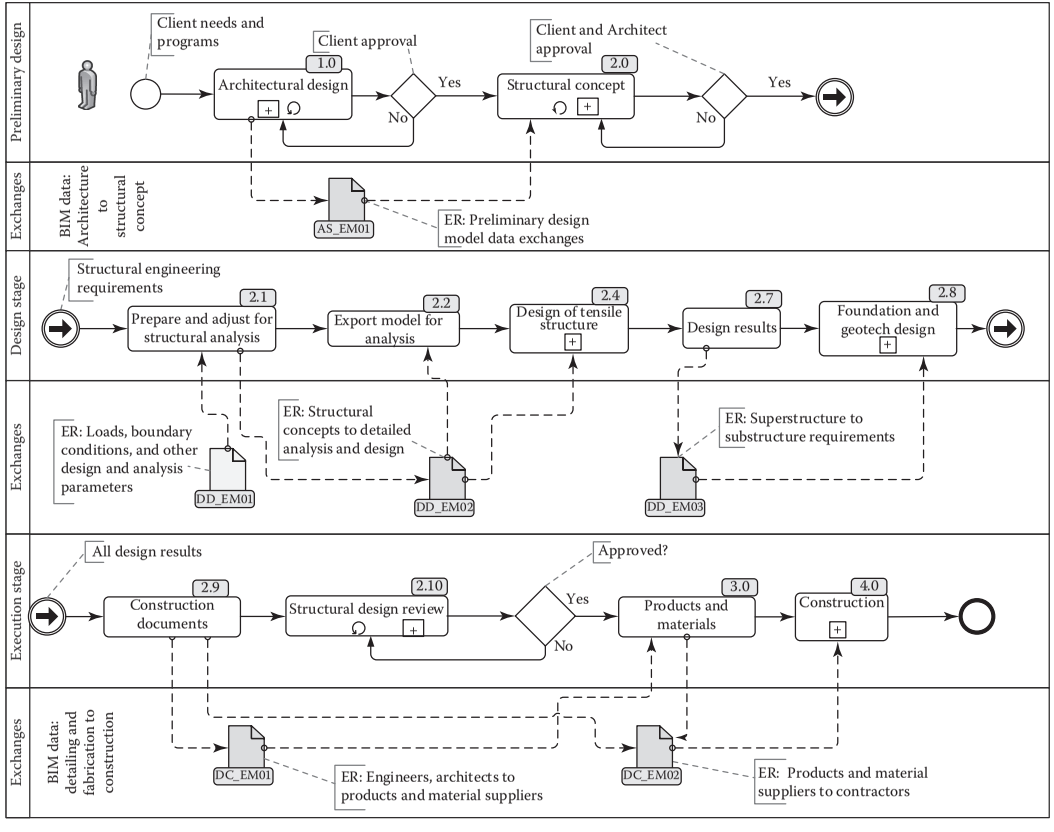


FIGURE 3.12 Process map for tensile structures (From Nawari, N.O., BIM standard: Tensile structures data modeling, *15th International Conference on Computing in Civil and Building Engineering*, Orlando, FL, June 22–25, 2014.)

systems, pretentions of cable beams, prestressed membranes and cable nets, arch-supported saddle surfaces, edge-supported saddle surfaces, etc.), (2) kinds of anchorage, masts, and membrane joint properties, (3) materials (steel, polyester, or vinyl meshes, laminated fabrics, coated fabrics, etc.), (4) temperature, fire, and moisture conditions, (5) exterior and interior finishes.

- Translate the IDM into technical requirements for software applications, relying on an appropriate set of standard data models defined through the MVD. These include all implementation agreements for all IFC concepts, such as classes, attributes, relationships, property sets, quantity definitions, and so on, used within a particular subset of the IFC data schema. In other words, MVDs provide the guidelines specifying how the information must appear in the IFC schema. They generally define a subset of the IFC data model that is necessary to support the specific data exchange requirements of the AEC industry during the life cycle of a construction project.

The definition of the MVD can take place in two main steps:

1. IFC independent: The IFC-independent part describes the view on a generic level without making any references to the IFC data schema.
2. IFC release specific: This part is performed separately for each supported IFC release and it describes how the generic definition in step (1) is implemented using a specific IFC release.

Figure 3.13 depicts a generic MVD definition for some of the exchange requirements given in Figure 3.12.

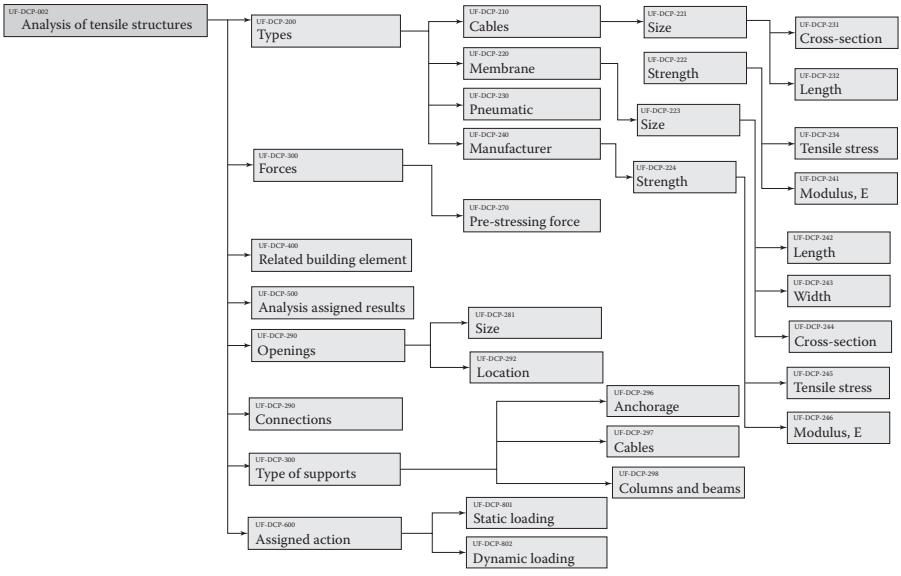


FIGURE 3.13 An example of a generic MVD for tensile structures.

- The last step is the implementation and testing of the exchange specifications, with possible final certification. Executing these steps will result in an IFC model that is more powerful and closely reflects real project requirements and accelerates the use of BIM in construction projects. IFC aims to enable the standardized sharing of information between project team members and across the software applications that are commonly used in the industry.

LEVEL OF DEVELOPMENT

Understanding the degree of information required for the elements in the model plays a significant role in ensuring its utilization for code compliance–checking purposes. The Level of Development (LOD) is the degree to which the element's geometry and attached information have been identified as necessary and sufficient by project team members to perform the required tasks when using the BIM model. In other words, the Level of Detail is simply how much detail is included in the model objects. In terms of terminology, Level of Detail can be thought of as input to the element, while LOD defines reliable outcomes.

The American Institute of Architects (AIA) defines LOD in BIM as "the minimum dimensional, spatial, quantitative, qualitative, and other data included in a Model element to support the authorized uses associated with such LOD" (AIA, 2013). In 2008, the AIA developed its first set of LOD definitions in AIA Document E202™-2008, "Building Information Modeling Protocol." Due to the rapidly evolving nature of the use of BIM, the AIA evaluated E202-2008, including the LOD definitions. The result is the updated and reconfigured digital practice documents AIA E203™-2013, "Building Information Modeling and Digital Data Exhibit," AIA G201™-2013, "Project Digital Data Protocol Form," and AIA G202™-2013, "Project Building Information Modeling Protocol Form," which are accompanied by a detailed guide document entitled "Guide and Instructions to the AIA Digital Practice Documents." The AIA's updated digital practice documents include revised LOD definitions (AIA, 2013).

Other efforts in LOD are represented by the work of the BIMForum (2015). The BIMForum developed the LOD specification, which is defined as a reference that enables practitioners in the AEC industry to specify and articulate with a high level of clarity the content and reliability of building information models at various stages in the design and construction process. The aim of this specification is to assist in explaining the LOD framework and standardize its application so that the LODs become more implementable in collaborative BIM environments. The LOD specification utilizes the basic LOD definitions developed by the AIA for AIA G202-2013 and is organized by the Construction Specifications Institute (CSI) in UniFormat 2010. It describes and illustrates the characteristics of model objects of different building systems at different LODs. This clear definition permits BIM model authors to define what their models can be relied on for, and allows downstream model consumers to evidently recognize the suitability and the limitations of models they are receiving. The intent of this specification is to help explain the LOD framework and standardize its use so that it becomes more useful as a communication tool. It does not prescribe what LODs are to be reached at what point in a project but leaves the specification of the model.

Basics of LOD

According to AIA G202-2013, there are levels of development known as LOD 100, 200, 300, 400, and 500. The BIMForum added LOD 350. A brief description of these LODs is given in Table 3.1.

TABLE 3.1
Building Information Model Levels Of Development (LOD)

Level of Development (LOD)	Description	Example
LOD 100	This level represents the conceptual design phase (schematic design). The model element may be graphically represented in the model with a symbol or other generic illustration (e.g., massing) but does not satisfy the requirements for LOD 200.	Massing or symbols showing the existence of an object but not its exact geometric size or location. For instance, lines used to represent columns and beams in the structure of a building.
LOD 200	The level generally represents the design criteria phase. The model element is graphically represented within the model as a generic system, object, or assembly with approximate quantities, size, shape, location, and orientation. Addition nongeometric information may also be devoted to the model element.	Building elements are modeled to reserve spaces—that is, as placeholders. Data resulting from this level of development object must be treated as approximate data.
LOD 300	Model object is graphically represented within the model as a specific system or assembly in terms of quantity, size, shape, location, and orientation. Nongraphic information may also be attached to the model element. Data can be extracted directly from the model without referring to nonmodeled information such as notes or dimension call-outs.	Model elements are more specific at this level. They are at the detailed design and construction document phase. Examples include 2D construction documents, general estimating, and 3D spatial validation.
LOD 350	The model element is graphically represented within the BIM model as a specific system, object, or assembly in terms of quantity, size, shape, location, orientation, and interfaces with other building systems. Additional nongraphic data may also be adhered to the model element.	This level generally goes beyond the detailed design. Details include such items as supports, connections, detailed estimating, and 3D spatial validation. The quantity, size, shape, location, and orientation of the element as designed can be measured directly from the model without referring to nonmodeled information such as notes or dimension call-outs.

(*Continued*)

TABLE 3.1 (CONTINUED)
Building Information Model Levels Of Development (LOD)

Level of Development (LOD)	Description	Example
LOD 400	The BIM element is graphically represented within the model as a specific system, object, or assembly in terms of size, shape, location, quantity, and orientation with detailing, fabrication, assembly, and installation information. Additionally, nongraphical information may also be attached to the model element. At this LOD, element is modeled at sufficient detail and accuracy for the fabrication of the represented component.	This LOD represents the fabrication details of the model and all the shop drawings associated with that level. The quantity, size, shape, location, and orientation of the element as designed can be measured directly from the model without referring to nonmodeled information such as notes or dimension call-outs.
LOD 500	Each BIM Element is a field-verified representation in terms of size, shape, location, quantity, and orientation. Any additional nongraphic information may also be attached to the BIM element.	This LOD is considered for project hand-over, maintenance and renovation, and other related facility management activities.

Many organizations have incorporated the contract specifications for LOD and/or developed their own customized ways to monitor and report the LOD during project development. For instance, the US Army Corps of Engineers (USACE, 2014) has developed the Minimum Modeling Matrix (M3) for this purpose. It is an interactive spreadsheet-based tool that provides instructions for recording the LOD for each element in the model (structural, architectural, mechanical, and plumbing). It is aimed at ensuring clarity and consistency regarding information exchanges between all the project stakeholders and avoiding any potential conflicts.

Defining LOD for Code Compliance Checking

Code compliance checking includes many data and functions with widely varying needs from models not adequately covered in the existing LODs. For example, all walls, windows, roofs, and slab types shall contain detailed information about structural components and energy performance (*U-value*), noise reduction, and fire rating. Further examples include, for instance, support for the remodeling and repurposing of a building, which requires a complete fabrication model, some of it field verified. Certainly, existing definitions of LOD have helped the building industry to formalize and communicate vital data among major project stakeholders; however, they are not sufficient for automated building regulation compliance checking. They need to be expanded to cover the needs of automated code compliance auditing.

The primary requirement in the proposed LOD is that object-based building models must have the necessary information to allow for automated code compliance

checking of the model. BIM objects being created normally have a family, a type, and properties. For example, an object that represents a structural column possesses a type and properties such as steel, wood or concrete, and sizes, and so on. Thus, the requirements for a building model to be adequate for code conformance checking are stricter than other requirements defined in current LODs. Architects and engineers creating building models that will be used for code conformance auditing must prepare them to that level of development.

The automated code conformance domain represents a new level of details and requirements for IFC models. This should be achieved by developing the appropriate IDMs (ISO 29481-1 and ISO 29481-2) and MVDs for automated code conformance checking (Nawari, 2011).

SUMMARY

BIM is now acknowledged to be the most effective platform in exchanging information across AEC industries for design, construction, and facility management activities. BIM supports the development of various applications that conduct rigorous analyses and simulations for the building before construction works start, thereby improving the efficiency and accuracy of the design and construction of a facility. At the center of that are the tools that facilitate the automatic or semiautomatic checking of compliance against building regulations and standards, hence improving the certification process for all major participants, such as the owner, certifying authorities, designers, and contractors.

In reference to automatic code checking, BIM data can be represented in two ways: IFC and proprietary BIM data representation (e.g., the Autodesk Revit platform). The application of proprietary BIM data representation may be essential to protecting intellectual property; however, the concealed nature of data representation is a real problem for data sharing and interoperability between applications. In contrast, the IFC data model is a free and open standard that enables interoperability across various BIM-authoring tools. Furthermore, the IFC data representation has the following beneficial characteristics: (1) the IFC schema is intended to include all disciplines of the AEC industry and all phases of a construction project; (2) the IFC schema is defined using EXPRESS, which is a conceptual schema language, certified as the official international standard ISO 10303-11; and (3) the IFC schema is also recognized as international standard ISO/IS 16739.

EXPRESS LANGUAGE

As defined earlier in the IFC section, EXPRESS is a conceptual schema language that provides the description of classes of certain domains, the information or attributes pertaining to those classes (Walls, Columns, Beams, etc.), and the constraints on those classes (unique, exclusive, etc.). It is also used to define the relations that exist between these classes. The EXPRESS language became an international standard in 1994 (ISO 10303-11), titled “The EXPRESS Language Reference Manual.” The language aims at extensive collaboration to develop a means by which to specify the exchange, sharing, and archiving of technical information. Since that date, many different international standards have used EXPRESS to represent neutral data schema. EXPRESS enables the definition of information models that create a common

language to address both data interoperability and integration issues. Data interoperability occurs when two interfacing applications comply with a common information model. These applications can then create and access either exchanged or shared data sets. Data integration arises when entities try to establish data repositories containing unified database systems. These data are typically derived from many different information systems. EXPRESS also offers the basis for reliable archiving systems. Its information model provides an explicit specification of data semantics, and thus, even without access to the original software, future users can analyze the model and understand the content of the corresponding archived data sets.

Establishing the information models in EXPRESS provides a basis on which data consumers can have unconstrained access to the data sets within the system. The language fundamentals were introduced earlier in the IFC section. Complete details of the language can be found in ISO 10303-11, which defines two basic representations of the language—namely, a lexical and graphical representation called EXPRESS-G defined in annex E.

IFCXML DATA MODEL

ifcXML is defined as the XML equivalent to the EXPRESS-based specification of the IFC data file. The ifcXML data representation can provide a way to work with a subset of the IFC model definitions (Figure 3.14). This enables one to reduce the complexity of the IFC data model and thus to simplify implementation. It also reduces the file size, which leads to lower resource requirements. ifcXML is not only based on a subset of the entire IFC model but also makes further use of the configuration capabilities of the ISO 10303-28 standard for translating the IFC EXPRESS model into the ifcXML XML Schema Definition (XSD) model. In addition to the configuration settings for the entire ifcXML XSD model, it mainly adjusts relationships for more efficient data management (buildingSMART International, 2016).

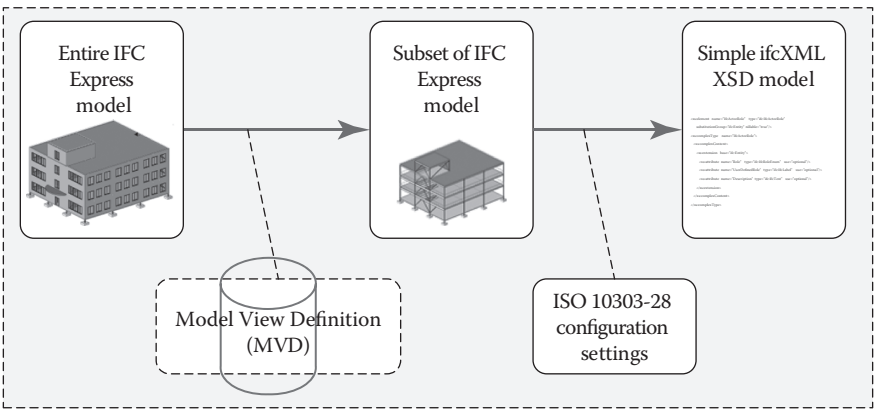


FIGURE 3.14 Specification for ifcXML 4 XSD. (Modified from buildingSMART International (2016). http://www.buildingsmart-tech.org accessed June, 2016.)

```
1       <IfcCartesianPoint id="ID36">
2         <Coordinates ex:cType="list">
3             <IfcLengthMeasure pos="0">0.</IfcLengthMeasure>
4             <IfcLengthMeasure pos="1">0.</IfcLengthMeasure>
5             <IfcLengthMeasure pos="2">0.</IfcLengthMeasure>
6         </Coordinates>
6       </IfcCartesianPoint>
```

(b)

```
<IfcCartesianPoint  id="ID36"  Coordinates="0. 0. 0."/>
```

(a)

FIGURE 3.15 (a) ifcXML 2 × 3 format (with tags *Coordinates* and *IfcLengthMeasure*), (b) ifcXML 4 format (with attribute *Coordinates* and space-separated values). (From buildingSMART International, 2012, http://www.buildingsmart-tech.org, accessed June, 2016.)

The ifcXML representation is an implementation of the ISO 10303-28 standard. This standard provides an XML schema specification that is an automatic conversion from the EXPRESS (ISO 10303 part 1) representation of the IFC schema. The mapping from EXPRESS to XML schema is guided by a configuration file that controls the specifics of the translation process, known as ifcXML XSD. For ifcXML, this configuration file is standardized and published for each version of the corresponding IFC schema.

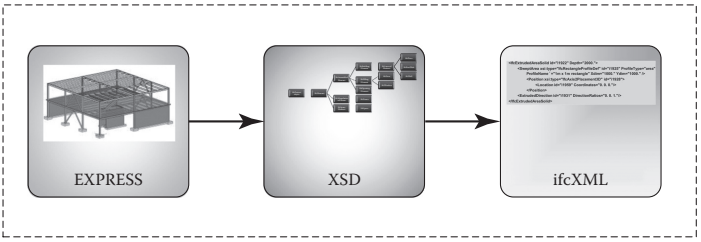
With the new development of the IFC4 standard, ifcXML has undergone a complete redevelopment. The new ifcXML has been simplified to obtain smaller and more compact XML data files. The development has been executed under the code word "simple ifcXML" (Liebich and Matthias, 2012). buildingSMART International publishes the official ifcXML XSD for all official releases of the IFC data model standard. The ifcXML4 release is published as an XSD, derived from the IFC EXPRESS model. The method of how to translate the IFC EXPRESS model into the ifcXML XSD model follows the international standard ISO 10303-28, "XML Representation of EXPRESS Schemas and Data" (buildingSMART International, 2016). Figure 3.15 illustrates examples of ifcXML for the IFC 2 × 3 format and the new IFC 4.

MAPPING EXPRESS INTO IFCXML

The entities and attributes that are declared within the EXPRESS data model are mapped either into XML attributes or into a sequence of inline XML elements. Only the explicit EXPRESS attributes are mapped by default; the inverse (with exceptions) and derived EXPRESS attributes are not mapped. The detailed mapping specification for EXPRESS attributes depends on their data type and on whether they are aggregates or not. Instance elements can be written *by-value* and *by-reference*. The by-value instance elements shall define a local XML identifier, which is defined by

the *id* attribute and can be used as a reference. The by-reference instance elements contain a reference to a by-value instance element, which is defined by the *ref* attribute (buildingSMART International, 2016). Figure 3.16 depicts an example of mapping and EXPRESS data definition into the ifcXML data model.

The published ifcXML XSD can be used by various software vendors to create equivalent ifcXML files (Figures 3.16 and 3.17).



(a)

```
1       ENTITY IfcActorRole;
2           Role : IfcRoleEnum;
3           UserDefinedRole : OPTIONAL IfcLabel;
4           Description : OPTIONAL IfcText;
5       END_ENTITY;
```

(b)

```
1  <xs:element  name="IfcActorRole"   type="ifc:IfcActorRole"
2      substitutionGroup="ifc:Entity"  nillable="true"/>
3  <xs:complexType   name="IfcActorRole">
4    <xs:complexContent>
5      <xs:extension  base="ifc:Entity">
6          <xs:attribute  name="Role"  type="ifc:IfcRoleEnum"  use="optional"/>
7          <xs:attribute  name="UserDefinedRole"  type="ifc:IfcLabel"  use="optional"/>
8          <xs:attribute  name="Description"  type="ifc:IfcText"  use="optional"/>
9      </xs:extension>
10   </xs:complexContent>
11 </xs:complexType>
```

(c)

```
<IfcActorRole  id="i1546"   Role="architect"   Description="general architect"/>
```

(d)

FIGURE 3.16 (a) Mapping process, (b) EXPRESS IFC entity and attributes, (c) XML Schema Definition (XSD), and (d) ifcXML file. (From buildingSMART International, 2012, http://www.buildingsmart-tech.org, accessed June, 2016.)

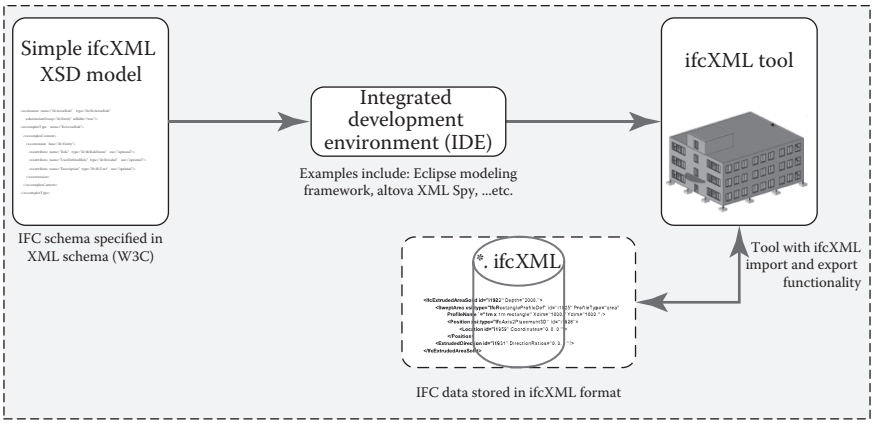


FIGURE 3.17 Code generation for ifcXML implementation.

BIMXML SCHEMA

BIMXML stands for Building Information Model Extended Markup Language. It describes data for the built environment (sites, buildings, floors, spaces, and equipment and their attributes) in a basic spatial building model (extruded shapes and spaces) for BIM collaboration. XML currently powers the Web and allows many diverse sets of information to be shared on it in real time. It is used to develop simple connections to other data and supports lean Web service transactions of partial models. BIMXML leverages the existing established standards of XML to allow

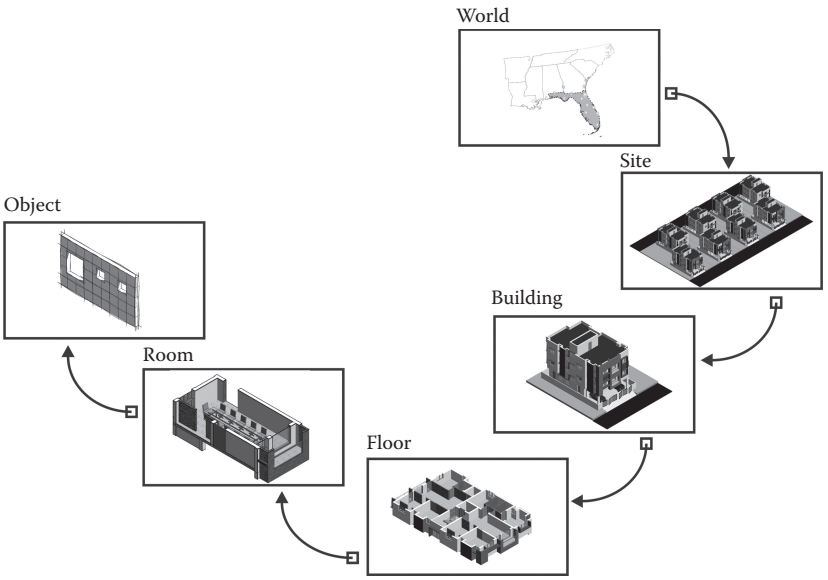


FIGURE 3.18 Object hierarchy in BIMXML.

AEC industry data to be shared via the Web the same way other ecommerce applications have utilized it to attain significant business gains. Onuma (2001) created BIMXML many years ago and has made it free to anyone who is interested in leveraging XML code that connects almost any Web information to the global building industry through BIM based on IFC data structures. The schema was developed as an alternative to full-scale IFC models to simplify data exchanges between various AEC applications and to connect building information models through Web services (BIMXML, 2016). BIMXML describes a specific view of BIM that includes georeferenced, simplified 3D objects of a site, multiple buildings, floors, spaces, and objects, and attributes related to each of those (Figure 3.18). It does not include many of the more complex elements of IFC. The main features of BIMXML, which satisfy IFC compatibility while offering BIM data for collaboration over the Web in real time, include (1) the capability to describe general object categories, (2) the ability to establish relationships between object properties, and (3) the capacity to generate partial BIM models on the Web.

BIMXML is currently used by some vendors for actual projects, such as the Onuma System (Onuma, 2011), Data Design System (DDS) Viewer, vROC, Tokmo, BIM Connect, and various plug-ins for CAD BIM-authoring tools (e.g., Revit, SketchUp, ArchiCAD). For example, the Revit Onuma plug-in employs BIMXML to improve Revit data sharing. The BIMXML plug-in allows projects to be imported to and from Onuma. The plug-in also permits Revit models to be loaded with meaningful data and basic geometry from SketchUp, ArchiCAD, Bentley products, ArcGIS, Google Earth, and other software compatible with the basic XML standard. The BIMXML plug-in for Revit and other programs comes with Onuma System Web-based software, Onuma Editor Pro, and Studio Pro.

The XSD schema is available at http://www.bimxml.org/xsd/001/bimxml-001.xsd. BIMXML is basically just a simplified ifcXML data model. Most of the XML tags in BIMXML start with the name "BIM," but if you change that to "ifc" you really will see the similarity with ifcXML. Figure 3.19 depicts a portion of a BIMXML file showing a sample export of a single building from the Onuma BIMXML system.

In summary, the chief features of BIMXML include the following:

- It describes objects using only key, but general, data (e.g., sites, buildings, floors, spaces, and equipment (Figure 3.18). However, it permits numerous amounts of attributes for these objects.
- It is extremely lightweight, with file sizes of complex buildings measured in kilobytes rather than megabytes. This allows complex BIM models to be viewed on smartphones, tablets, and other handheld smart devices.
- BIMXML is "IFC-like" but simplified and only focuses on a specific subset of the BIM.
- BIMXML was designed to have a very simple way to exchange data.
- The flat structure of BIMXML to exchange data and the ability to capture the geometry of spatial entities (stories, spaces, slabs, equipment, etc.) with simplified geometries are the two most significant differences to ifcXML.
- It supports local coordinates systems for adding geospatial data categories to all the sites, buildings, floors, spaces, and pieces of equipment as well

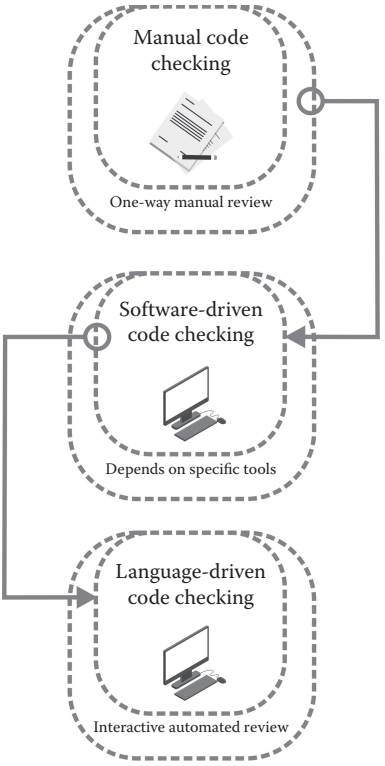


FIGURE 3.19 An example illustrating a part of the BIMXML data model. (BIMXML.org, website for Building Information Model Extended Markup Language (BIMXML), http://bimxml.org, accessed June, 2016.)

as data referenced in geospatial information systems (GIS) such as ArcGIS and Google Earth.

- Thus, every item in a BIM model, right down to a chair, can have a geospatial location associated with it in real time and can be represented in BIMXML. The same holds true for data in the Construction Operations Building Information Exchange (COBie). COBie is one of the main standards in BIM for facility management (FM). COBie is an open, vendor-independent industry standard that describes the product and process of collecting and validating building life cycle data during design, construction, and commissioning (BuildingSMART, 2010). All COBie data objects can be georeferenced with BIMXML.
- BIMXML supports 3D geometry; however, it is purposefully centered on capturing the maximum amount of data using as simple geometries as possible. This eradicates the need to resolve geometries to a level of detail that is not required to accurately represent the data under consideration. Detailed geometries are handled by IFC data model.
- To avoid confusion with IFC, BIMXML deliberately uses different terms than the IFC data model for similar entities.

- It supports service-oriented architecture and cloud computing.
- It is important to note that BIMXML is not intended to be a replacement for IFC schema but a means to easily share data (Onuma, 2011). It was developed as an alternative to full-scale IFC data models to simplify data exchanges between various AEC applications and to connect BIM models through cloud services.

BIMXML has been used successfully and found to be beneficial to a number of organizations such as the US Department of Homeland Security (DHS), the US Army Corps of Engineers (USACE), the US Coast Guard (USCG), the US GSA, and the California Community Colleges System (Smith and Bordenaro, 2011). In most of these cases, these organizations were interested in integrating their existing data in BIM projects with ArcGIS and Google Earth. Furthermore, utilizing the capability of BIMXML Web services' connectivity and open standards, they were able to share data using standard interfaces and BIMXML application programming interfaces with various proprietary software applications.

BUILDING ENVIRONMENT RULE AND ANALYSIS LANGUAGE

Overview

The Building Environment Rule and Analysis (BERA) language is intended to be a highly customized domain-specific language for handling design review issues in building models in the architecture, engineering, and construction (AEC) industry sectors. It aims to provide a rule-checking language architecture that is easy to use and has high fidelity, extensibility, and compactness, and subsequently, to develop a high-level and domain-specific language (Lee et al., 2015).

The BERA language is based on the BERA Object Model (BOM). BOM is a simplified concept of the complex state of building models rather than the computation-oriented abstraction that is normally proposed to cover broad-ranged issues. Originally, BERA data schema run on building objects, centering on assessing building circulation and spatial programs. However, BOM has the potential for extending to other areas of the AEC. Target users of the BERA language are domain experts such as architects, engineers, designers, reviewers, owners, managers, students, and so on, rather than BIM software developers (Lee et al., 2015). The proposed language seeks to handle building information models in an intuitive manner in order to outline and analyze rules that can be applied to verify the arrangement of building objects, the structure and layout of buildings, and other code requirements (Figure 3.20).

The BERA language has developed to become one of the programming languages that handles building data models represented in EXPRESS or other similar data schema. It is a high-level computer programming language. However, it is not a general-purpose programming language such as C++ or Java. The BERA language is a domain-specific programming language for various building information models and rule checking. The syntax of the BERA language

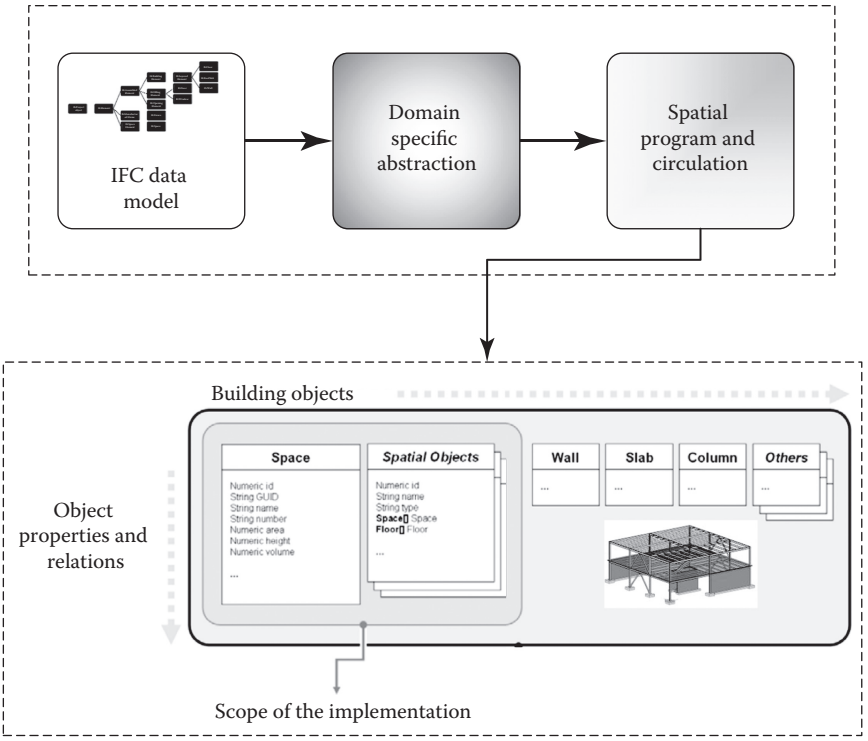


FIGURE 3.20 The concept of BERA in comparison with other methods of code checking. (From Lee, J.K., Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program, Ph.D. thesis, Georgia Institute of Technology, 2011.)

is structured similarly to most programming languages such as the Java and ECMAScript languages. The following are the purposes and benefits of using the BERA language (Lee, 2011).

1. The BERA language is essentially a language for performing rule checking and enables the automatic review of building models. The concept design review is an important phase of a building project. The BERA language is aimed to help in the decision-making process in this early phase of building design in an easier and faster way.
2. Contrary to the general-purpose programming languages, the BERA language is meant to be easy to use for novices or nonprogrammers. This means that domain experts such as architects or engineers can obtain much more control and better handling of reviewing building models.
3. It is intended to be easier than general-purpose languages but still powerful enough to handle domain-specific problems. The BERA language supports various functional statements for handling both the complexity of design rules and the complex relations of space objects and properties. Similar to

general-purpose programming languages, BERA offers logic operations, logic values, recursion, negation, inheritance, polymorphism, algebraic operations, and so on.

4. In reference to its extensibility, the BERA language provides an open-ended model for the abstraction of a building. According to its development, the BERA language will also be expanded to other building objects. In its first version, the initial BERA language and its tool implementations aim to handle a subset of building objects—space objects, groups of spaces, their properties, and their relations—to validate the application for checking the conformance of building circulations and spatial program rules.

Main Components of BERA

One of the main components of the BERA language is its abstraction of the building objects, as its root data structure is derived and computed from the given building model. Normally, a facility encompasses a wide range of objects in multiple hierarchical structures. Many researchers have focused on the area of building product modeling for the digital representation of buildings, as discussed in the previous sections. The main goal of BERA is to attain an intuitive concept of building information for the usability of the BERA language, similar to the Document Object Model for object-oriented languages such as C# or Java. For example, BERA uses the *Space* object as a wrapped data type inherited from *BuildingObject* (in terms of object-oriented concepts, this corresponds to the superclass *BuildingObject* and its subclass Space) from the *IfcSpace* entity, like other wrapper classes that are composed of an array or collection of user-defined data from the *IfcSpace* entity, including primitive data types (Lee, 2011). Similarly, the Wall object or any other type of building object can be occupied within the BERA objects, according to its domain rules and specifications (Figures 3.21 and 3.22).

The *Space* object is one of the most important data entities in the BERA language for assessing building circulation and spatial programming rules in the execution scope of the first version of BERA. A facility is naturally recognized in terms of semantics as an assembly of spaces (rooms) during the design process. In such a facility, these spaces are recognized by names or numbers (e.g., room names or room numbers) rather than columns, walls, or slabs, even if they are also named/numbered for other purposes. This type of building object-oriented approach marks fundamental issues in BERA language principles such as its objects, properties, and operators.

The BERA language has a strong tie to the IFC data model (Figure 3.20). An abstract model from these existing building models, especially from IFC, is BOM. Abstraction is a significant process in programming language design, since some of the language keyword tokens and syntaxes will be derived from this model. Moreover, this simple abstraction will be utilized by end users, while the complicated relationship with IFC data schema is executed using the implementation backend of the BERA language (Lee, 2011). Users of the BERA language can utilize the benefits of the BOM in the real program. For instance, if there is a group of a space called *office* and its name and related properties are given in the object *IfcClassification*,

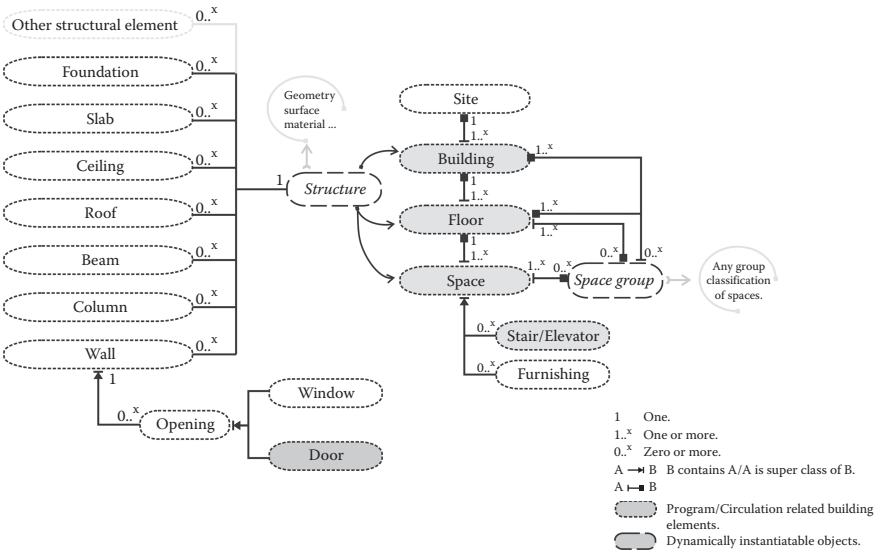


FIGURE 3.21 BERA language overview.

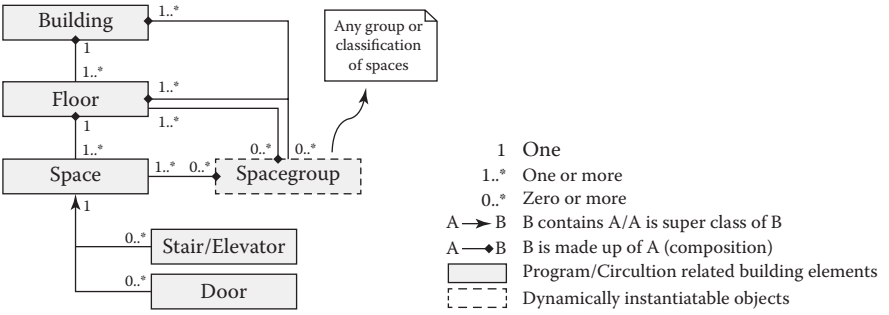


FIGURE 3.22 Overview of BERA object model. (From Lee, J.K., Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program, Ph.D. thesis, Georgia Institute of Technology, 2011.)

BERA users can regain the name of an invoked space object utilizing this simple dot notation (Figure 3.23).

However, for obtaining the *office* object using IFC-centered implementation, the following code statement is required (Figure 3.24).

This dot notation-based example depicts how BOM is intuitive and easy to use when developing BERA programs. Users only need to know the statement depicted in Figure 3.23. At the highest level, a BERA program has four components:

- BERA reference directive: *bReference*
- BOM definition and declaration: *bBOMDef*

```
space.office
```

FIGURE 3.23 An example of using BOM.

```
1    space.ifcRelAssociatesClassification.ifcClassificationReference.ifcLabel.getString()
```

FIGURE 3.24 An IFC equivalent to the statement in Figure 3.23.

- BERA rule definition: *bRuleDef*
- BERA execution statement: *bExeStat*

The reference declaration (*bReference*) has two sub-elements: reference statement (*bRefStat*) and building type (*bBuildingTypeStat*). They can be used as various references in actual BERA programs. These are elective objects that can be invoked by users. The reference statements mostly rest on back-end applications and are equally valuable for importing external data sets. The key objective is to improve the usability of the BERA language without extra programming extensions. This is quite similar to the directives in general-purpose programming languages, such as INCLUDE in C++, IMPORT in Java, and USING in C#.

The *bRefStat* statement can be invoked to import supplementary libraries similar to the directives in general-purpose programming languages, importing external definitions for building type–specific or project-dependent spatial data, importing external predefined rules for a certain rule scheme, and so on. Specifically, the keyword *reference* supports external data connections via internet protocols, similar to the way one imports an XML scheme definition in an application.

On the other hand, *bBuildingTypeStat* can be invoked to describe building type using either double-quoted strings or predefined building-type terminal tokens in capital letters. It offers a reference for the set of space types, properties, and rules within a spatial analysis domain. In other words, the *bRefStat* statement allows one to invoke many instances, while *bBuildingTypeStat* allows only one statement. It is important to note that the BERA *reference* declaration statement supports not only the access to the local paths but also the network access to the web URL.

BOM is the central data schema for the BERA language definition and implementation. It is a standardized data structure for all different BERA implementations and environments. In BERA, a building is made up of building objects or classes. The declaration of the object model definition through the declaration of *bBOMDef* allows access to the various classes of objects of the building model. Access to all objects and the related parameters can be achieved using BERA syntax through the so-called dot notation. For example, the statement depicted in Figure 3.25 illustrates the instantiation of the *Space* object that has the *Floor* class of objects, which is a subclass of *Space* and has a property called *name*. That property has been assigned the value of *Ground Floor*.

In its first version, BERA spaces are the key objects of the current implementation of the BOM. In definite BERA applications, users can define spaces not only through predefined explicit space objects directly from the building model space object but

```
Space.Floor.name = "Ground Floor"
```

FIGURE 3.25 Access objects and properties in BERA using BOM.

```
Space  myOfficeSpaces (Space ss) {
        ss.name  =  "office";
        ss.Floor.height  >  12;
}
```

```
ObjectType  name(args) {
        Statement 1;
        Statement 2;
        …
}
```

FIGURE 3.26 (a) Object definition using BOM and (b) an example BERA code.

also via dynamically grouped space objects such as a circulation path, departments, and so on (Lee, 2011). The general format of the statements for BOM are shown in Figure 3.26. Examples of valid BERA application segments within the rule *bBOM-Def* are shown in Figure 3.27.

In the example on line 2 of Figure 3.26, a user variable *bigSpaces* encompasses all the space objects that are larger than 1000 ft^2 from the given model. On lines 5 and 6, the examples show that users can define their own dynamic and varied space groups consuming as many as properties defined in the *Space* or *Floor* objects. Figures 3.28 and 3.29 illustrate the thorough spatial objects within the definition of BOM.

The third component of the BERA program is rules definition. Like the BOM, the rule definition also takes advantage of the use of dot notation access to BOM, in addition to the expressive notations by quantification and conjunction issues. To deal with the complexity of the rules, the BERA rule definition offers (1) easy access to the objects' properties and as many as BOM provides and (2) rich predicates to express many kinds of rule declarations, including logic operations, logic values, recursion, autoiteration, autocasting, negation, inheritance, polymorphism, and so on. Furthermore, BERA permits the inheritance of rules by invoking the *extends* keyword. If another rule, named Rule2, is required to be described utilizing the user-defined rule Rule1, a statement covering Rule1 can be used. Rule definition statements for Rule2 need to handle only the new characteristics of rules on top of the rules from Rule1. The general format of such rule statements in valid BERA program segments within the rule *bRuleDef* are shown in Figure 3.30.

```
1       Space space1 = getSpace("office");
2       Space bigSpaces = getSpace(Spaces.area > 1000);
3       SpaceGroup commonSpaces = getSpace("corridor") + getSpace("lobby") –
        getSpace("private");
4       Path path1 = getPath("office", "lobby");
5       Space myOffice {
                Space.name = “office”;
                Space.area < 500.0;
                Space.Floor.height > 10;
        }
6        Floor lowGroundFloors {
                Floor.elevationHeight < 100;
                Floor.number < 6;
                Floor.number >= 1;
        }
```

FIGURE 3.27 Example of code segments in BERA showing user-predefined objects. (From Lee, J. K., Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program. Ph.D. thesis, Georgia Institute of Technology, 2011.)

The last component of the BERA language is the invoking of execution statements: *bExeStat*. Similar to the other three objects *bReference*, *bBOMDef*, and *bRuleDef*, *bExeStat* is also optional, since the execution of the program is not obligatory when users define rules for later usage. Some examples of valid BERA program segments within the *bExeStat* rule are depicted in Figure 3.31.

The BERA language architecture involves two main parts (Lee, 2011):

- *Front-end: BERA engine*: This covers user-produced language programs, the BERA translator/interpreter to the target language, and other intermediate representations and executors for generating the BOM. This front-end component is standard for all implementations and environments.
- *Back-end: Custom engine*: The BERA language could not be executed without a given building model. BIM-authoring tools are another important platform. SMC is one platform example that the BERA engine utilizes, both in reference to object models and implementations. This back-end implementation differs by platforms, but the target intermediate model is always the BOM.

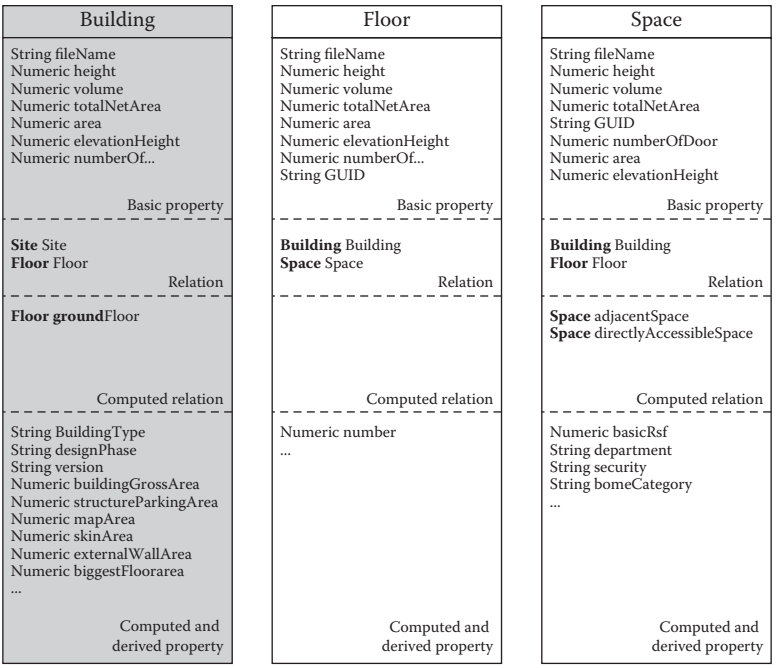


FIGURE 3.28 Spatial objects and their properties: Statically instantiable BERA object. (From Lee, J.K., Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program, Ph.D. thesis, Georgia Institute of Technology, 2011.)

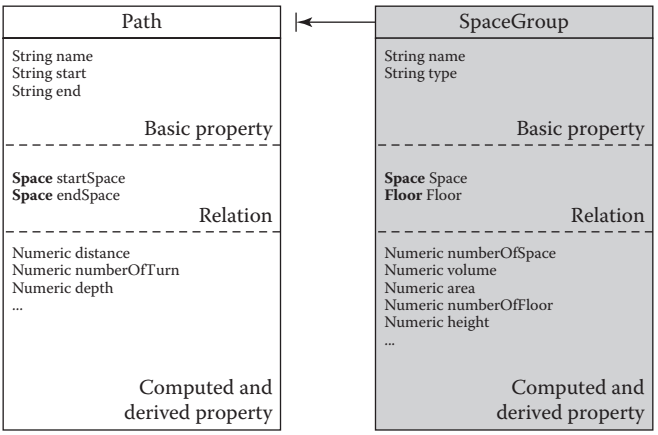


FIGURE 3.29 Spatial objects and their properties: Dynamically instantiable BERA object. (From Lee, J.K., Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program, Ph.D. thesis, Georgia Institute of Technology, 2011.)

```
1       Rule myrule1(Space space1) {
2       space1.area > 500;
3       }
4        Rule myrule2(Space space2) {
5                space2.area > 1000;
6                space2.Floor = "Level_1";
7                space2.security = "public";
8       }
9       Rule circRule1(Space start, Space end) {
                 path = getPath(start, end);
                 path.Space.security = "restricted";
        }
4       Rule circRule2(Space start, Space end) {
                 path = getPath(start, end);
                 path.Space.security = "restricted";
                 path.one.Space.name = "gate";  or path.distance < 10;
}
```

FIGURE 3.30 Examples of rule statements.

```
1       get(Space);
2       get(mySpaces);
3       get(myPaths);
4       myRule("department");
5       myRule(departmentSpaces);
6       myRule(getSpace("classroom"));
7       myCirculationRule01("lobby", "entry");
```

FIGURE 3.31 Execution statements within the rule *bExeStat*.

In summary, A BERA program is enfolded with *BERABEGIN* and *BERAEND* statements to separate it from other embeddable programing languages such as C#, C++, or Java. It is reliant on the target language to be interpreted and compiled for

the final program execution phase. If the *begin* statement is absent, BERA will treat all input codes as pure BERA language to the end of the lines. A semicolon (;) is used to terminate a single statement if there are no other code block indicators such as curly brackets or parenthesis. To denote BOM properties and relations, the classical dot notation known in object-oriented programming languages is used. To acquire a user-defined object, *get (arguments)* is used. Other building objects can be accessed using basic predefined methods such as *getSpace*, *getFloor*, *getPath*, and so on.

The main statement types in BERA are building objects and rules. These represent typical class, method, user-variable definitions, or a user-defined object, as shown in Figure 3.32.

A valid example of a BERA code is illustrated in Figure 3.33.

In Figure 3.32, the *ObjectType* can be *Space* or *Floor*, as defined in the BOM lexicon. Variable objects can be user-defined variables, and a statement is written using the dot notation access to BOM objects and properties. Statements such as *ss.name = "office"* are logical settings filtering what object instances are selected. The curly brackets are used to group statements under specific conditions. In the case of a rule definition, *ObjectType* will be *Rule*, and the given arguments *args* can be expressed in the parenthesis. This is a basic format of invoking BERA instructions; however, BERA language syntax also allows some shorter forms of instantiation for users. In general, the BERA language syntax follows the existing style of object-oriented languages such as the dialects of the Java programming language. Detailed lexicon and syntactic rules of the BERA language are described by Lee (2011).

```
ObjectType name(args)
{
    Statement;
    Statement:
     ...
}
```

FIGURE 3.32 Basic statements format using BERA language.

```
 Space myOfficeSpaces(Space ss)
{
     ss.name = "office";
     Area = ss.Floor.area;
}
```

FIGURE 3.33 Example of a BERA code segment.

BIMQL

BIMQL is an open query language for building information models (Mazairac and Beetz, 2013). It is a query language that is intended for selecting, updating, and deleting data stored in IFC data models.

BIMQL simplifies the access to the interconnected structures of the IFC data models (e.g., the aggregation building elements through the *project-site-building-story-building* component). Objects in IFC data are often normally linked to each other through a complex network of relationships. For example, an *IfcWindow* entity is not directly related to an *IfcWall* entity; instead, three other intermediate objects are necessary to make the connection to them (Figure 3.34).

BIMQL seeks to make the language more user friendly for practitioners. Thus, a syntax close to the natural language was aimed at. As an example, instead of using lexically correct IFC entity names such as *IfcWallStandardCase*, *ifcColumn*, and *IfcDoor*, BIMQL enables users to use natural language terms such as *Wall*, *Column*, *Walls*, and *Door*. It utilizes terms and uses look-up dictionaries and structured vocabularies indicated in the International Framework for Dictionaries (IFD), by which localizations of the query language can also be attained.

BIMQL uses the Model Driven Architecture (MDA) approach to software engineering. MDA is one of the architectural cornerstones of the bimserver.org framework. It is based on the concept that instead of developing the source code itself, the programmer develops a model that is used for automatically generating the source code. This approach normally increases transportability, efficiency, and cross-platform interoperability. In this approach, EXPRESS schema is first converted to an Eclipse Modeling Framework (EMF) model. The EMF is a software-modeling

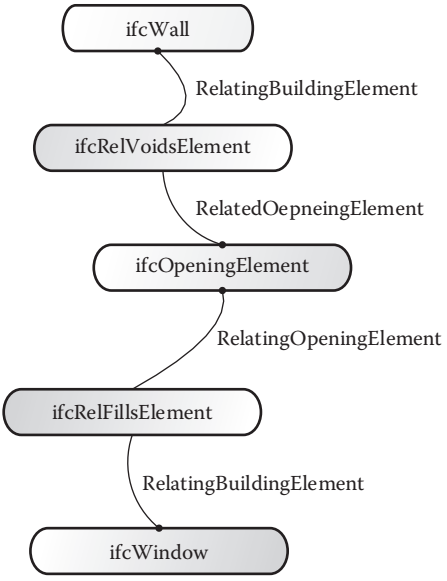


FIGURE 3.34 Relationship between IFC window and wall objects.

framework and code generation platform for developing applications based on a structured data model. It offers tools and runtime backing to generate a set of Java classes for the model, along with a set of other adapter classes that enable the displaying and editing of the model.

In the BIMQL project, first the EXPRESS schema is converted into EMF, then Java classes are generated for communicating with the bimserver.org database. Furthermore, the bimserver.org Java classes, which are based on the EMF Ecore model, are used to generate additional BIMQL Java classes. An outline of the system architecture of the BIMQL and bimserver.org framework is depicted in Figure 3.35.

The main components of the BIMQL include the following:

1. *EMF model*: EMF is utilized to generate a domain model from a concise description of the problem domain. The approach is based on the concept that developers focus only on high-level design, delegating tedious tasks to tools and frameworks to develop a complete software solution. At the core of EMF is the rich abstraction mechanisms for describing, composing, and manipulating structured information. The core of EMF includes a metamodel known as Ecore for describing models and providing runtime support for the models, such as notifications, persistence support with default XMI serialization, and a very efficient reflective API. For example, the Ecore model (Figure 3.36) depicts information about classes that are related to the data types and possible relationships, both described by the

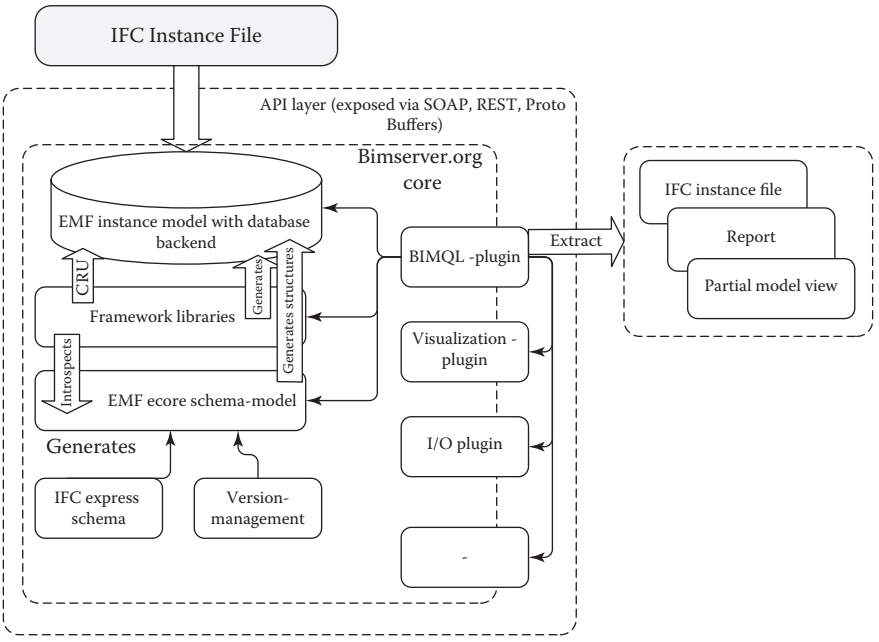


FIGURE 3.35 Overview of the BIMQL infrastructure. (Modified from Mazairac, W. and Beetz, J., *Advanced Engineering Informatics*, 27, 444–456, 2013.)

```
1 <eClassifiers xsi:type= “ecore:Eclass” name= “IfcDoor” eSuperTypes = “#//IfcBukldingElement”>
2       <eStructuralFeaturesxsi:type= “ecore:EAttribute” name= “OverallHeight” />
3       <eStructuralFeaturesxsi:type= “ecore:EAttribute” name= “OverallHeightAsString” />
4       <eStructuralFeaturesxsi:type= “ecore:EAttribute” name= “OverallWidth” />
5       <eStructuralFeaturesxsi:type= “ecore:EAttribute” name= “OverallWidthAsString” />
6 </eClassifiers>
```

FIGURE 3.36 EMF mode definition for the *IfcDoor* object shown in Figure 3.31. (From Mazairac, W. and Beetz, J., *Advanced Engineering Informatics*, 27, 444–456, 2013.)

```
1       ENTITY IfcDoor
2               SUPERTYPE OF (IfcDoorStandardCase);
3               SUBTYPE OF (IfcBuildingElement);
4               OverallHeight: OPTIONAL IfcPositiveLengthMeasure;
5               OverallWidth:  OPTIONAL IfcPositiveLenghtMeasure;
6       END_ENTITY
```

FIGURE 3.37 Express schema for *IfcDoor* object.

EXPRESS schema shown in Figure 3.37. The second important part of EMF is the code generation facility known as Codegen. This is capable of generating everything required to construct a complete editor for an EMF model. It includes a graphical user interface (GUI) from which generation options can be specified and generators can be activated. The Codegen model normally contains additional information for generating code, which in BIMQL are the bimserver.org Java classes (Mazairac and Beetz, 2013).

2. *BimServer.org Java classes*: These classes are mainly used to save BIM models to the database and manipulate objects already stored in the database. Each class contains *getters* and *setters*, which are methods used to define variables. For example, Figure 3.38 illustrates part of the bimserver.org Java class *IfcDoorImpl* and one of its getter's methods. The BIMQL Java classes are based on such methods.
3. *BIMQL Java classes*: These classes are generated from the bimserver.org Java classes. They are indirectly related to the IFC data model and establish a connection between the developed query language and the bimserver.org Java classes. It communicates to the database in which a BIM mode is residing. For instance, Figure 3.39 displays part of a BIMQL Java class, which is derived from the class shown in Figure 3.38.
4. *Grammar plug-ins*: BIMQL has an extendable mechanism to allow the execution of arbitrary grammar plug-ins through a plug-in interface. The colon

```
1  public class IfcDoorImpl extends IfcBuildingElementImpl implements IfcDoor {
2        public double getOverallHeight () {
3        return (Double) eGet(Ifc2x3Package.Literal.IFC_DOOR_OVERALL_HEIGHT, true);
4        }
5 }
```

FIGURE 3.38 Fragment of the bimserver.org *IfcDoor* Java class. (From Mazairac, W. and Beetz, J., *Advanced Engineering Informatics*, 27, 444–456, 2013.)

```
1  public class SetAttributeSubIfcDoor {
2        public void setAttribute () {
3        …..
4          else if  (attributeName.equals("overallHeight")) {
5                ((IfcDoor) object.setOverallHeight(Double.parseDouble(attributeNewValue)));
6          }
7        }
8 }
```

FIGURE 3.39 Partial listing of the BIMQL *IfcDoor* Java Class. (From Mazairac, W. and Beetz, J., *Advanced Engineering Informatics*, 27, 444–456, 2013.)

(:) character serves as a main token in the *plug-in rule* that marks an arbitrary variable keyword. After the initialization of the BIMQL engine prototype, plug-ins can be loaded into the engine. Plug-ins must implement a *BIMQLGrammarPluginInterface* interface class. The colon token indicates that all registered plug-ins are queried for the variable keyword following the colon. If present, the specific evaluation method of the appropriate plug-in is invoked. This will allow arbitrary amounts of additional keywords to be plugged into the language, making it extensible and open to the industry. Furthermore, specific keywords can be localized without difficulty to add natural language incorporation.

An example of BIMQL is depicted in Figure 3.40. The query shown returns all spaces whose area is larger than 20 m^2. The *property* operator provides a great shortcut to retrieving the value of a property. In this example, it will match instances of the *IfcSpace* object that have a *GrossFloorArea* property allocated to them by an *IfcElementQuantity* property set (Mazairac and Beetz, 2013).

Another BIMQL example is illustrated in Figure 3.41. The query returns all doors that are too small to provide access to an operating room. Lines 1 and 2 in

```
1 Select      ?MySpace
2 Where       ?MySpace.EntityType =IfcSpace
3 And         ?MySpace.Property.GrossFloorArea > 30
```

FIGURE 3.40 Example of BIMQL statements to select certain spaces.

```
1 Select  ?OperatingRoom
2 Where ?OperatingRoom .EntityType =IfcSpace And ?OperatingRoom .Attribute.Name=OR*
3 Select  ?OperatingRoomSpaceBoundary := ?OperatingRoom.Attribute.BoundedBy
4 Select  ?OperatingRoomSmallDoor :=
5    ?OperatingRoomSpaceBoundary.Attribute.RelatedBuildingElement
6 Where ?OperatingRoomSmallDoor.EntityType = IfcDoor
7 And ?OperatingRoomSmallDoor.Attribute.OverallHeight < 200
```

FIGURE 3.41 More Advanced BIMQL query statements selecting all doors with height less than 2 m.

Figure 3.41 select all operating rooms. Following that, in lines 3 and 4, the query statements select all building objects that define the operating room space. The last line limits that selection to the doors with a height less than 200 cm.

REFERENCES

AIA. (2013). Building Information Modeling and digital data exhibit. Retrieved from http://www.aia.org/aiaucmp/groups/aia/documents/pdf/aiab099084.pdf. AIA. Project Building Information Modeling protocol form. Retrieved from http://www.aia.org/aiaucmp/groups/aia/documents/pdf/aiab099086.pdf.

BIMForum (2015). Level of Development specifications, version 2015. Retrieved from https://bimforum.org/lod.

BIMXML.org. (2016): Website for Building Information Model Extended Markup Language (BIMXML). http://bimxml.org (accessed June, 2016).

buildingSMART alliance® (2015). National BIM Standard-United States® (NBIMS-US™) Version 3 (V3). https://www.nationalbimstandard.org/buildingSMART-alliance-Releases-NBIMS-US-Version-3.

buildingSMART International. (2015). Model Support Group of buildingSMART International. http://www.buildingsmart-tech.org/about-us/msg.

buildingSMART International (2012). http://www.buildingsmart-tech.org (accessed June, 2016).

GSA (2009). BIM guide for energy performance. GSA. https://www.gsa.gov/cdnstatic/GSA_BIM_Guide_Series_%281%29.pdf (accessed 12/20/2017).

IAI (1999a). *Specification Development Guide*. Eds J. Wix and R. See. IAI Specification Task Force, Oakton, VA.

IAI (1999b). *IFC Object Model Architecture Guide*. Eds T. Liebich and R. See. IAI Specification Task Force, Oakton, VA.

IAI (2000). *IFC Technical Guide – Enabling Interoperability in the AEC/FM Industry.* Ed. Liebich, T. and Wix, J. Modeling Support Group. International Alliance Of Interoperability (IAI), 46.

IECC (2006). *MDV for the International Energy Conservation Code*. http://www.blisproject.org/IAI-MVD/.

IAI (2000). *IFC Technical Guide: Enabling Interoperability in the AEC/FM Industry.*

ISO 29481-12010. Building Information Modeling: Information delivery manual; Part 1: Methodology and format. http://www.iso.org/iso/search.htm?qt=29481&sort=rel&type=simple&published=on (accessed September 29, 2010).

Lee, J. K. (2011). Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program. Ph.D. Thesis, Georgia Institute of Technology.

Lee, J. K., Eastman, C. M., and Lee, Y.C. (2015). Implementation of a BIM domain-specific language for the Building Environment Rule and Analysis. *J Intell Robot Syst*, 79, 507–522.

Liebich, T. and Wix, J. (2006). International Energy Conservation Code. Modeling Support Group, International Alliance of Interoperability (IAI), International Code Council (ICC), Munich, Germany.

Liebich, T. and Matthias, W. (2012). *ifcXML4 Specification Methodology*. Developed by Model Support Group (MSG) of buildingSMART International Ltd.

Mazairac, W. and Beetz, J. (2013). BIMQL: An open query language for building information models. *Advanced Engineering Informatics*, 27, 444–456.

Nawari, N.O. (2011). Standardization of structural BIM. *Proceedings of the 2011 ASCE International Workshop on Computing in Civil Engineering*, June 19–22, 2011, Miami, FL, pp. 405–412.

Nawari, N.O. (2012a). BIM standard in offsite construction. *Journal of Architectural Engineering (JAE)*, 18(2), June 1.

Nawari, N. (2012b). Automating codes conformance. *Journal of Architectural Engineering*, 18(4), 315–323. Retrieved from http://ascelibrary.org/doi/abs/10.1061/(ASCE)AE.1943-5568.0000049.

Nawari, N.O. (2014). BIM standard: Tensile structures data modeling. *Proceedings of the 15th International Conference on Computing in Civil and Building Engineering*, Orlando, FL, June 22–25.

Nawari, N.O. and Alsaffar, A. (2015). Understanding computable building codes. *Journal of Civil Engineering and Architecture*, 3(6), 163–172, San Jose, CA.

Nawari, N.O. and Kuenstle, M. (2015). *Building Information Modeling (BIM): A Framework for Structural Design*. Boca Raton, FL: CRC Press. ISBN-13: 9781482240436.

Onuma. (2011). BIMXML primer. http://www.Onuma.com, June 1, 2011.

Smith, B. and Bordenaro, M. (2011). BIMXML: Stepping forward onto proven ground. *Journal of Building Information Modeling (JBIM)*, Fall.

USACE (2014). Modeling Matrix (M3), US Army Corps of Engineers.

4 Automated Rule-Based Checking Systems

INTRODUCTION

Currently, there are a number of software platforms that have been developed to support the implementation aspects of automatic code provision–checking systems. They vary generally in their capability of automating the design-checking process, their flexibility in modeling design information, their flexibility in encoding building codes and domain knowledge, their capability of providing friendly reporting systems and 3D visualization, and their ability to integrate with other applications. Some of these systems are considered *black-box* methods, in which users have very limited access to the rule creation engine; others are *gray-box* or *white-box* methods, where users have varying degrees of customizations and interaction (Figure 4.1). The most commonly used platforms as well as recently emerging systems are described in the following sections.

CORENET SYSTEM

The term CORENET refers to the Construction and Real Estate Network. It is an information technology (IT) initiative that was started in 1995 by Singapore's Ministry of National Development to advance the construction and real estate sector into the new millennium by reengineering business processes with advanced information systems to enhance turnaround time, productivity, and quality. It also targets various trades in the construction and real estate sector to collaborate and exchange information flawlessly and efficiently. It is being employed by Singapore's Building and Construction Authority in partnership with numerous other public and private agencies.

The main objective of CORENET is to create an information systems infrastructure that facilitates total interoperability across several processes of a building life cycle, such as design, procurement, construction, and maintenance. However, the systems began with the creation of three modules for rule-checking automation during the design phase. These are CORENET e-Submission, CORENET e-PlanCheck, and CORENET e-Info.

CORENET e-Submission is an internet-based system that allows AEC professionals to submit project-related construction documents to regulatory authorities for different types of approvals such as planning approvals, building plan approvals, architectural plan approvals, structural plan approvals, temporary occupation permits, and so on. It provides a one-stop shop for professionals to submit construction drawings to multiple approving authorities from anywhere and to check submission status on the internet (Figure 4.2). It aims to integrate application forms and

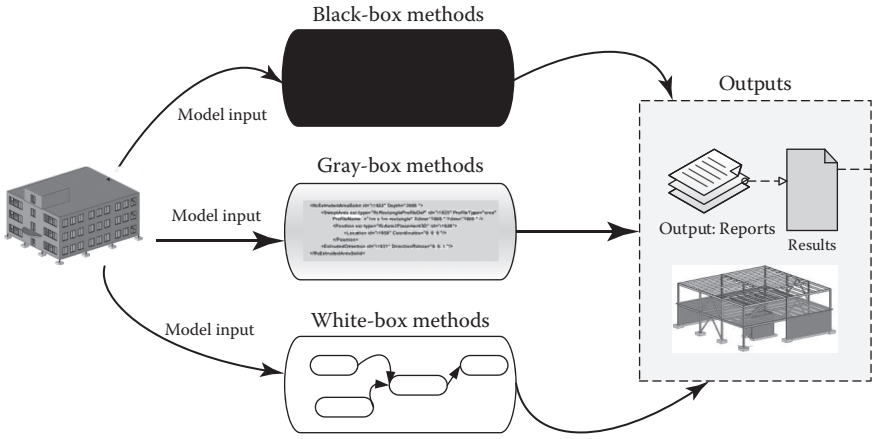


FIGURE 4.1 Illustration of the different systems of automated building code compliance checking.

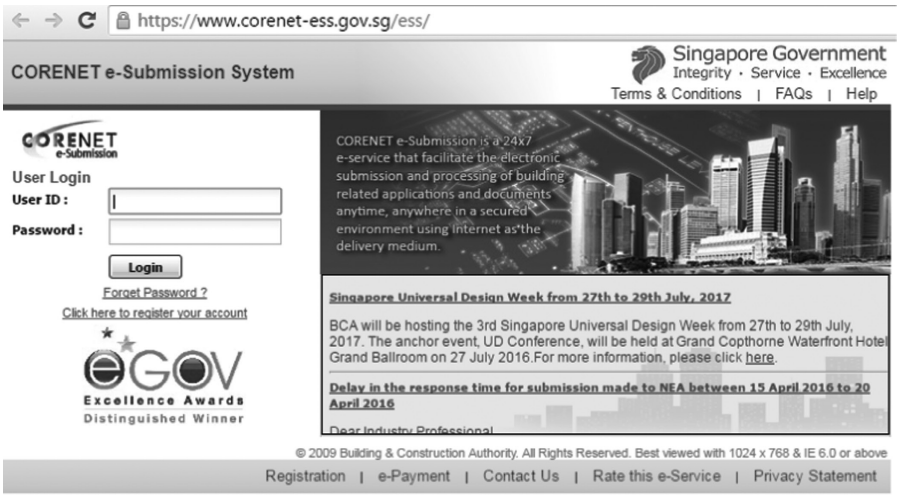


FIGURE 4.2 The CORENET e-submission login page.

fee collection as well as to simplify the work for approving authorities by providing them with the means to submit submissions statuses online. Overall, it results in faster processing and turnaround time, enhancing public services through better effectiveness and productivity in handling and processing digital construction document submissions. It also has the extra advantage of harmonizing rules and rationalization forms across the various agencies involved.

CORENET e-PlanCheck was proposed to provide a series of information system applications that could automatically check electronic building plans for compliance to regulatory requirements using artificial intelligence (AI) and

feature-based building information modeling (BIM) technologies. e-PlanCheck, which formally began in September 2000, centers on the building model rather than on 2D construction document representations. e-PlanCheck presently includes code checking for specific aspects of architecture and construction services and will ultimately be extended to cover structure and other areas as well. The main technology in the implementation of e-PlanCheck is the Industry Foundation Classes (IFC) building model. However, the IFC schema by itself is not adequate in implementing a code conformance–auditing application. This is due to the fact that the IFC only denotes the basic building information that can be modeled by a BIM-authoring tool during the design phase. What is required are higher-level semantics of building elements, and in the case of the e-PlanCheck system, these are provided by an independent platform known as FORNAX, developed by novaCITYNETS, an e-government solution provider in Singapore. FORNAX is a C++ object library that derives new data and creates extended views of IFC data. FORNAX accepts the main building model information from the IFC and augments it with higher-level semantics that are pertinent to code conformance–checking requirements. This is generally performed by capturing building elements into a set of FORNAX objects, each of which defines related attributes and behaviors. These objects are designed to be extendable for customization to deal with different conditions and formats of building codes. FORNAX basically offers a development and implementation platform for e-PlanCheck, and the building codes are understood and constructed utilizing FORNAX as an add-on to the platform (Figure 4.3).

An example of how FORNAX objects are utilized in code conformance verification is described here using a *flat unit*, a collection of spaces typically found in an apartment dwelling (Khemlani, 2005, 2007). Referring to the IFC format, the flat unit is simply characterized as a collection of spaces, making it cumbersome, for example, to verify the compliance of codes related to fire safety requirements, such as the circulation distance of any point in the apartment unit to the nearest exit, or the area and volume of the unit to verify the restriction of fire compartmentalization. When utilizing the FORNAX platform, a FORNAX object *FXApartmentUnit* is invoked for a facility which includes various methods to perform computation and define parametric properties. Some of the methods that are used to determine fire safety requirements are

GetSpaces: Retrieve all spaces constituting this facility.
GetExit: Retrieve the exit door of this facility.
CalculateRemotePoint: Compute the remote point in a space from its door(s).
CalculateTravelDistance: Compute the travel distance between the given point in the space and the nearest exit door.
CalculateArea: Compute the area of this facility.
CalculateVolume: Compute the volume of this facility.

The following steps demonstrate how the circulation distance computation for a flat unit, utilizing the approach listed previously, would be executed (Figure 4.4).

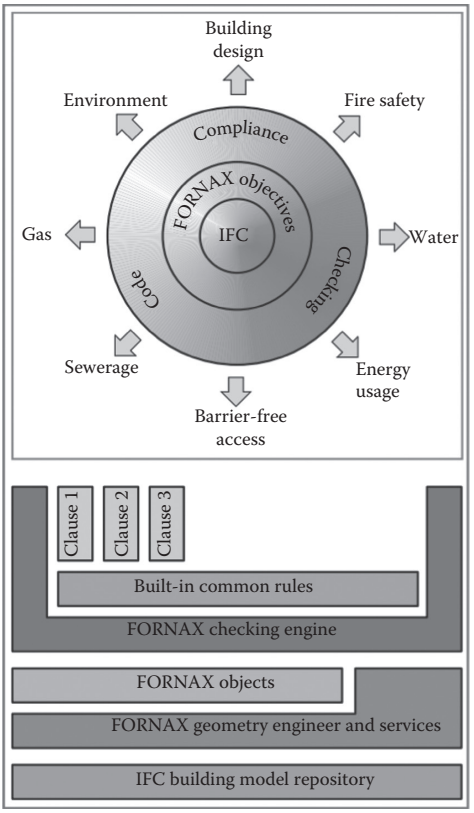


FIGURE 4.3 The FORNAX platform captures higher-level semantics of building components for code checking and provides the development and deployment environment for the rules capturing building codes. (Modified from Khemlani, L., CORENET e-PlanCheck: Singapore's automated code-checking system, http://www.aecbytes.com/buildingthefuture/2005/CORENETePlanCheck.html, accessed June 2009, 2005.)

Get all the spaces forthis flat unit (GetSpaces).

For each space:

- Calculate the remote point in this space from its door(s) (CalculateRemotePoint).
- Check the travel distance of the calculated remotepoint (CalculateTravelDistance) to the main exit (GetExit) of the apartment unit.

FIGURE 4.4 An example of steps for circulation computation using FORNAX method. (Modified from Khemlani, L., CORENET e-PlanCheck: Singapore's automated code-checking system, http://www.aecbytes.com/buildingthefuture/2005/CORENETePlanCheck.htm, accessed June 2009, 2005.)

A visual representation of this travel distance calculation as performed in FORNAX is shown in Figure 4.5.

In summary, FORNAX is a C++ object library that derives new data and generates extended views of IFC data to assist in the automated checking of building regulations. FORNAX objects provide rules for assessing themselves, providing good object-based modularity to perform automated checks on building models against building regulations.

SOLIBRI MODEL CHECKER

Solibri Model Checker (SMC) is a Java-based stand-alone platform application that reads an IFC model and maps it to an internal object structure that facilitates access to the model data and compliance processing. It includes a variety of built-in methods, such as a library for pre-checking a model for basic geometric properties and attributes errors, as well as object existence, fire code exits, path distance checking, space program checking against the actual spaces in a building, and others (Figure 4.6).

Rules can be parametrically varied through table-set control parameters. However, entirely new rules can be added in Java using the SMC application programming interface (API). SMC (Corke, 2013) currently includes a set of 300 preformatted rules that allow for some degree of user customization. However, such customization does not allow for the creation of new rules. The development of new rules in SMC requires professional Java software engineering expertise and a good understanding of the SMC software's environment and data structure (Corke, 2013).

SMC is basically a rules-based checking and auditing tool for BIM data (Figure 4.7). It is designed to improve the quality of the BIM model and the quality of the information

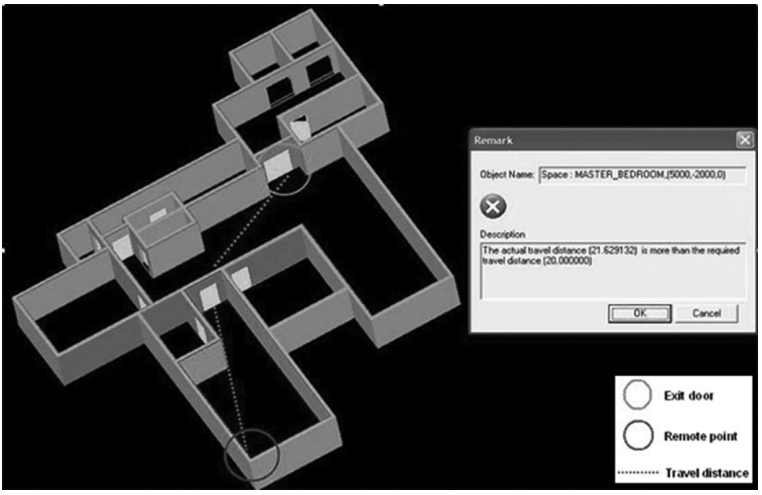


FIGURE 4.5 An example depicting travel circulation computation using FORNAX. (Modified from Khemlani, L., CORENET e-PlanCheck: Singapore's automated code-checking system, http://www.aecbytes.com/buildingthefuture/2005/CORENETePlanCheck.html, accessed June 2009, 2005.)

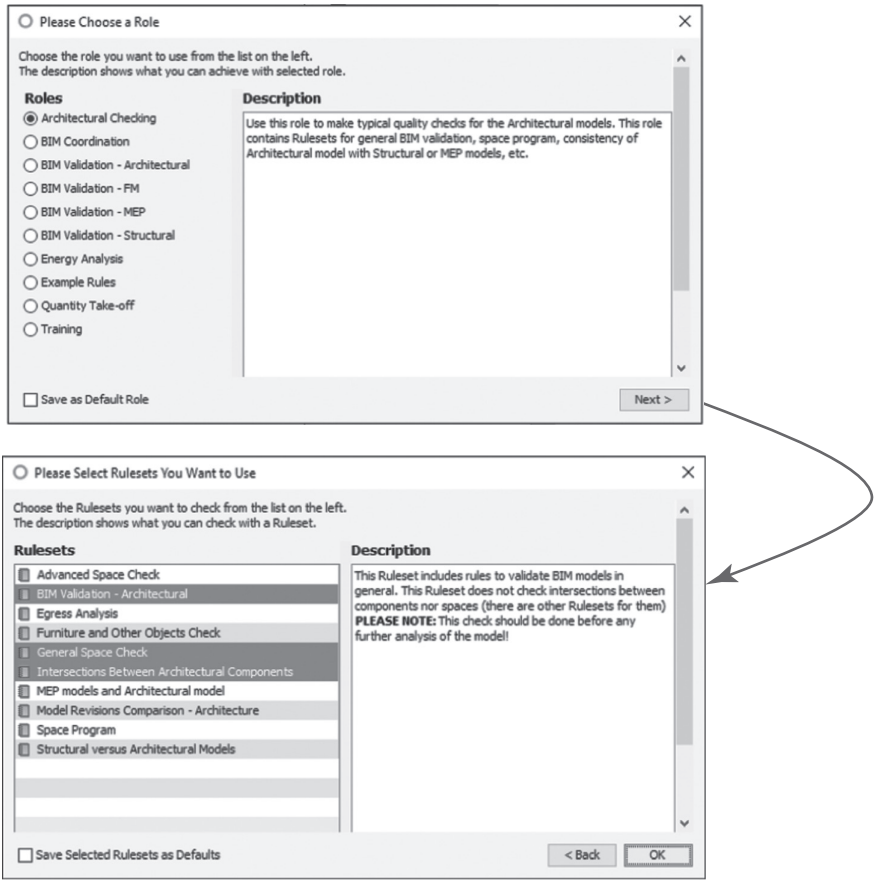


FIGURE 4.6 Rule set selection in SMC application.

within it. Depending on how it is applied, it can even be used to improve the quality of the IFC data generated.

SMC offers a variety of means for reporting checking issues that include PDF, XML, and XLS formats, as well as proprietary SMC visualization and reporting formats suitable for design reviews using the free Solibri Model Viewer.

SMC only accepts models in IFC format. Thus, for the BIM model to be examined against code regulations, it has to be in IFC format. When opening an IFC model in SMC, the first step is to select a discipline—architect, MEP, or structural engineer, for instance. This assists when collaborating with other disciples, but also helps track responsibility for any changes that need to be made as a result of the checking process.

Multiple IFC models can be combined for coordination and saved as a single SMC file, the native format of SMC. In general, the resulting SMC data file is much more compressed than the original IFC files as SMC tries to optimize the import of data for improving visualization and information presentation.

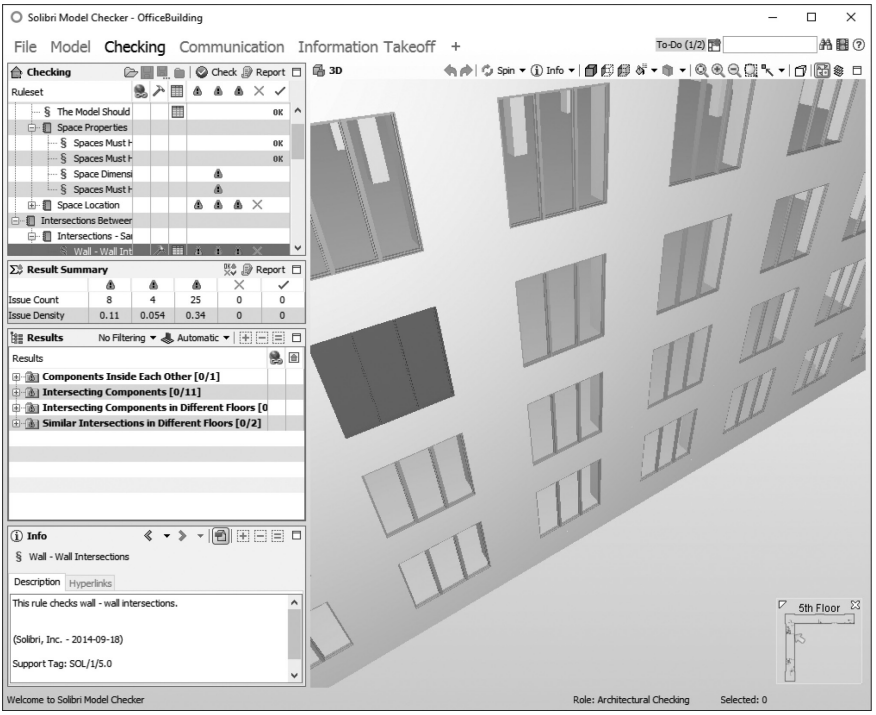


FIGURE 4.7 SMC user interface depicting code-checking results.

Once the BIM model is opened in SMC, the IFC file can be browsed from the model tree, which is located on the left-hand side of the user interface (UI). Users can drill down by level, IFC model type, family name, and item (Figure 4.7). The model can also be browsed directly by IFC component or layer if the original BIM-authoring tool supports these features. Double clicking on any IFC component zooms to its position within the model, which is displayed to the right of the UI.

SMC software presents a real-time rendered view of the building and includes all the standard model manipulation tools—panning, zooming, rotating, walkthroughs, and so on. Examining clash detections between various building elements can be toggled on and off so users can walk through the building using stairs rather than going through walls. Sections can be taken in real time to reveal internal details. Any object or group of building objects can be highlighted or hidden from view as needed. As the model provides a direct link to the underlying attribute data, selecting any component reveals all related information, including location, quantity, area, height, and type. The model can also be displayed thematically according to the attributes of its components.

In terms of verifying the designed model, SMC can literally check anything in the model—for example, checking that all rooms have openings, that there are no duplicate components (e.g., an architectural wall on top of a structural wall), that there are no gaps between building elements (e.g., walls and floors), that a structural column goes through a slab, or that spaces are properly allocated within the model (Figure 4.8).

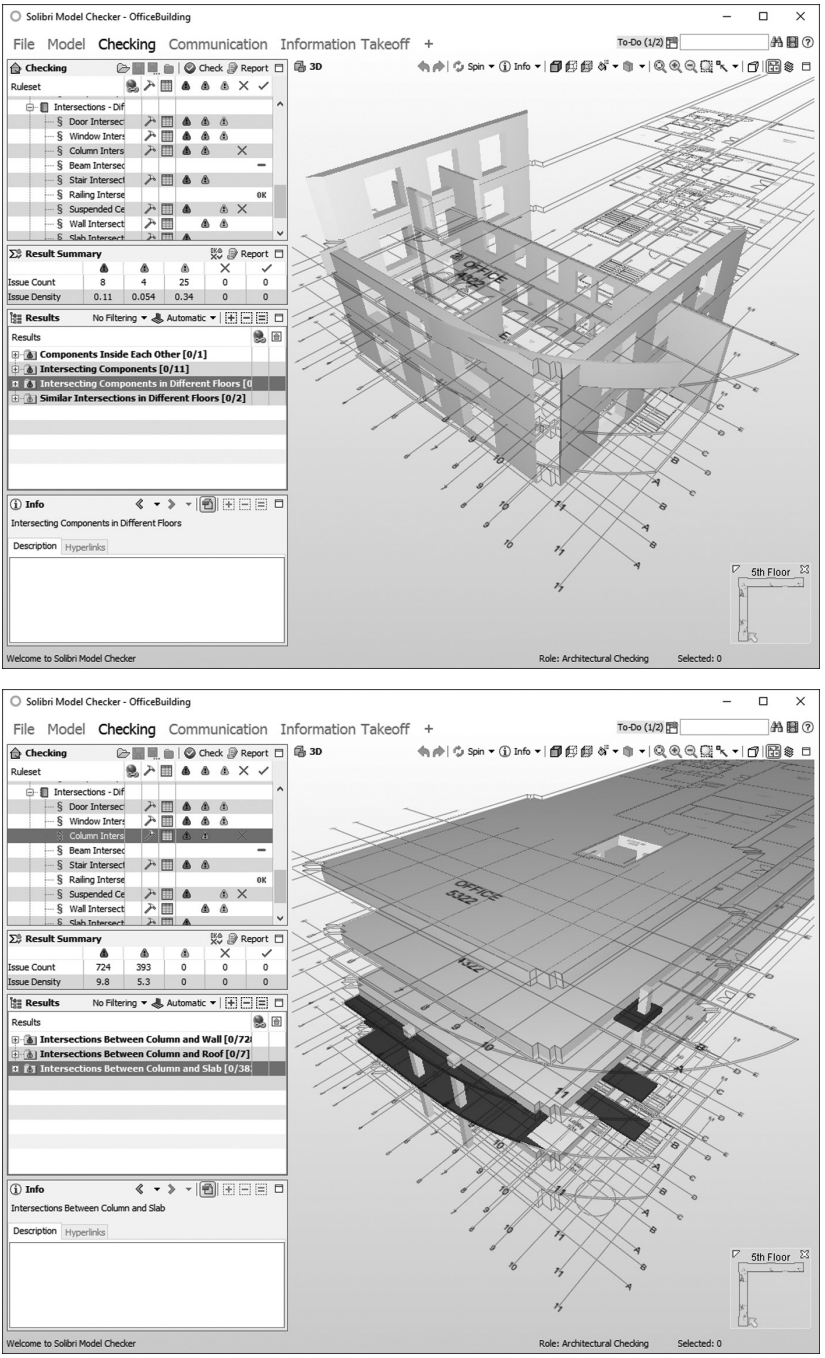


FIGURE 4.8 (a) SMC user interface depicting code-checking results: regulation failure, intersections of various building elements, (b) SMC user interface depicting code-checking results: regulation failure, structural column goes through a slab.

In addition to the existing rule sets, SMC allows users to request new rules throughout their subscription (normally based on consultancy fees) using the Ruleset Manager (Figure 4.9), in order to develop rule sets that are closely aligned with the client's needs.

To begin the checking process in SMC, users simply need to click on the "CHECK" button (Figure 4.8). The system then runs the chosen rule sets through the model, presenting each issue and the description of each failure in a navigable tree table. Double clicking on any issue listed takes the user to its location in the 3D model, highlighting the component(s) in question together with a plan view of its associated floor to add some context. Users have the choice to accept/reject, add annotations and comments, or assign responsibility to other members of the project. An audit report can be generated that details all of the errors and issues that require review. The report can be in any conventional format, such as PDF or Excel spreadsheet.

The next step is to fix any problems that have been identified in the report. Issues are usually resolved in the original BIM-authoring tool by the relevant consultant/contributor. However, this workflow also has the challenge of locating the reported components from SMC in the BIM-authoring tool. Quite often, this is performed manually. Because of the difficulties of manual cross-referencing, SMC suggests exporting the results to the BIM Collaboration Format (BCF). Essentially, BCF introduces a workflow communication capability connected to IFC models. The idea is to separate the "communication" from the actual model. BCF is based on XML schema from buildingSMART and is specifically designed to allow an object or object view in one software package to be identified and located automatically in another software package. It is important to note that the BCF format is independent of the IFC schema version being used. It seems that BCF is a promising format to deal with such issues, but it still has a long way to go as a universal format for IFC model-checking

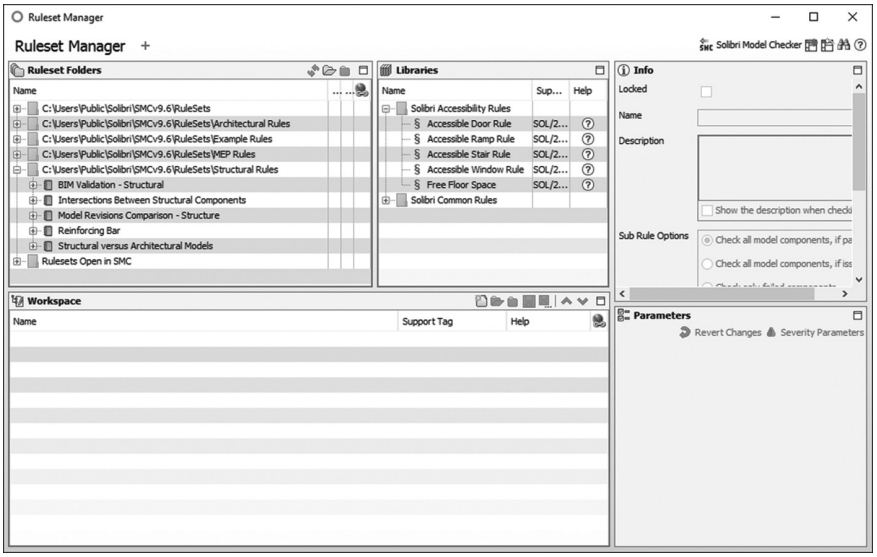


FIGURE 4.9 SMC Ruleset Manager.

intercommunication. Furthermore, SMC offers tools to compare two versions of the same model. This can be useful to identify what has been added, what has been deleted, and what has been changed. This can be beneficial to confirming that the latest revision is being utilized and to understanding the changes being undertaken.

JOTNE EDMMODELCHECKER

The Express Data Manager (EDM) application was developed by Jotne EPM Technology in Norway in 1998 as an object database with tools to manage complex product data models. It began as a collaboration tool but has since incorporated several additional modules, including EDMmodelChecker, which supports open development using the EXPRESS modeling language (ISO 10303-11). EDMmodelChecker can be also utilized to validate a data set and ensures that it conforms to all rules and constraints defined in one or more EXPRESS schemata.

EDM offers functionality for interoperability domains such as data interoperability solutions that resolve business issues such as data exchange, data sharing, data integration, and data archival. It provides an object database and supports the open development of rule checking using the EXPRESS language, which is the language in which the IFC model schema is written. New model views can be developed using EXPRESS and EXPRESS-X, which is a language for mapping instance data from one EXPRESS schema to another and supports extensive queries and reports. All EDM functionality, including mappings, queries, and rules, can be accessed via the C, C++, Java, and .NET developing platforms (EDM, 2016).

EDM can be also used to create data translators/converters from one data format to another one, where one of them may be, but does not need to be, an international standard, such as ISO 10303 STEP. These systems make EDM open to advanced user extensions. Jotne EPM also offers reporting engines for textual reports and other related services.

NORWEGIAN STATSBYGG'S DESIGN RULE–CHECKING EFFORTS

The CORENET system was modified and adopted in Norway with the ByggSok system (Haraldsen et al., 2004). This is an e-government system encompassing three modules: an information system, a system for the e-submission of building applications, and a system for zoning proposals. It is driven by the Norwegian building and construction industry and supported by Standards Norway and the Norwegian buildingSMART. It is mainly based on the IFC data format standard.

Also, based on CORENET e-PlanCheck pilot projects, Norwegian developers Statsbygg have experimented with multiple systems as part of their efforts to extend the use of IFC to the entire project life cycle in support of the Norwegian mandate that all properties will use IFC-based BIM by 2010 (Sjøgren, 2007). The resulting systems have been tested on real projects, with data being exchanged through a wide selection of software to conform to the various phases of the project life cycle. The test project used for the code compliance verification efforts focused predominately on circulation design. In this project, the building model data are stored and accessed through the EDM model server using the IFC data format. The accessibility rules are parameterized, mapped to

their related building objects, and executed using SMC Ruleset Manager. SMC communicates directly with building model data in IFC format but retrieves only the objects it needs—that is, those mapped to the circulation rules. The rules implemented to date focus predominantly on geometrical constraints defined by various objects and parameters. The Statsbygg Solibri system does not support the enhancement of these data models or exporting to IFC format. Thus, the system can currently only be used for the regulation compliance auditing of attributes supported by the original BIM-authoring tool. New rules, however, must be custom-made in collaboration with the SMC software developers and as such are not easily adapted for other software.

REGION-SPECIFIC BUILDING REGULATIONS COMPLIANCE–CHECKING SYSTEMS

International Code Council

The International Code Council (ICC) is the organization responsible for issuing the master building code used by authorities in North America. It issues codes for residential and commercial buildings and most institutional buildings. In 2006, it began supporting the concept of developing SMARTcodes; the development effort was being carried out by AEC3 and Digital Alchemy. SMARTcodes were supposed to provide the mapping of written building codes into interoperable computer-interpretable code instructions. The project for creating SMARTcodes focused on systematizing and automating code compliance auditing against the ICC codes and federal, state, and locally accepted versions of the ICC. A proof of the SMARTcodes concept implementation was demonstrated in several venues in 2007/2008. The project lost funding due to the recession that began in 2008 (Digital Alchemy, 2016).

The concept of SMARTcodes had been demonstrated using the International Energy Conservation Code 2006 (AEC3, 2006). The SMARTcodes used an IECC dictionary for the definitions of terms for objects and properties that are critical for model checking. The idea was to offer SMARTcodes Builder, a Web-based software application to facilitate the creation of SMARTcodes. The software offers users the opportunity to identify the main expressions and their logical parts, then formalizes the expressions by means of using terms from the dictionary. Rule statements, regulations, and provisions are interpreted by the SMARTcodes developer utilizing the IECC dictionary that defines the characteristics connected with each term and the listing of properties, data types, and units connected with each property. The dictionary is used not only for rule interpretation but also for messaging between the SMARTcodes model verification system and the IFC building model (Figure 4.10). IFC object characteristics described in the dictionary offer model views for rule examination. The model views are classified by the terms and properties of the dictionary. The early version of SMARTcodes for SMC was developed by Digital Alchemy (DA) and AEC3 (AEC3, 2006). It supported rule-based building code compliance checking for IECC 2006. The software developed requires input values such as the building model, building location, and code to be checked. In its initial form, only selected portions of IECC 2006 are available for checking as well as a limited number of preconfigured test models that satisfy the modeling requirements needed

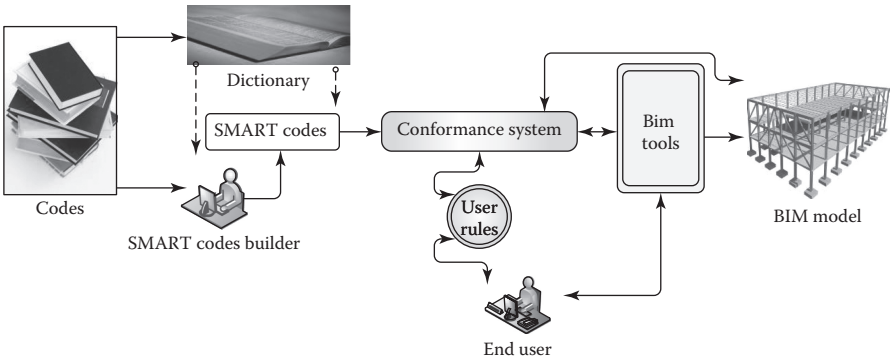


FIGURE 4.10 Concept of SMARTcodes in the framework of an automated model–checking system.

for checking. They contain a prototypical office building, the Lawrence Berkeley National Laboratory, the National Association of Realtors headquarters, and four building models provided by the US Coast Guard (Eastman, 2009).

Figure 4.10 depicts the general architecture of the SMARTcodes concept. The output reporting is supposed to utilize the reporting engine of the model-checking software (MCS). The software provides end user auditing results in various formats, such as HTML, PDF, RTF, XLS, and XML. Analysis results include tabular summaries and graphical reports. The text-based tabular outputs briefly report whether a building model violates the codes for compliance checking or not. Also, detailed descriptions of violated rules can be displayed graphically about elements of the building model. This includes location, property, and the geometric form of a building component in the browser along with the reason for noncompliance.

Recently, ICC efforts have aimed toward a collaborative project between ICC, SMC, and Fiatech together with other software developers to create AutoCodes. This is currently a prototype system that promises an integrated compliance-checking capability for US building model codes (Fiatech, 2011). The Fiatech AutoCodes Project initiated the transformation of the building code review process by leveraging digital data and technology to change the manual paper-dependent process into an automated design review process.

In the first phase of the Fiatech AutoCodes Project, the inconsistency in code reviews by different jurisdictions in the US was verified and documented. This phase of the project has successfully demonstrated the feasibility of conducting project reviews for access and egress in a building, based on a BIM model. Phase two of the AutoCodes Project set the framework for future Standards Development Organization (SDO) validation of the digital review process, producing model protocol guidelines (the Minimum Modeling Matrix or M3). It sets the workflow guidelines for the automatic code review process and expands digital transition education and training for state and local governments. The third phase of this project is supposed to expand on the work of the second phase by continuing to develop M3 in the areas of accessibility and egress, expand review capabilities to other codes and industries, and update and improve the delivery of the digital transition education

and training. Furthermore, in order to facilitate the use of BIM and AutoCodes for automated code auditing, the AutoCodes project has proposed a supplemental guideline (Table 4.1) for both model authors and reviewers, in the absence of a standard that specifies requirements on model content, information exchange, and information delivery.

US General Services Administration Design Rule Checking

The US General Services Administration (GSA) is one of the main drivers of BIM applications and the validation of BIM models. The GSA began issuing a series of BIM guidelines in 2007 (BIM Guide 01, 2007). The latest publication in the series is BIM Guide 07, "Building Elements" (GSA, 2016). The GSA requires that all their future facilities be delivered using BIM models that satisfy GSA BIM guidelines. The GSA recognizes that the information about its buildings is an asset of equivalent value to the buildings themselves. This awareness leads to the understanding that authorized users should have access to facility information from a single, accurate, reliable, and up-to-date information source. The GSA developed various sets of rules to automatically examine models for compliance with GSA requirements. Generally, the GSA utilizes SMC to automatically verify models for compliance with GSA guidelines. Some of these rules cover checking BIM models for the following (GSA, 2015): (1) the model contains a building with levels; (2) all building objects are contained on a level; (3) the model has spaces with specific properties; (4)

TABLE 4.1
Proposed Minimum Modeling Standards for AutoCodes

Relevant Sections in Existing BIM Standards/Guidelines	Recommended Content for AutoCodes Guidelines
Modeling requirements	• Codes that can and will be checked with models • Minimum modeling requirements for automated code checking (using M3) • New LOD required for existing properties • New properties required • Naming conventions required for automated code checking
Information exchange	• Information exchange standard for code checking (IFC and XML) • Model exchange portal for model-based code review
Delivery requirements	• Delivery requirements for code checking (formats and content)
BIM roles and responsibilities	• No additional roles or responsibilities required
Process planning	• Transition plan from paper to 3D model-based process • Communication plan for the new process
System requirements	• New hardware, software, services, and training for regulatory agencies • New hardware, software, services, and training for stakeholders
Data security	• Data security issues with the submitted models

Source: Fiatech, AutoCodes project team phase II report (Element 6: Project management), 2015.

model elements make sense dimensionally by meeting minimum and/or maximum dimensions; (5) the intersection of building objects, such as walls intersecting with other walls or a beam intersecting with a column, are sound; and (6) building objects are properly located; for example, doors and windows must be contained in walls.

Furthermore, the GSA has funded research to develop a rule-checking system for the circulation and security validation of US courthouses called Design Assessment Tool (DAT), which was developed by the Georgia Institute of Technology (Eastman et al., 2009). The basis for DAT is the circulation and security rules found in the US Courts Design Guide (CDG, 2007). The CDG defines the guidelines for federal courthouse design, founded on best practices obtained over the last 100 years (Figure 4.11).

The CDG was encoded into 302 computable parametric rules in DAT. Circulation rules are grouped into four high-level conditions: start space, intermediate space, destination space, and transition. Each space condition has a space name as well as a security level (public, restricted, or secure). Figure 4.12a depicts an example of rules associated with occupant circulation relating to connection between spaces: "Judges' chambers are accessed from restricted circulation with convenient access to the courtroom" (CDG, 2007). Figure 4.12b illustrates that route conditions can denote walking distance length, security level, vertical accessibility, and so on. These are defined in an SMC parametric table. A circulation rule can be described by setting up combinations of these parameters, which can then be utilized for automatic auditing using an SMC plug-in developed by Georgia Tech.

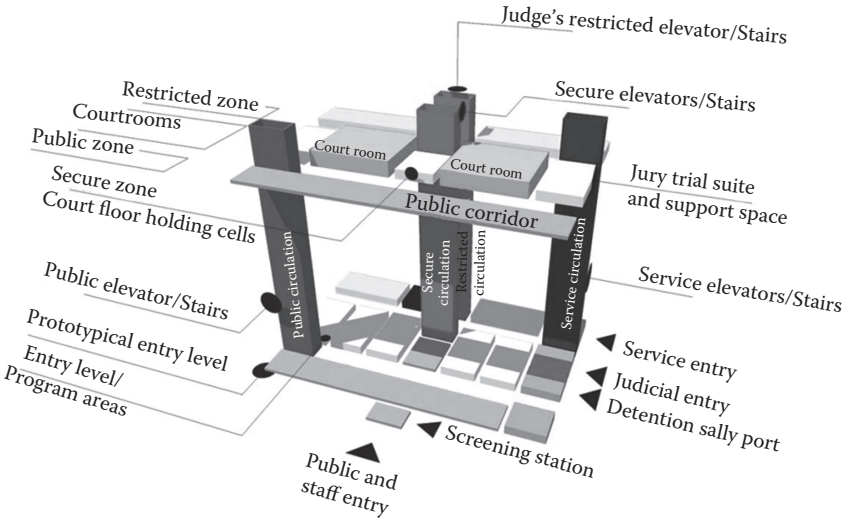
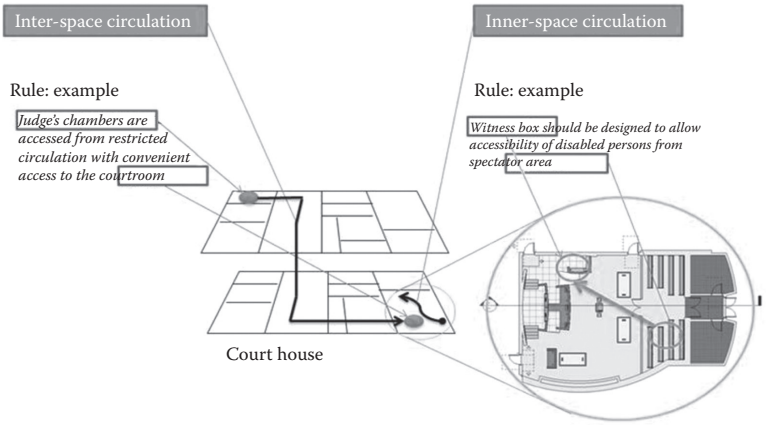
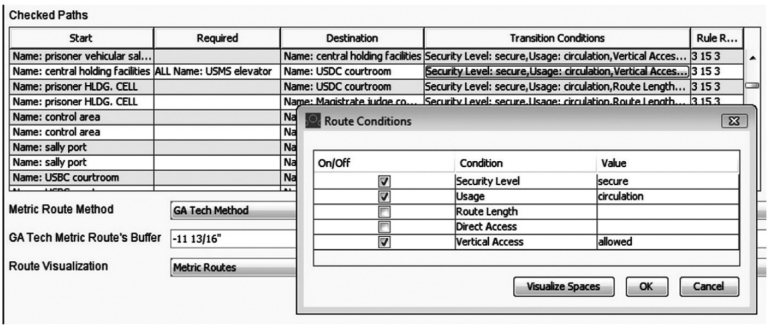


FIGURE 4.11 General courthouse zoning and circulation. (*Note*: This view is illustrative and not meant as a standard for design). (CDG, US Courts design guide, Administrative Office of the US Courts, Space and Facilities Division, http://www.gsa.gov/graphics/pbs/Courts_Design_Guide_07.pdf, 2007. Accessed September, 2015.)






```
IF
        Ssi = FindStartSpace (StartSpace)
        Tsj = FindTargetSpace (TargetSpace)
THEN
WHILE (Ssi is Exist)
FOR (Each Tsj)
        IsaValidRoute = False;
        AreAllRouteValid = True;
        WHILE (R is Exist)
        R = GetaRouteFromStartSpacetoTargetSpace(Ssi, Tsj );
        Result = Condition.Validation(R); -
        If(Result == True)
                IsValidRoute = True;
        If(Result == False)
                AreAllRouteValid = False;
        If(Result == True && Tcc= at least one && Rcc = at least one )
                Return True;
```

FIGURE 4.12 (a) Rules describing occupant circulation related to inter-space circulation vs. inner-space circulation. (From Lee, J., Building Environment Rule and Analysis (BERA) language, Ph.D. thesis, Georgia Institute of Technology, 2011.) (b) Parameters for describing circulation rules of CDG. (From Eastman, C., et al., Automatic rule-based checking of building designs, *Automation in Construction*, 18(8), 1011–1033, 2009.) (c) Example of the algorithm for checking a circulation rule using pseudo code. (From Lee, J., Building Environment Rule and Analysis (BERA) language, Ph.D. thesis, Georgia Institute of Technology, 2011.)

The application of the circulation requirements includes the derivation of a circulation graph of the entire building, identifying all spaces that can be occupied and their interconnectivity, as determined by walls, doors, stairs, ramps, elevators, and boundaries between directly adjacent spaces (without separating walls). In DAT, building model elements are automatically mapped to graph nodes and edges. Two types of graphs are used in this system (Figure 4.13). Namely, (1) a topological connection graph represents networks between spatial elements, which is used to check the routing paths defined in parameterized circulation rules; and (2) the metric graph denotes distances that people travel within a space; this type of graph is utilized to examine movement distances and visualize results (Lee, 2011).

Using graph structures, the circulation compliance–checking system can evaluate whether circulation paths between two spaces of the examined building model satisfy the CDG regulations. In addition to that, the system provides a cross-reference to the original rules in the specific section and page number of the CDG in natural language to end users, since the rule checking is carried out on the basis of interpreted parameters within SMC.

Noncompliance with circulation rules are displayed on the graph traversal structure as a route between starting space and destination space. For instance, if the BIM model does not satisfy a circulation rule, one violated route from all conceivable routes is visualized. The circulation rule states that the courtroom should be accessible from the judge's chambers through a restricted zone (Figure 4.14). Since the GSA's circulation validation–checking system was developed on the SMC platform, the other main functions for analysis reporting are comparable to other rule-checking systems built on the SMC platform.

Korean Research Efforts

Some of the efforts of South Korea in automating code compliance checking were focused on safety regulations of super-tall buildings. For instance, Kim et al. (2013) used SMC and its application programming interface (API) to develop rule set in checking regulations regarding the evacuation elevator and safety evacuation zone sector. Other efforts were focusing on studying the development of code compliance–checking system for building permits and other building administration processes, using an open BIM-based process (Choi, 2014). The study was based on

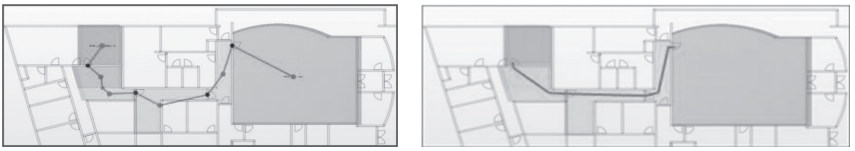


FIGURE 4.13 Illustration of space connection graph and metric graphs. (Modified from Lee, J., Building Environment Rule and Analysis (BERA) language, Ph.D. thesis, Georgia Institute of Technology, 2011.)

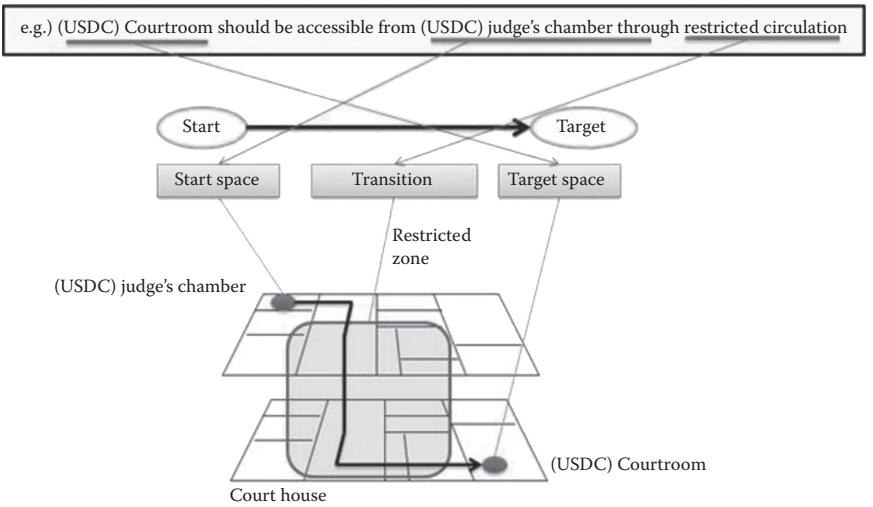


FIGURE 4.14 Circulation rule checking according to CDG. (Modified from Lee, J., Building Environment Rule and Analysis (BERA) language, Ph.D. thesis, Georgia Institute of Technology, 2011.)

IFC Pset (Property set content for IFC2x3 release). This includes the property set definitions (PSD) schema for IFC. The purpose of the PSD schema, is to provide an XML schema definition for describing properties and property sets outside of the IFC specification, but including information, such as applicable IFC entities or types (buildingSMART International, 2016). It is used to define the IFC defined property sets and properties, but is can also be used to:

- Define regional or project specific property sets and properties
- Define a mapping, for example using XSLT, between the PSD schema and another, application specific property definition schema, in order to translate the IFC defined property content into various applications

The PSD Release for IFC2x3 offers multilingual supports, that is Names and Descriptions of properties can be translated into other languages for example localizing the property content in various regions (buildingSMART International, 2016). This will allow also to improve the reliability and interoperability during information exchange. In addition, in the case where IFC structural objects and properties does not already contain in the Korean local building codes, this development provides a solution for including these properties.

Other Korean efforts were focused on transforming human-readable language sentences into a form of computer-readable building codes objects and properties of building regulations (Lee et al., 2015). It is a logic rule-based mechanism for translating natural language sentences in Korea Building Act into the executable and computer-readable format (Figure 4.15).

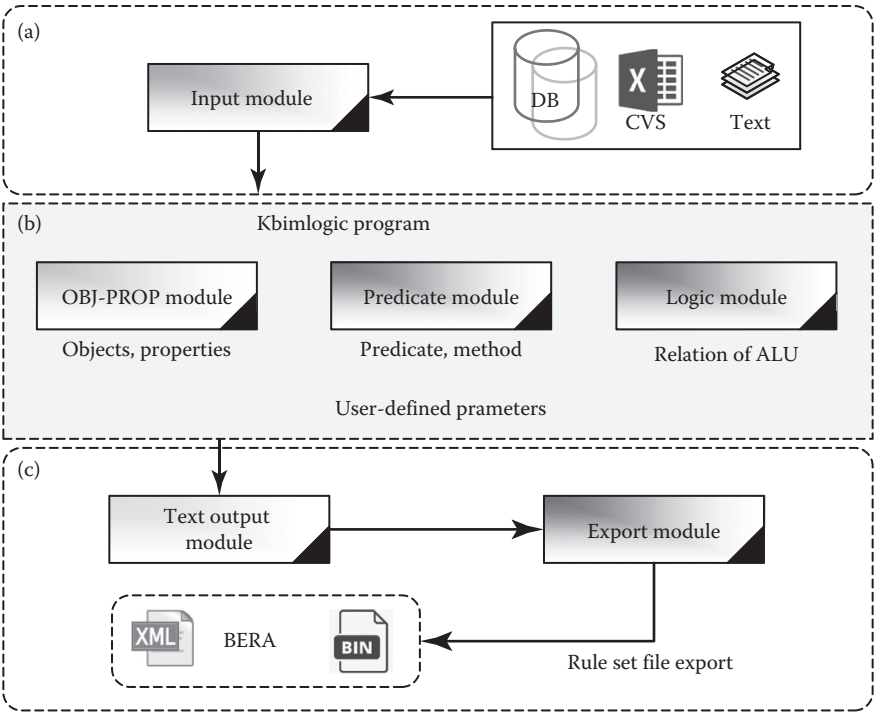


FIGURE 4.15 The concept of encoding Korean building codes: (a) human-readable language part (original sentences of Korea Building Act), (b) intermediate part between human-readable and computer-readable code, (c) computer-readable part (XML, binary code, BERA, etc.). (Modified from Lee et al., 2015.)

Choi (2014) derived entries based on the Korean building permission regulation checklist and developed building code compliance–checking system using Pset of the IFC. For example, in case of determining whether or not to use entries in the living room space, entries derived through the licensing checklist and additional attribute information were randomly checked by defining the Pset.

Further efforts were focused on developing a logical rule-based approach to the Korea Architecture Code text for BIM-enabled design compliance–checking systems (Lee et al., 2015). It aims to define a translation process of converting regulation text into specific computer-readable format with logical rule-based framework.

Australia's DesignCheck

DesignCheck is an automated code compliance–checking system developed by a research team of CRC for Construction Innovation, Australia (Ding et al., 2006). The DesignCheck system develops an object-based rule system using EDM (Express Data Manager™) for encoding design requirements from building codes. The EDM provides a shared data store that is compatible with the IFC. DesignCheck internal structure is based on IFC for modeling extended design information. It offers

flexibility by allowing a design to be checked by selected clauses or object types and support for checking various phases during the design process, such as at the conceptual stage of design, detailed stage of design and the construction phase of design.

The DesignCheck internal structures extends the IFC model to cover applications in specific domains, in this case the information required by building codes. For example, the Building Code Australia Part D Access and Egress (BCA D3) clauses involve checking building classes to define specific access requirements for disabled persons. A new type definition mapping onto building classes is defined in the DesignCheck internal structure as depicted in Figure 4.16 (Ding et al., 2006):

Entities in the DesignCheck internal engine are formatted to include new attributes mapping onto building codes and the properties transferred from IFC property sets that are associated with building codes. An example of the Door entity with new attributes in the DesignCheck internal engine is depicted in Figure 4.17.

The DesignCheck system is based on expressing building code regulations utilizing an object-based interpretation and then encodes them into the EDM rule bases. An illustration of the general process of transforming a building regulations clause into an object-based version and then to the EDM rules is show in in Figure 4.18.

The overall architecture of DesignCheck is depicted in Figure 4.19. The three main parts of the system include UI, EDM database engine, and a reporting engine.

In principle, DesignCheck is similar to that of e-PlanCheck in Singapore; however, DesignCheck has the capability of offering checks for compliance at various phases in the design process, as it has a rule schema for early and detailed design stages as well as for construction documents phase. Thus, DesignCheck can be used by Architects and Designers and not only building authorities (Ding et al., 2006).

```
TYPE BUILDING_TYPE_ENUM
        CLASS_1A_SINGLE_DWELLING
        CLASS_1B_BOARDING_HOUSE
        CLASS_2_SOLE_OCCUPANCY_UNITS
        CLASS_3_RESIDENTIAL_BUILDING
        CLASS_4_A_DWELLING_IN_A_BUILDING
        CLASS_5_OFFICE_BUILDING
        CLASS_6_SHOP_BUILDING
        CLASS_7A_CARPARK
        CLASS_7B_STORAGE
        CLASS_8_LABORATORY
        CLASS_9A_PUBLIC_HEALTH_CARE_BUILDING
        CLASS_9B_PUBLIC_ASSEMBLY_BUILDING
        CLASS_9C_PUBLIC_AGED_CARE_BUILDING
        CLASS_10A_NON_HABITABLE_BUILDING
        CLASS_10B_NON_HABITABLE_STRUCTURE
```

FIGURE 4.16 New type definition in DesignCheck. (From Ding, L., et al., *Clients Driving Innovation: Moving Ideas into Practice*, Cooperative Research Centre (CRC) for Construction Innovation, Brisbane, Australia, 2006.)

```
ENTITY DOOR_CRC
        OVERALLHTIGHT
        OVERALLWIDTH
        DOOR_TYPE
        DOOR_STYLE
        IS_EXTERNAL
        SELFCLOSING
        FIREEXIT
        AS1428_ COMPLIANCE
        DC_LEVEL_HANDLE_CLEARANCE
        DC_DOOR_RECESS_DEPTH
        DC_SURFACE_MOUNTED
        DC_DOOR_HANDLE_OPERATION
END_ENTITY
```

FIGURE 4.17 Door entity in the DesignCheck engine. (From Ding, L., et al., *Clients Driving Innovation: Moving Ideas into Practice*, Cooperative Research Centre (CRC) for Construction Innovation, 2006.)

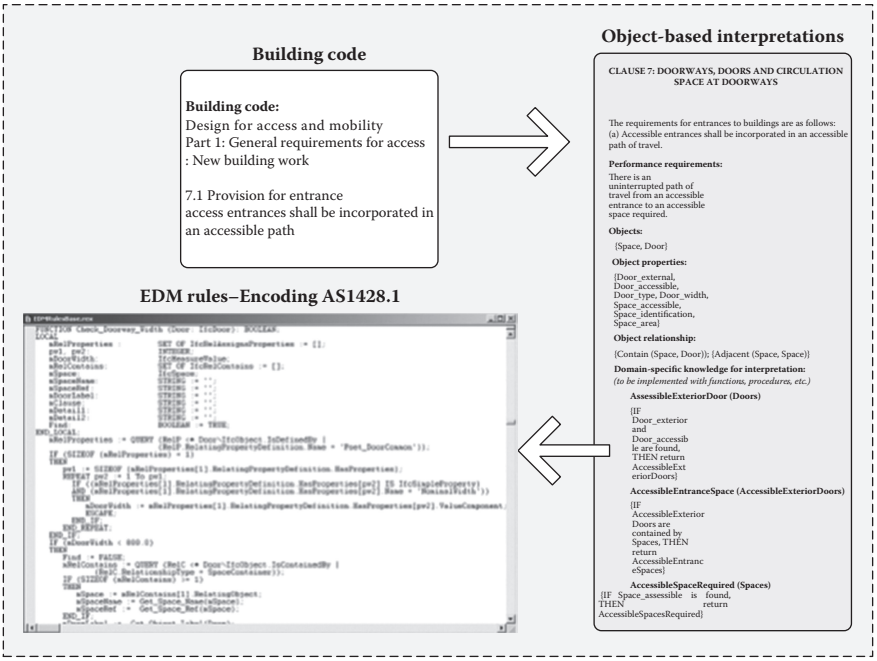


FIGURE 4.18 Process of encoding a building code clause into the EDM rules in the DesignCheck system. (Modified from Ding, L., et al., *Clients Driving Innovation: Moving Ideas into Practice*, Cooperative Research Centre (CRC) for Construction Innovation, 2006.)

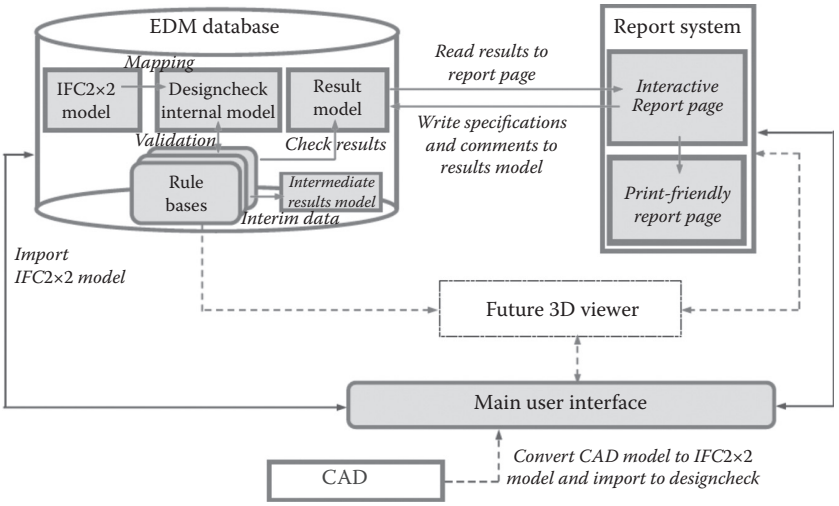


FIGURE 4.19 DesignCheck architecture. (Modified from Ding, L., et al., *Clients Driving Innovation: Moving Ideas into Practice*, Cooperative Research Centre (CRC) for Construction Innovation, 2006.)

Portugal's LicA System

LicA is a software system that performs the automated code compliance checking for domestic water system network projects according to Portuguese code. This software system was developed at the College of Engineering, University of Porto, Portugal. The conceptual architecture of LicA is depicted in Figure 4.20.

The LicA code-checking system is based primarily on a conventional relational database. The database includes a set of tables with all the objects needed to represent the water distribution network regulations and related calculation preferences. The system also includes hydraulic analysis tools and code compliance–checking routines, developed in T-SQL. Results are gathered in tables for postprocessing. The database can be accessed through Open Database Connectivity (ODBC), which enables any application based on programming languages such as Java or C# to access the LicA database.

LiCAD, the graphical user interface for LicA, enables results to be represented both in graphical and textural forms. The LicA database and LiCAD were developed as separate applications to foster the modularity of the system. In general, since databases usually have a larger life span than user interface applications, the modularity of the system is seen as a way of preventing future versioning and interoperability problems. LiCAD was developed in VB.Net, using ADO.NET objects to access the database. The user interface uses DirectX-based technology and Windows Presentation Foundation (WPF), which allows for rich visual representations and also to browse and edit the model using 3D interface elements such as rotation, translation, and zoom.

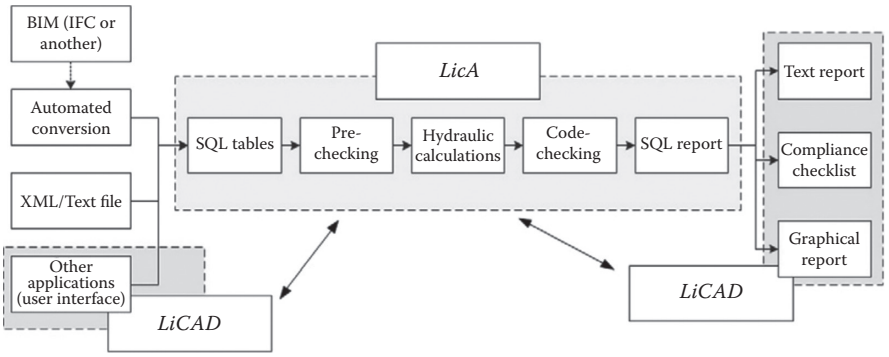


FIGURE 4.20 Conceptual architecture of LicA. (Modified from Martins and Montiero, 2013).

In its current version, LicA doesn't support the IFC data format. This is the main disadvantage of the LicA software application for code compliance auditing. The system is a strictly domain-specific application for code checking. The database was developed primarily according to the Portuguese building codes. Since building codes are originally developed for manual reviews, encoding regulations as a set of parameters and rules is not always possible. Ambiguities and other imperfections found in the regulations obstruct objective data interpretation by the computer or require the manual interpretation of the design information. LicA addresses these issues by creating different categories for the compliance-checking results, including a class for checks that were performed but that should be reviewed manually.

The LicA data model supports both hot and cold water pipe networks. The network representation entails nodes and flow segments. Nodes generally serve various functions. They may be simple geometric entities or they may have other functions and properties (Martins and Monteiro, 2013). Hydraulic analysis results are computed for all the nodes in the model, including consumptions. A node or nodes can represent devices along with their hydraulic variables, such as flow and pressure. Flow parts can assume the form of a pipe, a valve, or a pump. Pipe classification is determined according to the placement (inside or outside the building), thermal isolation (existing or nonexistent), and material type (Martins and Monteiro, 2013).

The location of each device is explicitly linked to the architectural design by means of a parameter that defines the position of the device relative to the building spaces. Information about the location of each device is a necessary requirement for the code compliance–checking system. The Portuguese regulations for domestic water systems are mostly incorporated into the LicA data model definition. However, it is important to note that it doesn't include every class of objects that are needed for typical MEP designs. Thus, before the code compliance checking with LicA begins, the software runs an initial model-checking routine to examine the integrity of the model and to ensure that the model satisfies the prerequisites for the software to run code auditing correctly.

Figure 4.21 depicts the user interface of LicA. It has four main tabs. The first tab from the left is for model definition and data input. The second tab includes

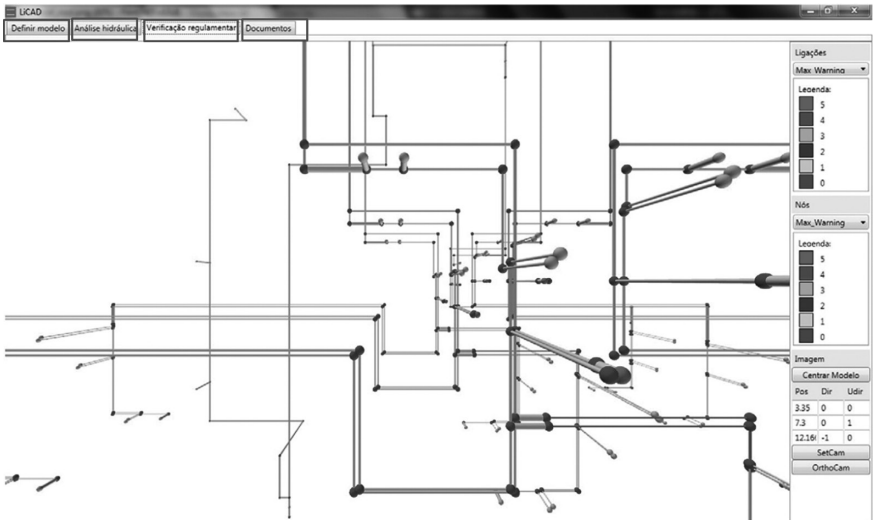


FIGURE 4.21 Results of code compliance checking using LicA. (From Martins, J.P. and Monteiro, A., *Automation in Construction*, 29(2013), 12–23, 2013.)

functions to run the hydraulic computations. The third tab executes the code compliance checking and presents results visually in a color-coded format (Figure 4.22). The last tab provides the results of the hydraulic computations and code compliance checking in textual reports.

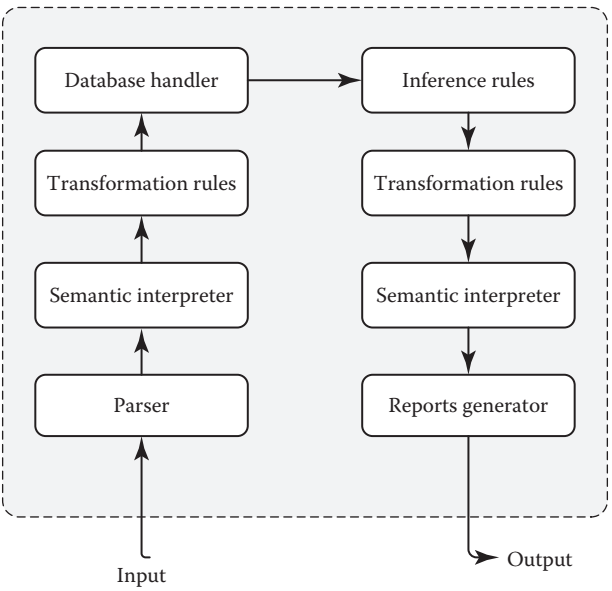


FIGURE 4.22 Overview of a general NLP system.

ARTIFICIAL INTELLIGENCE APPROACHES

Background

For thousands of years, humans have been in the quest for understanding how human intelligence works and reproducing it using some kind of machine. Using machines intelligently to assist in simple tasks such as pulleys lifting heavy weights, trains and automobiles in transportation, and electronic calculators for performing arithmetic operations seem not to satisfy human curiosity. Instead, engineers and scientists continuously seek to automate more challenging and complex tasks, such as recognizing faces and voices, flying an airplane, driving a car, or building a house. All these tasks require a level of human intelligence that is not found in simple machines such as digital calculators. In recent history, human culture has shown a great deal of fear and had many fantasies of machines with a high level of human-like intelligence (so-called AI). For example, when Gary Kasparov, the world chess champion, was beaten by the IBM Deep Blue computer in 1997, many people were scared of the potential of AI just as much as they celebrated that historic accomplishment.

The subject was finally formalized as AI in the 1950s. Some researchers were convinced that machines with human-like intelligence would appear within a couple of decades or so. But AI had a rough time at the early stages, and progress in the discipline has experienced a great deal of delay. In the 1970s, AI suffered a devastating academic challenge in terms of funding cuts and a loss of interest. Researchers came to the conclusion that machines based on the logic of absolute 1s and 0s that is the binary system would never be able to achieve the nuanced organic, often fuzzy thought processes of human biological brains. After a slowed period of progress in the field of AI, an extremely powerful idea emerged to lift the search for machine intelligence out of its lowest level of advancement: why not try to create an intelligent machine by emulating how the human brains works? Computing machines with brains and neurons instead of logic rules and probability decisions, more natural reasoning instead of binary logic and traditional black-and-white algorithms. Thus, the focus has become allowing computers to function somewhat like a human brain. However, it is important to note that this does not mean that AI seeks to emulate every aspect of the human brain. The degree to which an AI algorithm emulates the actual functioning of the human brain is called *biological plausibility.* Such emulation mostly occurs at a higher level.

Most people know relatively little about the internal operations of the human brain. On the other hand, most humans do know a great deal about its external operations. In other words, the brain is basically a black box connected by nerves. These nerves carry signals between the brain and the body. Normally, a number of inputs causes a certain output. For example, feeling hot in the room will result in various nerves sending commands to the body to move and change the thermostat to a more comfortable temperature. Human nerves provide the actual perception of the world. Thus, the nerves represent the inputs to the brain. Similarly, the only ways to interact with the world are the outputs from human nerves to the muscles. Certainly, the resulting output from the human brain depends entirely on the inputs and the internal state of the brain. In summary, the human brain responds to any input by altering its

internal state and producing a certain output. The consequence of the order of the inputs is handled by the internal state of the brain. This model of inputs, outputs, and internal states holds true for most AI algorithms, regardless of whether you are developing AI for flying a drone or converting a handwritten note to text. Obviously, some algorithms are more complex than others.

One approach to modeling an algorithm directly from the human brain is the neural network. Neural networks are one part of AI field, and the neural network concept parallels many aspects of human brain activity. Computer-based neural networks are not like the human brain in that they are not general-purpose computation devices. Neural networks, as they currently exist, carry out very specific tasks. An AI algorithm experiences its reality by providing output based on the algorithm's internal state and the input it is presently receiving.

AI is currently based on many algorithms. The AI algorithm is the technique that one uses to solve a problem. An AI algorithm is also often referred to as a *model*. Examples of the most common AI models include neural networks, support vector machines, Bayesian networks, natural language processing (NLP), and hidden Markov models.

Natural Language Processing

As indicated earlier, the goal of AI techniques has always been to simulate, in some way or other, human intelligence, knowledge, and perception. The AI methods of modeling human languages (English, Spanish, Hindi, etc.) use various NLP algorithms to drive meaning from the provided natural language text. NLP as a discipline has been developing for many years. It began in 1960 as a subfield of AI and linguistics, with the objective of investigating the automatic creation and understanding of human language. In other words, NLP is concerned with the interaction between natural languages and computers. NLP applications can range from summarizing news to understanding verbal commands in your smart device, such as Apple Siri and Microsoft Cortana, or translating languages.

Many researchers have indicated that the ability of computers to process natural language as skillfully as humans do will indicate the arrival of truly intelligent machines. The basis of this belief is the fact that the fluent and articulate use of language is closely related to human cognitive ability. Thus, one of the first NLP applications was the Machine Translation (MT) conceived by Weaver in 1945.

In the sixties and seventies, the two key approaches of NLP were the statistical approach and the linguistic emphasis. Both methods vary considerably, although in practice, NLP stems utilize a mixed approach, combining techniques from both approaches. The statistical approach (Manning, 1999) represents the classical model of information retrieval systems and is characterized by extracting the nouns, determining their types, and providing some "scoring" (relevance or sentiment) of the entity within the text using statistical methods such as frequency of occurrence and probability functions. On the other hand, the linguistic approach is based on the application of different techniques and rules that explicitly encode linguistic knowledge (Sanderson, 2000). Text documents are analyzed through different linguistic

levels by linguistic tools that incorporate each level's own annotations to the text. The different steps in a typical NLP linguistic analysis of documents include the following:

- *Morphological analysis* concerns word formation. It deals with the analysis of words according to their components, called *morphemes*. In this analysis, nonwords (i.e., punctuation, etc.) are separated from words. Morphemes are the minimal meaningful units in a language that cannot be further broken into smaller units. Furthermore, morphological analysis describes matching the derivational and inflectional forms of a word to its base form, which may facilitate the identification of semantic features such as concepts from an ontology. This analysis is performed by *taggers* that assign each word to a grammatical category according to the morphological characteristics found.
- After having identified and analyzed the words in a text, the next step is to see how they are related and used together in making larger grammatical units, phrases, and sentences.
- Next, a syntax analysis of the text is performed. This is when *parsers* are applied (i.e., a descriptive formalism that establishes the text's syntax structure). The methods utilized to apply and generate parsers differ and depend on the purpose of the syntax analysis. In case of information retrieval, it is often used for a shallow analysis aiming to only identify the most meaningful structures: nominal sentences, verbal and prepositional sentences, values, and so on. This level of analysis is usually used to optimize resources and improve the system's response.
- After analyzing the text's syntax structure, the next step is to get the meaning of the sentences within it. The goal is to acquire the sentence's semantic representation from the elements that make it up.

Currently, NLP employs various probabilistic and data-driven models. Algorithms for parsing, part-of-speech tagging, reference resolutions, and discourse processing all began to incorporate probabilistic and neural network principles as well as evolution strategies from speech recognition and information retrieval. It is in essence an interdisciplinary area where computational linguistics combine with AI. NLP utilizes AI tools such as algorithms, data structures, formal models of knowledge representations, models for reasoning processes, and so on. The goal of NLP is to specify language comprehension and production theory to such a level of detail that a person and a machine can communicate naturally. Some of the major tasks of the NLP system include language reading and writing; automatic summarization; voice and character recognition; IE; IR, which is concerned with storing, searching, and retrieving information; machine translation; translation from one language into another; natural language generation; natural language understanding; spoken dialog system; text-to-speech; text proofing; and text simplification.

Figure 4.23 illustrates a general NLP system that includes many of the aspects discussed previously. First, the input is provided in a natural language. This input

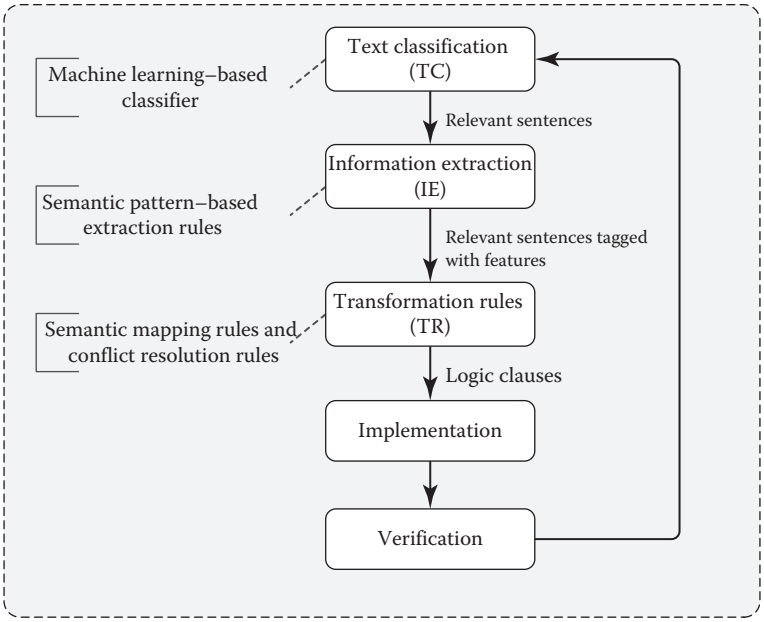


FIGURE 4.23 NLP rule extraction approach for automating code conformance checking. (Modified from Zhang, J. and El-Gohary, N., *Journal of* Computing *in* Civil Engineering, 2015, 29(4), B4015001, ASCE, 2015.)

is then given to a parser to generate the syntactic structure of the sentences. Then a semantic interpreter captures their semantics details and generates a deeper structure. The conversion rules process the deep structure of the sentences and makes them compatible with the database storage system. The database handler treats the information received to generate processed forms of the sentences. All these steps represent the input side of the system. The reverse process is indicated in Figure 4.17 to generate natural language results.

One of the main differences between data processing and NLP is the utilizing of knowledge by NLP. The knowledge is related to various linguistic structure and rules. These can be grouped into a number of knowledge categories, such as (1) morphology (the knowledge of meanings of words and word components), (2) phonetics and phonology (the knowledge of linguistics and sounds), (3) syntax (the structural relationship between words and grammar), (4) semantics (the meanings of words and sentences), (5) pragmatics (the knowledge of how language is used to achieve certain tasks), and (6) discourse (the science of joining linguistic units larger than a single unit of speech).

As indicated previously, one of the main critical problems with natural language is ambiguity, which is the inherent weakness in any human language. To deal with the vagueness of natural languages, NLP seeks to identify certain types or categories to allow for their modeling. These categories include the following:

1. *Lexical ambiguity*: This refers to the ambiguity resulting from the several meanings of a word. To resolve such ambiguity, NLP methods normally utilize additional knowledge from the context.
2. *Syntactic vagueness*: This deals with the fact that a sentence can be parsed differently and requires common sense to resolve it.
3. *Referential ambiguity*: This refers to the vagueness resulting from the use of pronouns and other anaphora.
4. *Pragmatic ambiguity*: This type of vagueness underlies the meaning of a sentence. It arises from the different intentions of the writer or the speaker. To resolve such vagueness, both contextual knowledge and commonsense knowledge are required.

In sum, NLP is generally broken into a series of levels of analysis—namely, phonological analysis, morphological analysis, lexical analysis, syntactical analysis, semantic analysis, and pragmatic and discourse analysis. Phonological analysis is the analysis of spoken language and deals with speech recognition and generation. Morphological analysis is the most basic phase of NLP. It is the task of breaking words into their morphemes. Semantic analysis deals with the meaning of human language sentences. NLP semantic analysis generates a knowledge representation (linguistic model) to capture the meaning of a sentence. Lexical analysis focuses on the validity of words according to a certain lexicon (dictionary). Details about words such as nouns, verbs, adverbs, and others are normally examined in such analysis. It is the first stage of processing input text. The lexicon provides the necessary data and rules to carry out the first phase of the analysis. Syntactic analysis is the study of formal relationships between the words of sentences. This analysis examines the validity of a sentence according to grammar rules. Pragmatic analysis aims at the relationships between language and the context of use (the contextual environment).

Artificial Intelligence for Building Regulations Compliance Checking

AI methods in the field of building code conformance auditing aim to fully automate the process of building code conformance checking by extracting and encoding regulatory requirements to enable computer processing. These methods are generally based on the use of NLP models that predict the probability distribution of language expressions. They include two main types: a rule-based approach or a machine learning (ML)-based approach. A rule-based method uses manually developed rules to process documents, whereas an ML-based approach refers to a system learning from available data or previous experience and employs ML algorithms (e.g., support vector machines, hidden Markov models) to process text. ML-based methods can be one of the following types:

1. *Supervised*: Human supervision is provided in the form of labeled documents (all documents are given one or more predefined labels), in which a training data set is used to train the classifier system to automatically categorize a given document according to a predefined set of labels, and a testing data set is used to test the performance of the classifier.

2. *Unsupervised*: Documents are not manually categorized for training; consequently, instead of classifying given documents according to a predefined set of labels, classifiers automatically (and without human direction) cluster documents into potentially useful categories.
3. *Semisupervised*: Only a segment of the training data set is labeled, which provides partial human guidance.

In contrast to unsupervised and semisupervised, supervised ML-based algorithms require intense manual effort for preparing the training data set. However, their precision and performance are typically higher than the other methods. In general the rule-based approaches achieve better text-processing performance in comparison to ML-based approaches (Crowston et al., 2010).

NLP methods can be also classified into shallow and deep approaches, distinguished from each other by their different emphasis on text processing. If the emphasis of an NLP approach is on analyzing incomplete sentences or particular topics, then the NLP method is considered shallow. On the other hand, if the focus of an NLP approach is on processing full sentences or entire meanings, then the approach is regarded as deep (Zouaq, 2011). The NLP techniques that have achieved the most reasonable performance results are shallow NLP tasks. On the other hand, deep NLP methods are still challenging, because efficient deep NLP methods need elaborated knowledge representation and efficient reasoning about the domain, both of which remain challenging in AI (Tierney, 2012).

In the AEC domain, there have been many research works that have focused on NLP techniques. For instance, Caldas and Soibelman (2003) have worked on the ML-based text classification of construction documents. Zhang and El-Gohary (2012, 2013a, 2015) proposed various approaches for automating building regulatory conformance checking using NLP techniques. These approaches utilize semantic modeling and semantic NLP techniques (text classification, information extraction, and transformation) to facilitate the automated processing of building regulatory documents for extracting regulatory rules and formalizing these requirements into computable formats. They generally include developing a set of algorithms into a computational platform that has (1) ML-based algorithms for text classification (TC), (2) rule-based, semantic NLP algorithms for IE, and (3) rule-based, semantic NLP algorithms for information transformation (ITr). Discipline-specific, semantic NLP-based information processing techniques can attain in some cases complete sentence processing and information extraction, in contrast to partial-sentence processing and information extraction. This means that only definite concepts and terms are extracted and processed. Domain-specific semantic methods are supposed to enable the analysis of composite sentence structures that would otherwise be too complex and vague for automated IE and ITr techniques.

Zhang and El-Gohary (2015) proposed an NLP approach for automated building code checking that is based on a domain-specific semantics approach. This method has five steps: TC, IE, transformation rules (TR), implementation, and evaluation (Figure 4.23). The core processing phases are TC, IE, and TR.

TC identifies relevant text in a building regulatory corpus. Relevant texts are sentences that encompass building code requirements. Target information in those

pertinent sentences is extracted and transformed (IE and TR processes). In other words, the TC process eliminates unrelated texts, thus saving the unnecessary processing of extraneous information. Also, such cleaning mechanisms circumvent needless extraction and transformation errors that may be produced by the processing of irrelevant sentences.

IE consists of extracting entities, events, and existing relationships between elements in a text or groups of documents. It centers here primarily on words and phrases in the relevant sentences that convey target information and extracts information from these terms and phrases. In other words, IE applies many NLP methods to detect target information from unstructured documents and characterizes the target information into a formal schema (Jurafsky and Martin, 2009). One of these IE techniques is the ontology-based IE method, which utilizes ontology to extract meaning-driven information to support capturing semantic information that is specific to a knowledge domain (Wimalasuriya and Dou, 2010; Karkaletsis et al., 2011).

The preprocessing phase of IE may include text splitting, tokenization (splitting the text into tokens such as words, numbers, punctuations, symbols, and whitespaces), sentence splitting (dividing the text into individual sentences based on sentence-ending symbols such as periods, question marks, exclamation marks, etc.), and morphological analysis (matching the derivational and inflectional forms of a word to its base form, which may facilitate the identification of semantic features such as concepts from an ontology). IE then labels each of these information entities with predefined tags. An information tag is a symbol or an identifier for a specific meaning. For instance, the information tag *subject* conveys the semantic meaning that the information instance is an *object* (column, beam, or any building element) that is subject to certain regulations or specifications, while the information tag *HH* carries the syntactic meaning that the information instance is an adjective that defines a noun as a convertor. Target information is the data required to examine certain kinds of building regulations (Zhang and El-Gohary, 2015). IE can be ontology based, where it utilizes ontology to identify meaning-based data to assist in extracting semantic information that is specific to a domain (Karkaletsis et al., 2011; Wimalasuriya and Dou, 2010). (Ontology is a semantic model; it is defined as a conceptualization technique and is generally comprised of concept hierarchies, relationships between concepts, and axioms.) In contrast to non-ontology-based IE, which only uses the lexical and/or syntactic information of the text in the extraction of target information, ontology-based IE further relies on the meaning-driven information from a particular discipline to improve the extraction performance in a specific domain and/or application.

Information TR receives the extracted information objects and transmutes them into logic statements (that can be further used in logic programs) using a set of pattern-matching-based rules (Figure 4.23). The types of rules that are used by Zhang and El-Gohary (2015) in their study include semantic mapping (SeM) rules and conflict resolution (CoR) rules. The logic section elements in a concept predicate are referred to as *concept logic clause elements*. The logic section elements in a relation predicate are called *relation logic clause elements*. Table 4.2 depicts an example of a sentence and the associated source and target logic elements.

The SeM rules and CoR rules utilize a total of 40 information tags for TR (Zhang and El-Gohary, 2016). A summary of these information tags is depicted

TABLE 4.2
Example Transformation Rules

Requirement Sentence	Source: Information Tag	Source: Information Instance	Target: Logic Clause
Courts shall not be less than 3 ft in width	Subject	Court	Compliant_Width_of_Court:
	Compliance checking attribute	Width	width(Width), court(Court), has(Court,Width),
	Comparative relation	Not less than	greater_than_or_
	Quantity value	3	equal(Width,quantity(3,feet))
	Quantity unit	Feet	
	Quantity reference	Not applicable	

in Figure 4.25. The SeM rules describe how to handle the extracted information cases according to their semantic connotation. For instance, in Table 4.2, "subject" describes the semantic meaning of "court" (i.e., it defines "court" as the "subject" of compliance checking), and the contextual environment is determined by the information tags of its surrounding information instances.

In general, the semantic meanings of information instances are utilized in patterns of SeM rules. For the instance, in Table 4.2, the equivalent SeM rule form is "subject" + "modal verb" + "negation" + "be" + "comparative relation" + "quantity value" + "quantity unit" + "preposition" + "compliance checking attribute." An SeM rule form will convert the information instances into the logic clause depicted in the last column of Table 4.2. Consequently, the following logic clause objects are created for the following statement because "court" is recognized as a "subject" information instance and "width" as an "attribute" information instance. In the sentence "Courts shall not be less than 3 ft in width," the corresponding logic clause statement is court(Court), width(Width), has (Court, Width).

CoR rules target the resolution of conflicts between various information tags. Two types of CoR rules are proposed by Zhang and El-Gohary (2015)—namely, deletion CoR rules and conversion CoR rules. Deletion CoR rules solve the conflict issues

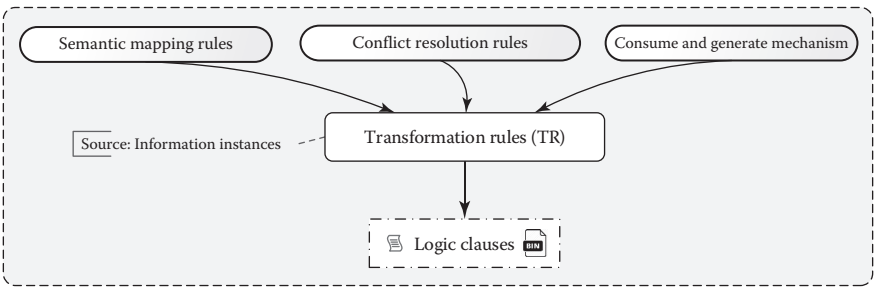


FIGURE 4.24 TR rules (Modified from Zhang and El-Gohary, 2015).

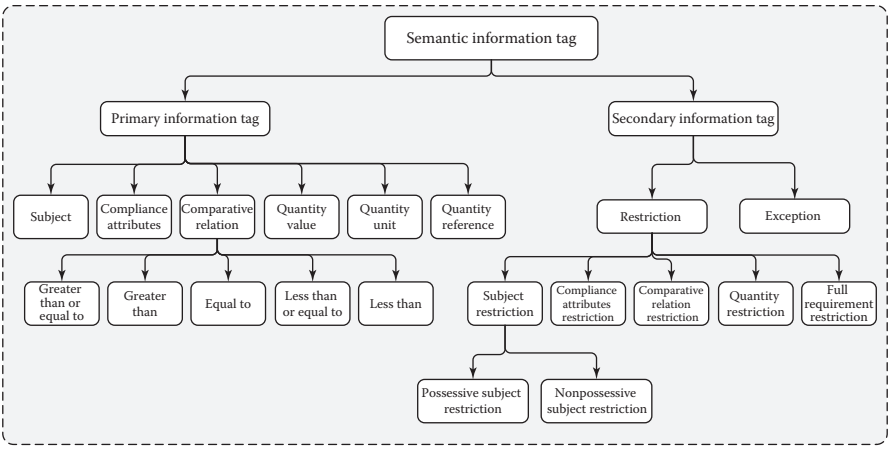


FIGURE 4.25 Semantic information tags.

between information tags by deleting redundant information instances. On the other hand, modification CoR rules resolve conflicts among information tags by changing the information tags of information instances into other types of information tags.

The final step in this NLP approach for the automatic verification of building codes is the verification phase. The results of this approach are examined in terms of *precision*, *recall*, and *F1 measure*. Precision is determined by the ratio of the number of properly produced logic clause statements to the total number of created logic clause statements. The recall parameter is determined by the ratio of the number of correctly generated logic clause statements to the total number of logic clause statements that should be created. The F1 measure is defined as the *harmonic mean* (also known as *subcontrary mean*, the reciprocal of the arithmetic mean of the reciprocals) of the precision and recall parameters, assigning equal weights to precision and recall. Table 4.3 summarizes the main parameters used for the verification of this approach. Preferably, both 100% recall and precision are desired in automated

TABLE 4.3
Verification Parameters Used in the NLP Method Proposed by Zhang and El-Gohary

Parameters	Value
Number of logic clause elements that should be generated	Integer
Total number of logic clause elements generated	Integer
Number of logic clause elements correctly generated	Integer
Precision	Decimal or percentage (ratio)
Recall	Decimal or percentage (ratio)
F1 measure	Decimal or percentage (ratio)

Source: Zhang, J. and El-Gohary, N., *Journal of Computing in Civil Engineering*, 2015, 29(4), B4015001

TABLE 4.4
Examples of SeM Rule Patterns

SeM Rule Pattern	Action	Condition	Logic Clause Generated	SeM Rule Type
["a" "s" "cr"] (a) "OF" (b) ["a" "s" "cr"] (c)	—	—	a(A),c(C),has(C,A)	Simple
"c" (a) "v" (b)	Look-back search for attribute or subject(s)	*n* exists	not a(S, quantity(b,u))	Complex
"VB" ^ "be"] (a) "IN" (b) ["cr" "a" "s"] (c)	Look-back search for subject or attribute(s)	—	s(S),c(C),b(S,C)	Multiple action
"TO" (a) "VB" (b) ["s" "cr" "a"] (c)	Look-back search for attribute or subject(s)	*s* does not exist	c(C),a_b(C)	Complex

Source: Zhang, J. and El-Gohary, N., *Journal of* Computing *in* Civil Engineering, 2015, 29(4): B4015001.

code compliance checking. However, given the inherent complexity of building code provisions and specification, it is difficult to achieve such a result.

This methodology outlined above has been implemented and demonstrated in an example of building regulation conformance checking by Zhang and El-Gohary (2015). The methodology was implemented using the Python programming language. The NLP processing steps for this implementation are illustrated in Figure 4.26. In Figure 4.26a,b, the IE process tags the initial sentence with information tags. The information TR algorithm then transmutes each information case in the tagged sentence into a four-tuple (Figure 4.26c). Figure 4.26d,e depicts how the system executes a set of SeM rules to process each tuple in the list to create logic section elements founded on the matching of SeM rule patterns. An example of these SeM rule patterns is given in Table 4.3.

The following is part of the tag legend for the semantic information element (for a complete list, please refer to Zhang and El-Gohary, 2015).

A = candidate compliance checking attribute
C = for comparative relation
G = the comparative relation greater than
Cr = candidate restriction
S = candidate subject
VB = base form verb, based on POS tag "VB"
IN = preposition, based POS tag "IN"
JJ = adjective, based POS "JJ"
TO = literal "to," based on POS tag "TO"

The NLP method suggested by Zhang and El-Gohary (2012, 2013a,b, 2015) showed promising results for specific types of regulatory text in terms of automatically converting the extracted information instances into logic clauses for additional compliance checking. The main limitations of this technique are that the proposed methodology was only tested on processing quantitative regulatory data.

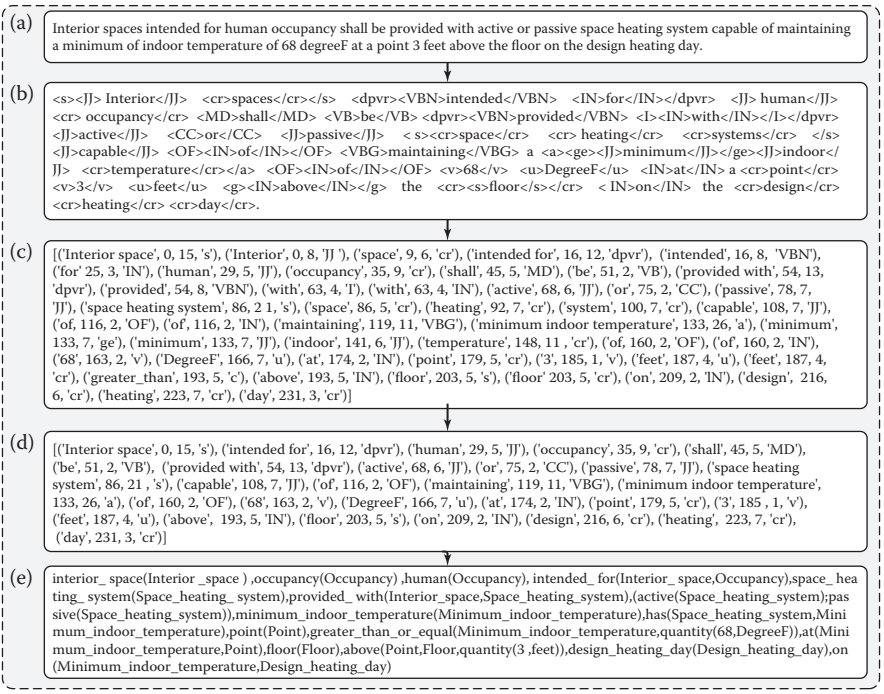


FIGURE 4.26 Example illustrating the NLP method for automating the code-checking compliance: (a) original code provision sentence, (b) sentence tagged with information tags, (c) information instance tuple list generated by TR, (d) conflict resolution rules applied to produce information instance tuple list, (e) logic clauses produced by the consume and generate mechanisms. (From Zhang, J. and El-Gohary, N., *Journal of* Computing *in* Civil Engineering, 2015, 29(4), B4015001, ASCE, 2015.)

Other types of semantic and subjective patterns may not result in the same degree of accuracy of methodology. Another limitation is that there is a great volume of manual effort required in developing a gold standard for the proposed TR algorithms. Further limitations are related to the computational efficiency of the proposed NLP approach. The proposed NLP approach for code compliance checking is not evaluated for compactional efficiency in the case of large regulatory provisions documents. A more practical working system for automated building code compliance auditing is still needed. Chapter 5 is devoted entirely to developing such a system, which is more valuable for practical applications than exploring the theoretical aspects of NLP.

REFERENCES

AEC3 (2006). An international consulting firm. http://www.aec3.com/en/5/5_013_ICC.htm (accessed 3/12/2017).

buildingSMART International. (2016). Model Support Group of buildingSMART International. http://www.buildingsmart-tech.org/about-us/msg (accessed 11/12/2016).

Caldas, C.H., and Soibelman, L. (2003). *Automating hierarchical document classification for construction management information systems*. Automation in Construction, 12, 395–406.

CDG (2007). US Courts design guide. Administrative Office of the US Courts, Space and Facilities Division. http://www.gsa.gov/graphics/pbs/Courts_Design_Guide_07.pdf (accessed 10/02/2010).

Choi, J., (2014). A Study on the Development of Code Checking System for Building Administration Process Applying Open BIM-based Process. Master dissertation, Kyung Hee University.

Corke, G. (2013). Solibri Model Checker v8 brings BIM model quality into focus with its powerful rules-based checking and auditing tool. *AEC Magazine*, February 2013.

Crowston, K., Liu, X., Allen, E., and Heckman, R. (2010). *Machine learning and rule-based automated coding of qualitative data*. Proc., 73rd ASIS&T Annual Meeting: Navigating Streams in an Information Ecosystem, Association for Information Science and Technology, Silver Spring, MD, 1–2.

Digital Alchemy (2016). http://www.digitalalchemy.com (accessed July 2016).

Ding, L., Drogemuller, R., Rosenman, M., Marchant, D., and Gero, J. (2006). Automating code checking for building designs: DesignCheck. In K. Brown, K. Hampson, and P. Brandon (eds), *Clients Driving Innovation: Moving Ideas into Practice* (12–14 March 2006). Cooperative Research Centre (CRC) for Construction Innovation, Wiley-Backwell, UK.

Eastman, C., Lee, J-M., Jeong, Y-S., Lee, J-K. (2009). Automatic rule-based checking of building designs. *Automation in Construction*, 18(8), 1011–1033.

EDM (2016). EXPRESS Data Manager™ (EDM), One-line: http://www.jotneit.no/express-data-manager-edm (accessed: 29/11/2016).

Fiatech (2011). *An introduction to ISO 15926*. Element, Workforce and Training. November. http://www.fiatech.org/workforce-training.

Haraldsen, M., Stray, T.D., Päivärinta, T., and Sein, M.K. (2004). Developing eGovernment portals: From life-events through genres to requirements. In K.H.R. Rolland (ed.), *Proceedings of 11th Norwegian Conference on Information Systems*, Stavanger, Norway, pp. 44–70.

Jurafsky, D. and Martin, J.H. (2009). *Speech and Language Processing*, 2nd edn. Prentice Hall, Upper Saddle River, New Jersey.

Karkaletsis, V., Fragkou, P., Petasis, G., and Iosif, E. (2011). Ontology based information extraction from text. In *Lecture Notes in Computer Science*, 89–109. Berlin: Springer.

Khemlani, L. (2005). CORENET e-PlanCheck: Singapore's automated code checking system. AECbytes. http://www.aecbytes.com/buildingthefuture/2005/CORENETePlanCheck.html (accessed June 2009).

Khemlani, L. (2007). Top criteria for BIM solutions: AECbytes survey results. AECbytes. http://www.aecbytes.com/feature/2007/BIMSurveyReport.html (accessed January 2008).

Kim, I., Choi, J., and Cho, G., (2013). Development of Rule-based Checking Modules for the Evacuation Regulations of Super-tall Buildings in Open BIM Environments. *Transactions of the Society of CAD/CAM Engineers*, 18(2), 83–92.

Lee, H., Park, S., Kim, I., and Lee, J., (2015). A Logical Rule-based Approach to the Korea Architecture Code Sentences for BIM-enabled Design Assessment Systems. *Journal of Korea Design Knowledge*, 34, 101–110.

Lee, J. (2011). Building Environment Rule and Analysis (BERA) language, Ph.D. thesis, Georgia Institute of Technology.

Manning, C. D., and Schutze, H. (1999). *Foundations of statistical natural language processing.* MIT Press, Cambridge, MA.

Martins, J.P. and Monteiro, A. (2013). LicA: A BIM based automated code-checking application for water distribution systems. *Automation in Construction*, 29(2013), 12–23.

Sjøgren, J. (2007). BuildingSMART: A smart way for implementation of standards. http://www.plc-sresources.org/papers/s1000d/day3/buildingSMART%20-%20Jons%20Sjogren.pdf (accessed June 2010).

Tierney, P. J. (2012). *A qualitative analysis framework using natural language processing and graph theory*. Int. Rev. Res. Open Distance Learn. 13(5), 173–189.

United States General Services Administration (2006). BIM guide for spatial program validation appendices, GSA BIM Guide Series 02. http://www.gsa.gov/graphics/pbs/GSA_BIM_02_Appendix_v09.pdf (accessed June 2010).

United States General Services Administration (2007). BIM guide for spatial program validation, GSA BIM Guide Series 02. http://www.gsa.gov/graphics/pbs/BIM_Guide_Series_02_v096.pdf (accessed June 2010).

Wimalasuriya, D.C. and Dou, D.J. (2010). Ontology-based information extraction: An introduction and a survey of current approaches. *Journal of Information Science*, 36(3), 306–323.

Yang, Q. (2003). IFC-compliant design information modelling and sharing. *Journal of Information Technology in Construction*, 8, 1–14. International Council for Research and Innovation in Building and Construction.

Zhang, J., and El-Gohary, N. (2012). *Automated regulatory information extraction from building codes leveraging syntactic and semantic information*. Proc., 2012 ASCE Construction Research Congress (CRC), West Lafayette, IN.

Zhang, J. and El-Gohary, N. (2015). Automated information transformation for automated regulatory compliance checking in construction. *Journal of* Computing *in* Civil Engineering, 29(4), B4015001. ASCE.

Zhang, J. and El-Gohary, N.M. (2013a). Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking. *Journal of* Computing *in* Civil Engineering, 10.1061/(ASCE).

Zhang, J. and El-Gohary, N. (2016). *Semantic-Based Logic Representation and Reasoning for Automated Regulatory Compliance Checking*. Journal of Computing in Civil Engineering, 04016037. ASCE.

Zhang, J. and El-Gohary, N.M. (2013b). Information transformation and automated reasoning for automated compliance checking in construction. *Proceedings of Computing in Civil Engineering (2013), ASCE*, Reston, VA, 701–708.

Zhong, B. T., Ding, L. Y., Luo, H. B., Zhou, Y., Hu, Y. Z., and Hu, H. M. (2012). Ontology-based semantic modeling of regulation constraint for automated construction quality compliance checking. *Automation in Construction*, 28, 58–70.

Zhang, J. and El-Gohary, N.M. (2016). Semantic NLP-based information extraction from construction regulatory documents for automated compliance checking. *Journal of Computing in Civil Engineering*, 30(2): 0401501–04015014, ASCE.

Zouaq, A. (2011). *An overview of shallow and deep natural language processing for ontology learning*. Ontology learning and knowledge discovery using the web: Challenges and recent advances, IGI Global, Hershey, PA, 16–38.

5 Practical Approaches

INTRODUCTION

This chapter centers on introducing practical approaches for the automated checking of building regulations conformance. Regulations are normative text that define laws, codes, specifications, standards, and so on. Regulations generally provide constraints for technical solutions or activities that take place under defined conditions. Another characteristic feature of regulations is that they have a limited vocabulary. Automated model checking against building regulations can be a significant contribution to the valid interpretation of regulations. This automated approach can also contribute to the checking of regulations that have not been checked due to a lack of awareness of the code specifications or limitations in time and competency. Thus, such automation or semiautomation can have a substantial impact on the AEC industry.

The main goal of the practical methods is to offer computable models with clear syntax and semantics that can be used to represent and reason about building code requirements and provisions. Furthermore, the approach should be well-suited to the software providers by providing, for instance, satisfying software engineering principles such as reducing model complexity or making implementations reusable in varying contexts. In these approaches, it is critical that a computable building rules model is achieved before one starts the development of automatic building code conformance–auditing systems. The goal is to develop a unified format to exemplify building regulations and building information modeling (BIM). The reduction of complexity is a key element for the development of practical methods to support the automation or semiautomation of building regulations compliance checking. An object-based representation should define the minimum extent of data essential for the automatic checking of building code compliance.

FRAMEWORK

In the intended framework, the compliance-checking component would read an object-based building model and audit the model against selected set of object-based standard constraints. For clauses requiring subjective qualitative performance, different methods need to be considered. Some of these methods depend on fuzzy logic techniques and others require manual verifications. Thus, the proposed framework starts with classifying clauses of any given building code into four main categories, as demonstrated in Figure 5.1. Namely, these are *provisory (conditional) clauses*, *content clauses*, *ambiguous clauses*, and *dependent clauses*.

Provisory (*Prov*) clauses are the parts of the regulations that can be transmuted directly from the textual format into sets of object rules. Examples of such clauses are very common and typical structures include rules with specific values such as

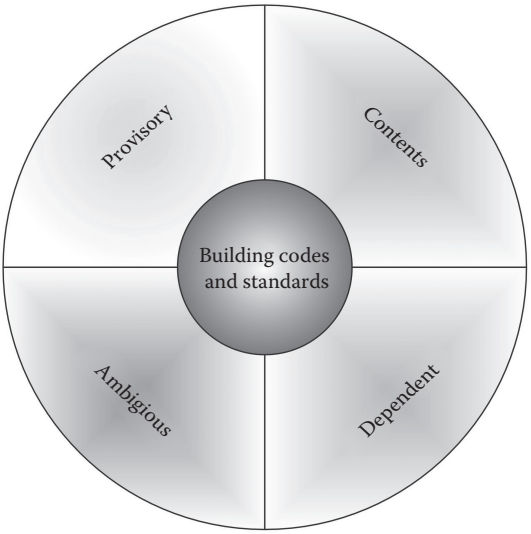


FIGURE 5.1 Main classes of building regulations and standards.

those given in Table 5.1. Contents (*Cont*) clauses are the sections of the building codes that cannot be translated into object rules. These clauses are usually devoted to definitions, such as the definition of types of loads, fire walls, fire rates, smoke evacuation, high-rise buildings, and so on. For example, the live load is defined by the ASCE7-10 as "a load produced by the use and occupancy of the building or other structure that does not include construction or environmental loads, such as wind load, snow load, rain load, earthquake load, flood load, or dead load." Ambiguous (*Ambi*) clauses are the subjective provisions. They normally include words such as *approximately*, *about*, *relatively*, *close to*, *far from*, *maybe*, and so on. An example of such provision is the footnote of the design lateral soil pressure for the clause given in ASCE 7-10:

> For *relatively* rigid walls, as when braced by floors, the design lateral soil load shall be increased for sand and gravel type soils to 60 psf (9.43 kPa) per foot (meter) of depth. Basement walls extending not more than 8 ft (2.44 m) below grade and supporting *light* floor systems are not considered as being *relatively* rigid walls.

This part covers all regulations that are not capable of being computerized, and some of them may have to be rewritten to enable implementation in automated code-checking environments. Interpretation and rewriting must adhere to terms from both the legal and construction perspectives.

Dependent (*Dep*) clauses specify that one clause is reliant on one or more other provisions. This means that some requirements are only appropriate for a specific condition when other clauses are satisfied. These clauses generally contain provisory clauses and are often problematic to transform into sets of immediate rules. Sometimes they may necessitate manual examination for compliance. For instance, Florida Building Code (FBC 2014) section 503.1, regarding building height and area, states

TABLE 5.1
Example of Conditional Clauses in ASE 7-10

Building Code	Provision	Encoding Using First-Order Logic (FOL)
ASCE 7-10: Standard for minimum design loads for buildings and other structures, has the following provision (3.2.1) for computing lateral pressure	In the design of structures below grade, provision shall be made for the lateral pressure of adjacent soil. If soil loads are not given in a soil investigation report approved by the authority having jurisdiction, then the soil loads specified in Table 3.2-1 shall be used as the minimum design lateral loads. Due allowance shall be made for possible surcharge from fixed or moving loads. When a portion or the whole of the adjacent soil is below a free-water surface, computations shall be based upon the weight of the soil diminished by buoyancy, plus full hydrostatic pressure. The lateral pressure shall be increased if soils with expansion potential are present at the site as determined by a geotechnical investigation	∀x *Structure*(x) ^ *BelowGrade*(x) ⟶ *Required*(x, lateral pressure)∀x ¬ *SoilReport*(x) ⟶ *Required*(x, Table 3.2.1) ∀x *Structure*(x) ^ *BelowGrade*(x) ^ *BlelowWaterSurface*(x) ⟶ (*Required*(x, lateral pressure submerged) ^ *Required* (x, hydrostatic pressure)) ∀x *Structure*(x) ^ *BelowGrade*(x) ^ *ExpansiveSoil* (x) ⟶ *Required* (x, increase lateral pressure)

Source: Nawari, N.O., The challenge of computerizing building codes in a BIM environment, *Proceedings of the International Conference on Computing in Civil Engineering*, ASCE, Clearwater Beach, FL, June 17–20, 2012b.

The building height and area shall not exceed the limits specified in Table 503 based on the type of construction as determined by Section 602 and the occupancies as determined by Section 302 except as modified hereafter. Each portion of a build separated by one or more fire walls complying with Section 706 shall be considered to be a separate building.

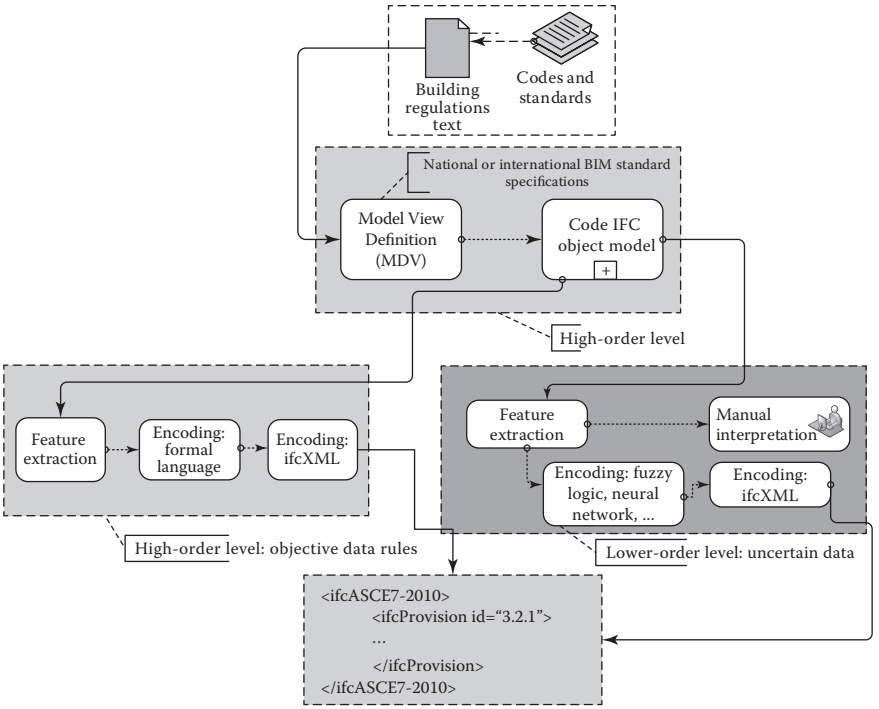


FIGURE 5.2 The framework.

The proposed approach includes the classification, transformation, and mapping of building regulation information into other simplified and thus less complex object-based schemas. This can be established by considering the following development levels (Nawari and Alsaffar, 2015) (Figure 5.2):

1. *High-order level 1*: Requires the development of the Model View Definition (MVD), which will lead to the Industry Foundation Classes (IFC) schema.
2. *Higher-order level 2*: Requires the feature extraction of specific data objective concepts leading to full encoding. This includes the transformation of conditional and dependent building regulations into sets of rules using object-based models.
3. *Lower-order level*: Necessitates the feature extraction of all ambiguous information and uncertain data, then employing partial encoding using fuzzy logic, neural networks, or combinations of these systems, and full manual checking (Figure 5.3).

BIM-based automated code conformance checking is one of the most effective ways to utilize BIM technology and to contribute to the development of new procedures in the AEC industry by expanding the AEC BIM matrix knowledge application. The AEC industry is highly regulated, and there are a large number of rules, codes, guidelines, and standards to comply with. The constraints in the

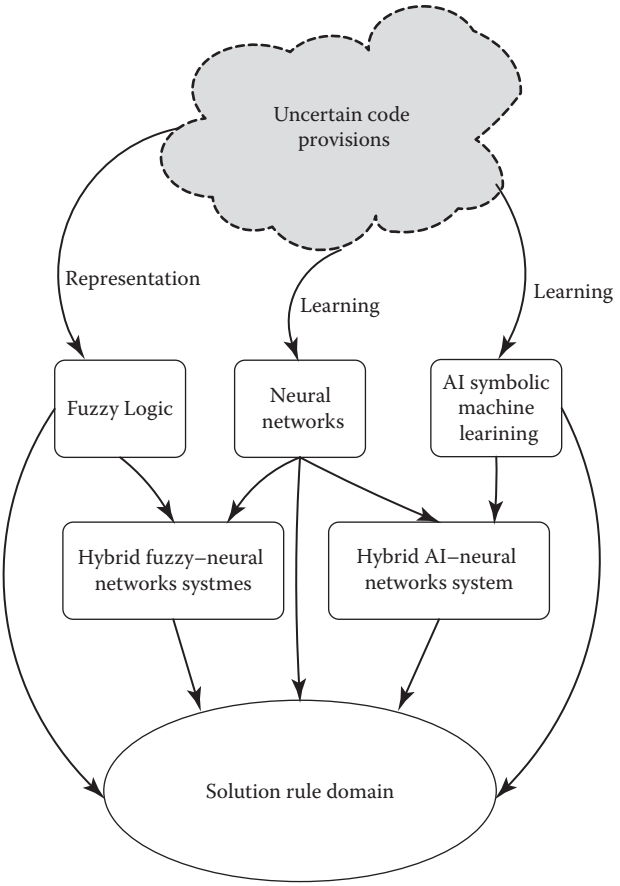


FIGURE 5.3 Framework for lower-order level of code provisions.

AEC industry require automated model-checking methods that are as practical and simple as possible. Thus, the proposed framework targets simplicity and uses combinations of principles from model information management and buildingSMART standards. This framework enables both an overview and an insight into the nature of building regulations, as well as enabling a predictable implementation into a BIM environment.

The proposed framework aims at the practicability of methods by which manual and semiautomatic procedures for transforming building regulations into a computable model are emphasized. The specifications of the computable rules can then be encoded into software and implemented as efficient processes for the quality control of building design. To demonstrate the suggested framework, FBC 2014 is considered as an example. The initial step for the development of a high-order MVD is to understand the requirements stated in the US National BIM Standard (NBIMS-US). According to the NBIMS-US specification, the process consists of two major steps:

1. Developing the Information Delivery Manual (IDM), which aims to provide the integrated reference for the process and data required by BIM by identifying the discrete processes undertaken within building construction, the information required for their execution, and the results of that activity (Nawari, 2012a).
2. Translating the IDM into an implementable specification for software applications, depending on a proper set of standard data models defined through the MVD. The MVD offers the standard stating how the information must appear in the IFC schema.

The execution of these steps yields an IFC model (based on ISO 16739:2013) that is more powerful and closely reflects real building code requirements, accelerates the integration of this framework into the NBIMS-US, and contributes to an open environment for software application developers and AEC professionals.

INFORMATION DELIVERY MANUAL

The IDM is the document that defines the necessities to set up BIM models for several building code regulations. The IDM provides the foundation for standardized data exchange. The main objectives of the IDM include the following (Nawari, 2012a):

1. Describe the workflow within a building project life cycle for which different trades necessitate information exchange.
2. Define the outcomes of process implementation that can be utilized in succeeding steps.
3. Identify the actors transmitting and accepting information within the process.
4. Assure that definitions, specifications, and descriptions are delivered in a form that is valuable and effortlessly understood by the target group.

The development of IDM for building code specifications starts with a description of the data exchange–functional requirements and workflow situations for interactions between BIM model data and the requirements specified in building codes. This is demonstrated in the process map displayed in Figure 5.4.

The process map was generated utilizing standard Business Process Modeling Notation (BPMN) as part of the IDM specifications (Nawari, 2014). These notations are illustrated briefly in Figure 5.5.

Moreover, the IDM utilizes notation for information exchanges between activities called *exchange models* (Figure 5.4). Each exchange model is distinctively recognized across all use cases and besides its name caries condensed designation of the use case it associates with, as follows:

- CODE_A_EM: Building Code Regulations and Architectural Design use case exchange models
- CODE_S_EM: Building Code Regulations and Structural Design use case exchange models

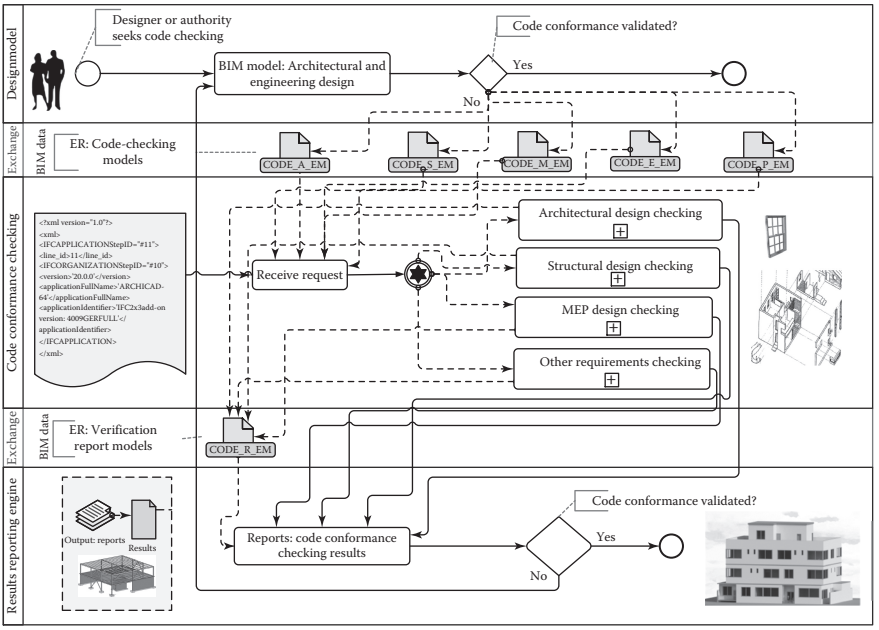


FIGURE 5.4 General process map for code conformance checking.

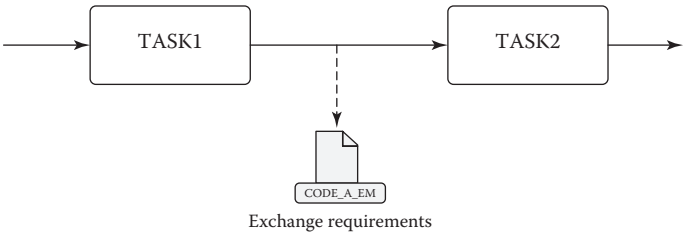


FIGURE 5.5 Process modeling notations for IDM.

- CODE_M_EM: Building Code Regulations and Mechanical System Design use case exchange models
- CODE_E_EM: Building Code Regulations and Electrical Design use case exchange models
- CODE_P_EM: Building Code Regulations and Plumbing Design use case exchange models
- CODE_R_EM: Building Code Regulations and Results Reporting use case exchange models

In the process diagram depicted in Figure 5.4, the following tasks and processes are identified.

- BIM model containing architectural and engineering systems designs
- Code conformance–checking processes for architectural design
- Code conformance–checking processes for structural design
- Code conformance–checking processes for mechanical, electrical, and plumbing (MEP) design
- Other building regulations conformance–checking processes
- Results-reporting process

Some of these main tasks may have various subprocesses that can also be defined using process maps and IDMs. For example, the code conformance–checking processes for the architectural design process may involve the following subtasks:

- Checking of allowable building areas and heights
- Checking of means of egress
- Verifying accessibility
- Verifying roofs and roof assemblies
- Checking requirements for elevators and conveying systems
- Checking detailed use and occupancy requirements

In each of these processes, the BIM model requirements have to be established and stated according to the BIM standard procedures.

The information exchange (IE) between these tasks and subprocesses is attained via *exchange models*. These exchange models describe the functional content of the data to be communicated in a typical building code conformance–checking case—for example, the geometrical properties that will be used for checking conformance with the building codes regarding spatial areas, height, and clash detection.

The more information specified in the exchange model, the greater the quality of the exchange data model. This in turn will lead to adequate information being employed in the MVDs (Nawari, 2012a). From the data defined in the IDM, an MVD is developed for each aspect and describes precisely how it is to be implemented in the IFC exchange.

The IFC data structure comprises a wide spectrum of data as it covers the whole life cycle of a building and its environment. For efficiency purposes, each code check must only deal with a subset of the entire IFC schema to avoid processing overpowering quantities of data. Thus, MVDs are to be developed as the principal tool for describing model subsets that are pertinent to the data exchange between the actors involved. The objective is that software vendors only deal with the parts of the IFC schema that are related to the particular field.

These phases and levels of delineations must also be developed for all subprocesses within the building code conformance realm. In this chapter, most of the efforts are centered on developing the IDM and MVD for checking building regulations related to a section of FBC 2014.

In order to develop the IDM (high-order level 1 development), the source regulation information needs to be classified as depicted in Figure 5.1. FBC 2014 is considered the source example for the implementation of the proposed framework (Table 5.2).

TABLE 5.2
Classification of FBC 2014, chapter 5, "General Building Heights and Areas"

Section	Classification	Concepts and Attributes	Dependencies
501	Content (Cont)	General scope; identification	
502	Content (Cont)	Definitions	
503.1	Dependent (Dep)	Building height and areas limits; fire walls	Table 503; Section 602; Section 706
503.1.1	Provisory (Prov)	Special industrial buildings; unlimited area; unlimited height	
503.1.2	Dependent (Dep)	Buildings on the same lot	Table 503; Section 504 and Section 506.
503.1.3	Dependent (Dep)	Buildings of Type I construction; unlimited area; unlimited height	Section 504.2; Section 504.31.1; Section 504.3.
504.1	Dependent (Dep)	Building height limits exceptions; automatic sprinkler system; automatic fire-extinguishing system; public ways or yards	Table 503; Section 504.2 and Section 504.3
504.2	Dependent (Dep)	Automatic sprinkler system; maximum height; building area; group R buildings; exceptions	Section 506.2 and Section 506.3;
504.3	Provisory (Prov)	Roof structures; unlimited height	
505.1	Content (Cont)	Mezzanines and equipment platforms	
505.2	Dependent (Dep)	Mezzanines; built areas; mezzanine area; mezzanine height; clear height above and below mezzanine floor	Section 503.1.
505.2.1	Dependent (Dep)	Allowable mezzanine area; room floor area; equipment platform; exceptions; Type I, II, and II construction; special industrial buildings; automatic sprinkler system; emergency voice/alarm communication system; group S2 occupancy	Section 503.1.1;
505.2.2	Content (Cont)	Means of egress for mezzanine	
505.2.3	Cont & Prov	Mezzanine openness; exceptions; means of egress; control equipment; automatic sprinkler system	
505.3	Cont & Dep	Equipment platforms area; building area; no. of storeys; fire area; mezzanine; walkways; stairs; alternating tread devices; means of egress	Sections 504.2, 506.2, 506.3, 507.
505.3.1	Provisory (Prov)	Area limitation	
505.3.2	Content (Cont)	Automatic sprinkler system	
505.3.3	Content (Cont)	Guards, equipment platforms	
506.1	Provisory (Prov)	Building area increase	

(*Continued*)

TABLE 5.2 (CONTINUED)
Classification of FBC 2014, chapter 5, "General Building Heights and Areas"

Section	Classification	Concepts and Attributes	Dependencies
506.2	Cont & Prov	Frontage; building area increase; public way	
506.2.1	Cont & Prov	Building width limits; exception	
506.2.2	Content (Cont)	Open space limits	
506.3	Provisory (Prov)	Automatic sprinkler system increase; exception	
506.4	Content (Cont)	Single-occupancy buildings with more than one storey; exception	
506.4.1	Prov & Dep	Allowable building area; exception	
506.5	Cont & Prov	Allowable building area for mixed occupancy	
506.5.1	Dependent (Dep)	Building area, one-storey building	Section 508.1
506.5.2	Dependent (Dep)	Building area, multistorey building	Section 508.1
507.1	Cont & Prov	Unlimited-area buildings; exceptions	
507.2	Provisory (Prov)	Non-sprinklered one-storey building; unlimited area; unlimited area	
507.3	Provisory (Prov)	Sprinklered one-storey building; unlimited area; exception	
507.3.1	Dependent (Dep)	Mixed-use occupancy buildings with group A-1 and A-2; unlimited area	Section 503.1; Section 508.4.4
507.4	Provisory (Prov)	Sprinklered two-storey building; unlimited area	
507.5	Dependent (Dep)	Reduced open space; public ways width; unlimited area	Sections 507.2, 507.3, 507.4, 507.6.
507.6	Provisory (Prov)	Group A-3 buildings of Type II construction; unlimited area; one-storey building	
507.7	Provisory (Prov)	Group A-3 buildings of Type II construction; unlimited area; one-storey building	
507.8	Dependent (Dep)	Group H occupancies; unlimited area	Sections 507.3, 507.4, 508.1, 508.2, 508.3, 508.4
507.8.1	Dependent (Dep)	Allowable area; group H occupancies; perimeter; group H floor area; unlimited area	Section 506.2, Table 503
507.8.1.1	Provisory (Prov)	Aggregate floor area; group H occupancies	
507.8.1.1.1	Provisory (Prov)	Liquid use; dispensing and mixing rooms; unlimited area	Florida Fire Prevention Code and NFPA 30

(*Continued*)

TABLE 5.2 (CONTINUED)
Classification of FBC 2014, chapter 5, "General Building Heights and Areas"

Section	Classification	Concepts and Attributes	Dependencies
507.8.1.1.2	Dependent (Dep)	Liquid storage rooms; floor area	Florida Fire Prevention Code and NFPA 30
507.8.1.1.3	Dependent (Dep)	Spray paint booths; unlimited area	
507.8.2	Dependent (Dep)	Located on building perimeter; group H occupancies; exterior wall; unlimited area	Section 507.8.1.1
507.8.3	Provisory (Prov)	Occupancy separation; group H occupancy; unlimited area	
507.8.4	Dependent (Dep)	Height limitations; two storeys; unlimited area; allowable height	
507.9	Dependent (Dep)	Aircraft paint hangar area; public ways; yards; unlimited area; building height	
507.10	Dependent (Dep)	Group E buildings; one storey; Type II, IIIA, IV; unlimited area	Section 1020
507.11	Provisory (Prov)	Area of motion picture theater; Type II construction; public ways and yards; automatic sprinkler system	
507.12	Dependent (Dep)	Area of covered and opened mall buildings; area of anchor buildings; unlimited area	

TABLE 5.3
Data Exchange Model Description

Building Code Regulations and Architectural Design

Type	Task
Name	CODE_A_EM
Version	1.0
Author	N. Nawari, School of Architecture College of Design, Construction and Planning, University of Florida, P.O. Box 115702, 1480 Inner Road, Gainesville, FL 32611-5702 Email: nnawari@ufl.edu
Documentation	Designer or building authority uses building code specifications to examine the compliance of the architectural model with specific code regulations. The building code used in this exchange mode is FBC 2014, Chapter 5. The exchange model objective is to provide feedback on allowable building heights and areas.

The range of the requisite exchange is the exchange of information about architectural spaces and height. The framework example focuses on the data exchange requirements (ERs) for the building code compliance checking between the BIM model and the code conformance–auditing model. The ER mode is described in Tables 5.3 and 5.4 using FBC 2014. The purpose of the exchange model description is to record the

TABLE 5.4
Part of the Information Exchange Requirements for FBC 2014

Type of Information	Information Needed	Required (Req)	Optional (Opt)	Req/ Opt	Data Type	Units
Building Code Identifiers	Name	X			String	n/a
	Year	X			Date	n/a
	Version	X			String	n/a
	Section Number	X			Double	n/a
	Section Title	X			String	n/a
	Clause	X			String	n/a
Classification Attributes	Content (Cont)			X	String	n/a
	Provisory (Prov)			X	String	n/a
	Dependent (Dep)			X	String	n/a
	Ambiguous (Amb)			X	String	n/a
Regulation Attributes	Allowable Area	X			Double	ft2; m2
	Allowable Height	X			Double	ft; m
	Areas limits	X			Double	ft2; m2
	Occupancy Type	X			String	n/a
	Building Group	X			String	n/a
	Building Area	X			Double	ft2; m2
	Building Height	X			Double	ft; m
	Building Perimeter	X			Double	ft; m
	Sprinklered Area		X		Yes/No	n/a
	Aggregate Area		X		Double	ft2; m2
	Unlimited Area		X		Yes/No	
	Equipment Platforms Area		X		Double	ft2; m2
	No. of Stories	X			Integer	n/a
	Fire Area		X		Double	ft2; m2
	Mezzanine		X		Yes/No	n/a
	Mezzanine Area		X		Double	ft2; m2
	Walk Ways Width		X		Double	ft; m
	Alternating Tread Devices		X		Yes/No	n/a
	Means of Egress		X		String	n/a
	Mixed Occupancy		X		Yes/No	n/a
	Reduced Open Space		X		Yes/No	n/a

(*Continued*)

TABLE 5.4 (CONTINUED)
Part of the Information Exchange Requirements for FBC 2014

Type of Information	Information Needed	Required (Req)	Optional (Opt)	Req/ Opt	Data Type	Units
	Liquid Storage Rooms		X		Yes/No	n/a
	Area of Motion Picture Theater		X			
	Type of Construction	X			String	n/a
	Automatic Sprinkler System		X		Yes/No	n/a
	Area of Covered and Opened Mall Buildings		X			
	Area of Anchor Buildings		X			
	Liquid Use Room		X		String	
	Dispensing and Mixing Room		X			
	Spray Paint Booths		X		Yes/No	n/a
	Control Area	X			Yes/No	n/a
	Fire Wall		X		Yes/No	n/a
	Doweling		X		Yes/No	n/a
	Exit Access		X		Yes/No	n/a
	Exit Discharge		X		Yes/No	n/a
	Opened Mall Building		X		Yes/No	n/a
	Covered Mall Building		X		Yes/No	n/a
	Lowest Floor	X			Yes/No	n/a
	Exterior Wall	X			Yes/No	n/a
	Exterior Wall Envelope	X			String	
	Dispensing Room		X		Yes/No	
	Mixing Room		X		Yes/No	
	Liquid Use		X		Yes/No	
	Type I Construction		X		Yes/No	
	Type II Construction		X		Yes/No	
	Type IIII Construction		X		Yes/No	

(*Continued*)

TABLE 5.4 (CONTINUED)
Part of the Information Exchange Requirements for FBC 2014

Type of Information	Information Needed	Required (Req)	Optional (Opt)	Req/ Opt	Data Type	Units
	Type IV Construction		X		Yes/No	
	Type V Construction		X		Yes/No	
	Group A		X		Yes/No	
	Group B		X		Yes/No	
	Group C		X		Yes/No	
	Group D		X		Yes/No	
	Group E		X		Yes/No	
	Group F		X		Yes/No	
	Group G					
	Group H					
	Group I					
	Area Increase		X			
	Public Way		X		Yes/No	

extend and range of the exchange requirements. It defines an overview of the ERs and is used as a general guide in the process of defining the ER model.

MODEL VIEW DEFINITION

The IFC schema encompasses a wide range of data objects since it deals with the whole life cycle of a building and its environment. Thus, it is recommended that each discipline domain should only consider a subset of the full IFC schema to avoid processing overwhelming amounts of data. An MVD is developed as the tool for creating model subsets that are pertinent to the specific data exchange between domain application types. MVD diagrams describe the MVD concepts that will be used in the data exchange, as well as the schema and relationships between these concepts. The concluding phase in MVD development concerns the implementation guidance specifications, which define the IFC entities used to exchange each concept as required by the IFC schema.

The process of developing the MVDs counts on the description of the information ERs in the IDM and how they will be utilized with respect to both domain users and software developers. From this information, the MVD is established for each attribute and describes how it is to be handled in the IFC schema. In essence, the MVD offers the specification for IFC-based data exchange implementation, and enables certification tests of how well implementations conform to this specification (Figure 5.6).

MVD development takes place in two phases (Figure 5.7). The initial phase deals with IFC-independent concept diagrams. The second phase focuses on creating IFC-dependent concept diagrams. The IFC-independent MVD approach offers the methodological solution of an exchange requirement. It provides the necessary data structure that can be supported by a software application for the exchange of data for

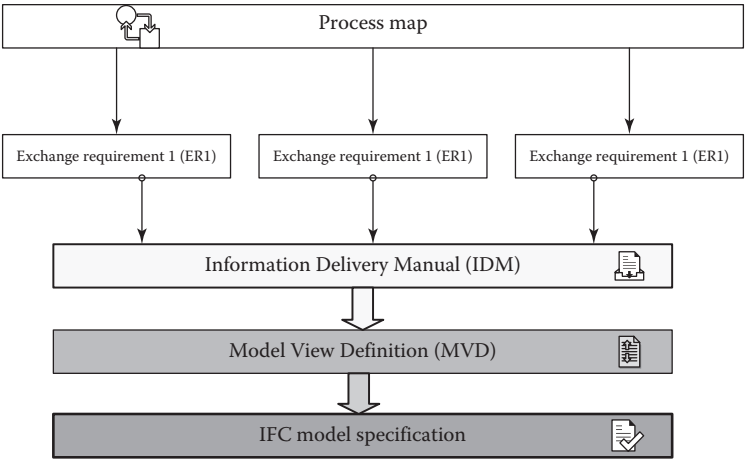


FIGURE 5.6 Overview of the process of developing IFC specification.

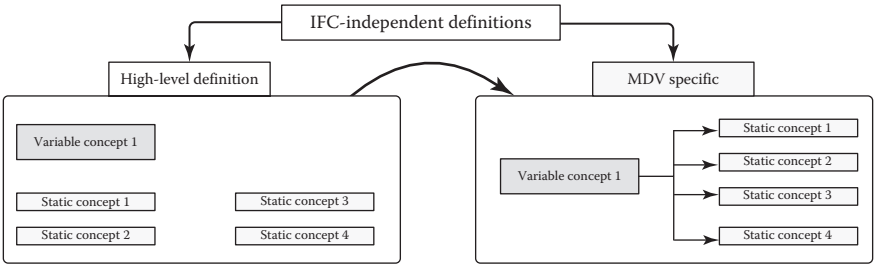


FIGURE 5.7 IFC-independent definitions for MVD.

the specific task, and satisfies all the requirements for supporting a project workflow according to the rules and methods specified in that field.

An MVD is an exchange requirement model. It is composed of exchange concepts and their relationships, describing the exchange of information for a specific domain (Figure 5.6). An exchange concept focuses on the individual actions that are carried out within a specific domain. An ER may have numerous ER models as technical solutions.

Each concept gives technical details about the information that should be exchanged as a result of the action. The specific action can take place within various ERs. The IDM provides the basis for the list of exchange concepts that can then be compiled to provide the ER model schema. Thus, an exchange concept could similarly relate to various ERs. Because of this, exchange concepts are specifically designed to be reusable within many ERs. However, certain exchange concepts concern primarily general ideas and may be expected to repeat frequently. Examples include exchange concepts dealing with geometric shape representations.

Exchange concepts give details about an action. They describe the use of each entity, each attribute, every property set, and every property concerned. On the other hand, ERs define information in a more readable format. Due to the nature of the details, exchange concepts can also be subdivided into other sub-exchange concepts.

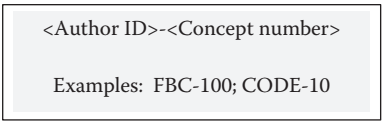


FIGURE 5.8 Format of the concept ID.

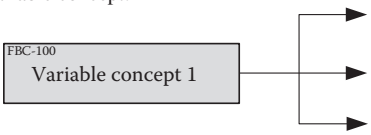
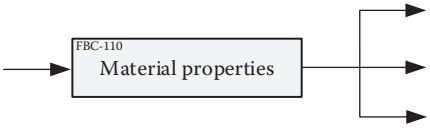
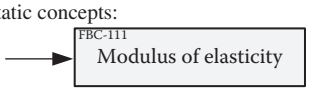
The MVD concept diagram enables the clear definition and reuse of data ERs as described in the IDM. It should display unambiguous requirement specifications for the data exchanges to be supported in software applications. Each concept in the MVD diagram has an ID that uniquely identifies the concept. The format for depicting this information is shown in Figure 5.8.

Further concept symbols used in the MVD are described in Table 5.5.

Figure 5.9 illustrates an MVD (architectural design to building regulations) demonstrating part of the exchanges for code checking (general building heights and areas regulations of FBC 2014) and architectural design.

This proposed MVD provides the basis for developing an MVD covering other parts of building regulations and standards. This will enable high-quality IFC implementations that satisfy automatic or semiautomatic code conformance checking.

TABLE 5.5
MVD Concept Symbols

Symbol	Description
Variable concept: 	The variable concept can be used in different MVDs; however, their content may vary. The variable concept is normally connected to other concepts such as group and static concepts. Examples include load, space, wall, etc.
Group concepts: 	Group concepts define the structure of the concept diagram by grouping together static concepts and/or other group concepts. Example include: wall geometry, material properties, etc.
Static concepts: 	Static concepts provide a constant status. They remain the same in all circumstances in which they are utilized. They don't possess any options, and thus can be reused without changes. Examples include modulus of elasticity, yield stress, height of the building, etc.
Comment: 	The is a text box with a yellow background for inserting any comments in the MVD diagram.
Note: 	A note can be attached to a concept using an outlined text box.

IFCXML

Extensible Markup Language (XML) is a text format derived from Standard Generalized Markup Language (SGML; ISO 8879). It is characterized by simplicity and flexibility. Originally, the language was designed to meet the needs of large-scale electronic document publishing. Currently, XML is playing an increasingly vital role in the exchange of a wide variety of business data.

The implementing of the IFC standard using XML technologies is known as ifcXML. It is an extension to the existing IFC data format and focuses on the specifics of the ifcXML specification, unlike the standard EXPRESS-based IFC object data model. The ifcXML data files are normally given the extension ".xml," or alternatively, ".ifcxml."

Offering an XML representation of IFC data will allow a wide range of applications to access a unified standardized schema representing the built environment and related resources. XML has a broader range of supporting technologies and database implementations and is currently being used in most business transactions. XML is also supported by many Web browsers, making the information immediately accessible on PC and most other mobile computing devices. Being able to consume the IFC content using the semantics and syntax of XML provides many benefits, such as the extraction, transmission, and merging of partial BIM model data for various applications.

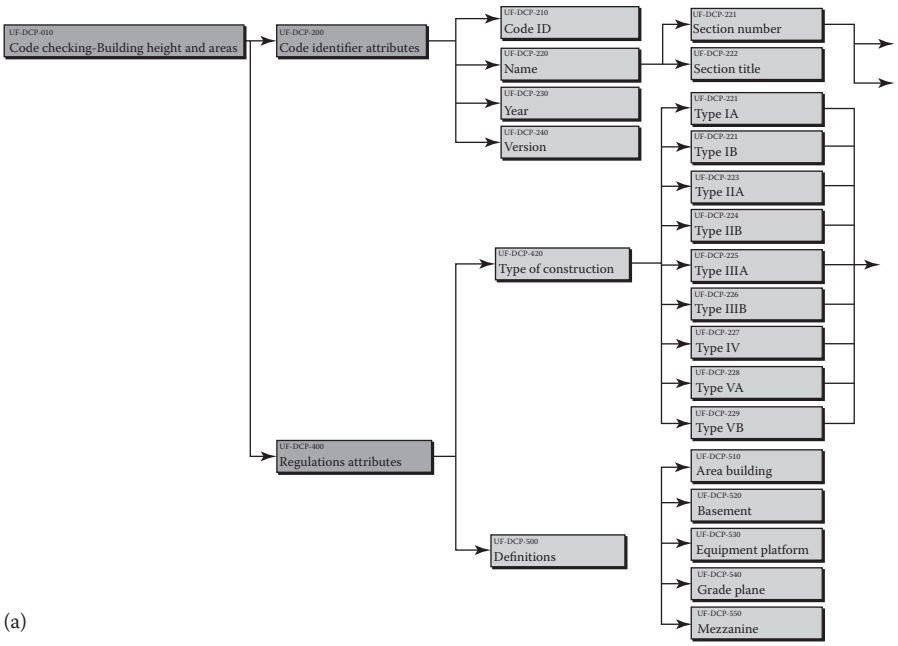


FIGURE 5.9 (a–c) MVDs for parts of the exchanges between architectural design and building regulations.

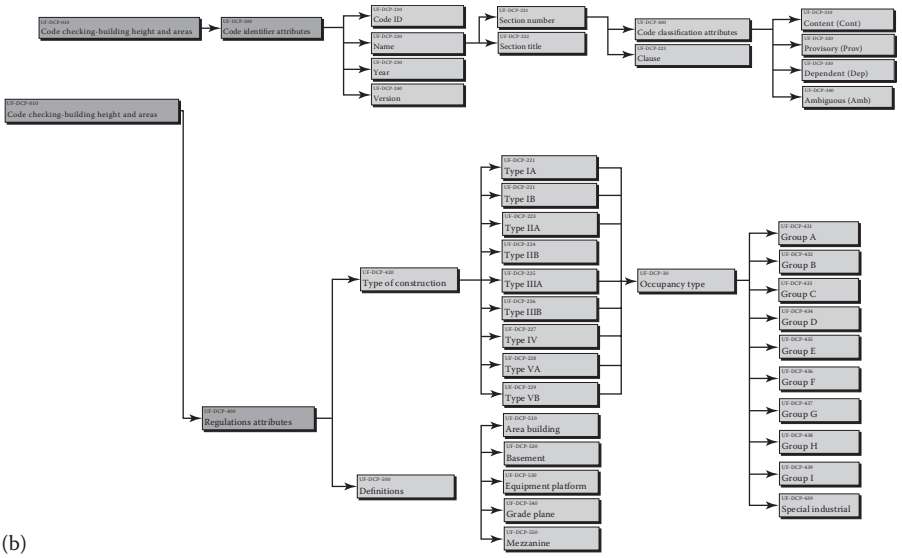


FIGURE 5.9 (Continued)

In terms of international standards, the official ISO 10303-28 standard includes a configuration capability that enables schema developers to reduce some of the overheads stemming from converting an EXPRESS schema (such as IFC) into an XML schema. Currently, the IFC data sets are converted into the new ifcXML 4 structure according to the conversion rules that are subsets of the configured ISO10303-28 XML language binding of EXPRESS-based schemas and data. These are produced as a conformation file that adheres to Chapter 10 of ISO 10303-28. The identical

configuration file is utilized to create the ifcXML XSD and the actual XML data files (Liebich and Wise, 2012).

The XML Schema Definition (XSD) states how to legally describe the elements in an XML document. Also, XSD can be utilized to define a set of rules to which an XML document must conform in order to be considered "valid" according to that schema. In IFC content, the XSD is normally generated from the IFC source definition. Figure 5.10 illustrates the process of generating ifcXML data. An example conversion of IFC EXPRESS data into ifcXML data is depicted in Figure 5.11.

The XML Linking Language (XLink) is a specification for any data elements to be inserted into XML files in order to establish and define links between resources. It uses XML syntax to create schema that can describe connections similar to the

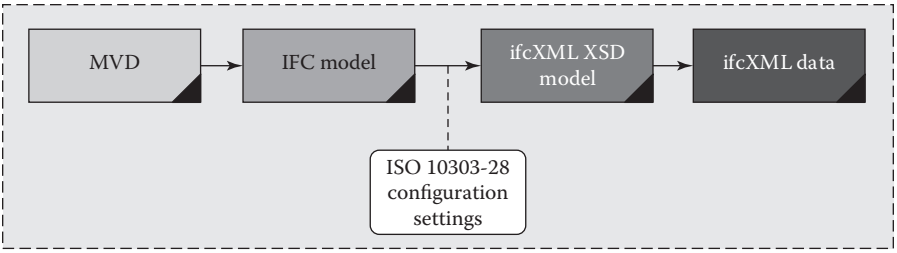


FIGURE 5.10 The general process of creating an ifcXML data file.

EXPRESS:

```
ENTITY IfcActorRole;
   Role : IfcRoleEnum;
   UserDefinedRole : OPTIONAL IfcLabel;
   Description : OPTIONAL IfcText;
END_ENTITY;
```

XSD:

```
<xs:element name="IfcActorRole" type="ifc:IfcActorRole"
   substitutionGroup="ifc:Entity" nillable="true"/>
<xs:complexType name="IfcActorRole">
   <xs:complexContent>
      <xs:extension base="ifc:Entity">
         <xs:attribute name="Role" type="ifc:IfcRoleEnum" use="optional"/>
         <xs:attribute name="UserDefinedRole" type="ifc:IfcLabel" use="optional"/>
         <xs:attribute  name="Description" type="ifc:IfcText" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

IFC File:

```
#20000= IFCACTORROLE(.ENGINGEER., $, 'engineer of record');
```

ifcXML File:

```
<IfcActorRole id="i1546" Role="engineer"   Description="engineer of record"/>
```

FIGURE 5.11 Example of IFC data conversion into ifcXM. (Modified from Liebich, T. and Wise, M., Simple ifcXML, 2012.)

simple unidirectional hyperlinks of classical HTML links, as well as more detailed links.

FRAMEWORK

After transforming the building code regulations into a standardized ifcXML data model, the entire domain will be available in ifcXML data format. This environment is now suitable for applying the Automated Code Conformance Checking (ACCC) framework proposed by Nawari (2012c,d). This framework is grounded on several information technologies. Specifically, these are ifcXML and Language-Integrated Query (LINQ). The framework is depicted in Figure 5.12.

LINQ is the term for a set of technologies based on the integration of query capabilities directly into conventional programming languages such as C# and VB.Net. It is a part of the Microsoft .Net framework, which permits query expressions to take advantage of the rich metadata, static typing, and IntelliSense, and compile time syntax checking (Nawari, 2012c). Moreover, LINQ facilitates a unified general-purpose declarative query platform to be used for all in-memory data, not just data from external sources. The .NET LINQ provides a set of standard, general-purpose query operators that perform various types of database operations such as traversal, filter, and projection to be expressed in a direct yet declarative way in any programming language. This allows queries to be applied to any object whose type implements the *IEnumerable<T>* interface or the *IQueryable<T>* interface-based information source. In other words, LINQ has two sets of query operators: one that operates on objects of the type *IEnumerable<T>* and the other that operates on objects of the type *IQueryable<T>*, which can be called by using either static method syntax or instance method syntax.

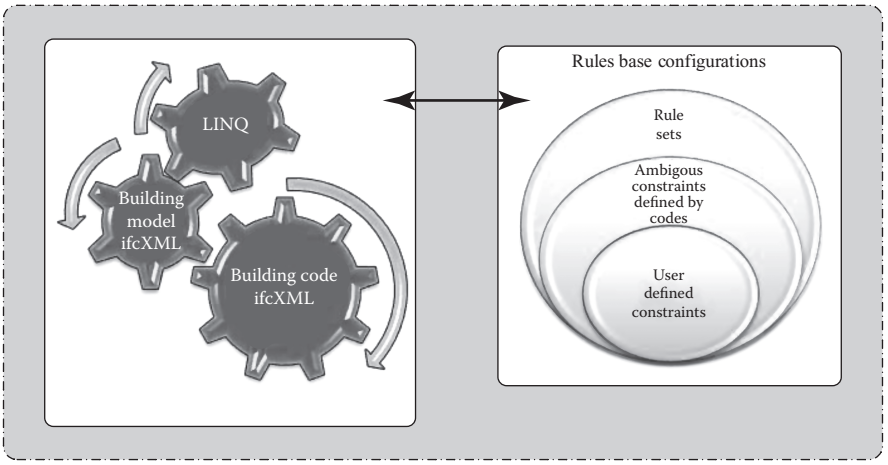


FIGURE 5.12 Automated Code Conformance Checking (ACCC) framework. (Modified from Nawari, N. O., *Journal of Architectural Engineering*, 18(4), 2012.)

LINQ allows anyone to work with XML from within the .Net framework programming languages. Furthermore. LINQ enables any industry to expand the set of standard query operators with new domain-specific operators that are suitable for that domain. Any industry is also free to replace the standard LINQ query operators with its own implementations that suit its needs. By following the LINQ pattern specifications, such implementations exhibit the same language incorporation and tool support as the standard Microsoft .Net query operators (Figure 5.13).

The proposed framework is centered on LINQ- to ifcXML-based data. It is essentially a LINQ-enabled, in-memory XML programming interface that facilitates communication with ifcXML. The beauty of the query architecture used in LINQ is that it provides implementations that work over both XML and SQL databases. The query operators over XML (LINQ to XML) are characterized by their ease of use and efficiency in providing XPath/XQuery functionality in the host programming language. This incorporation offers robust typing over relational data objects.

Further advantages of LINQ to XML include the ability to use query results as parameters and the *XElement* and *XAttribute* object constructors are an effective method to establish XML trees (Figure 5.14). This technique is often referred to as *functional construction*, which enables one to easily transform XML trees from one shape to another. Moreover, LINQ to XML offers an enhanced XML programming interface with the following features (Microsoft, 2017a):

1. Load XML from files or streams.
2. Serialize XML to files or streams.
3. Create XML from scratch by using functional construction.
4. Query XML using XPath-like axes.
5. Manipulate the in-memory XML tree by using methods such as *Add*, *Remove*, *ReplaceWith*, and *SetValue*.
6. Validate XML trees using XSD.

```
IEnumerable<XElement> WallHeight =

from item in ifcWalls.Descendanats("Item")

where (int) item.Element("Walls") *

  (decimal) item.Element("Height") > 12
```

FIGURE 5.13 An example of LINQ to XML showing a list, sorted by wall height, of the heights with a value greater than 12 ft.

```
<?xml version="1.0" encoding="utf-8" ?>
<IFcMaterial id="123" Name="Concrete">
  <Description>Content</ Description >
</IFcMaterial>
```

FIGURE 5.14 Example of ifcXML.

The process begins with loading an ifcXML document into a data store, such as DOM or LINQ to XML, then using a programming interface to extract data, insert nodes, delete nodes, or change the content of nodes, and then save the ifcXML to a file or transmit it over any type of network. LINQ to XML has also the advantage of enabling another approach known as *functional construction* (FC), which can be useful in many applications. FC deals with data manipulations as an issue of transformation rather than as an exhaustive modification of a database (Microsoft, 2017b). With this approach, one can take a representation of data and transform it efficiently from one form to another. In many cases, one can generate a transformational code in a fraction of the time that it would take to manipulate a typical database, and that created code is generally more robust and easier to understand and sustain. In such cases, although the transformational approach can take additional processing resources, it is a more efficient method of manipulating data.

The following example illustrates the difference between the conventional procedural approach and the FC approach to modifying XML data files. In this example, the attributers in the XML file are to be changed into elements.

One can generate conventional procedural code to create elements from attributes and then delete the attributes, as depicted in Figure 5.15.

On the other hand, an FC approach involves creating a code for a new tree, deciding which elements and attributes to choose from the source tree, and transforming them as needed as they are added to the new tree. The FC code can be generated as shown in Figure 5.17.

The code shown in Figure 5.17 outputs the same results as produced by the example depicted in Figure 5.16. However, there is a clear difference in terms of maintenance and clarity between the two approaches. One can now actually see the resulting structure of the new XML in the FC approach as depicted in Figure 5.17. One can easily see the code that creates the *Root* element, the code that extracts the *Description* element from the source tree, and the code that transforms the attributes from the source tree to elements in the new tree.

```
XElement root = XElement.Load("IFcData.xml");
foreach (XAttribute att in root.Attributes()) {
    root.Add(new XElement(att.Name, (string)att));
}
root.Attributes().Remove();
Console.WriteLine(root);
```

FIGURE 5.15 Conventional LINQ-to-XML example.

```
<IFcMaterial>
  <Description>Content</ Description >
  <id>123</id>
  <Name>Concrete</Name>
</IFcMaterial>
```

FIGURE 5.16 Output of the code generated in Figure 5.15.

```
XElement root = XElement.Load("IFcData.xml");
XElement ifcTree = new XElement("IFcMaterial",
    root.Element("Description"),
    from att in root.Attributes()
    select new XElement(att.Name, (string)att)
);
Console.WriteLine(ifcTree);
```

FIGURE 5.17 FC approach.

The FC example produces a code that is not any shorter than the procedural approach depicted in Figure 5.15. Nevertheless, if one has to do many changes to create an XML tree, the conventional LINQ-to-XML approach can become fairly complex. In contrast to that approach, using the FC method allows one to create from the given ifcXML data file a subset of data using embedded queries and expression and present it in the desired format. The FC approach is certainly much easier to maintain and allows for greater developer productivity.

In summary, the preceding example is a simple one; however, it serves to illustrate the difference in programming philosophy between the two approaches. The FC method seems to yield superior productivity gains for transforming larger ifcXML files.

EXAMPLE

By implementing the framework as described, the following automatic checking can be executed to examine FBC 2014, Chapter 5, "General Building Heights and Areas." The example shown below is the case of checking building areas and height. The IFC objects involved in this example are illustrated in Figure 5.18.

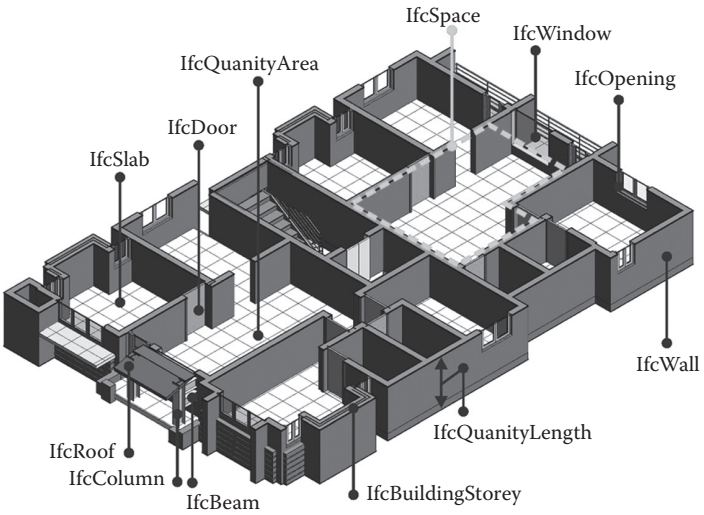


FIGURE 5.18 Main IFC objects referenced in the example.

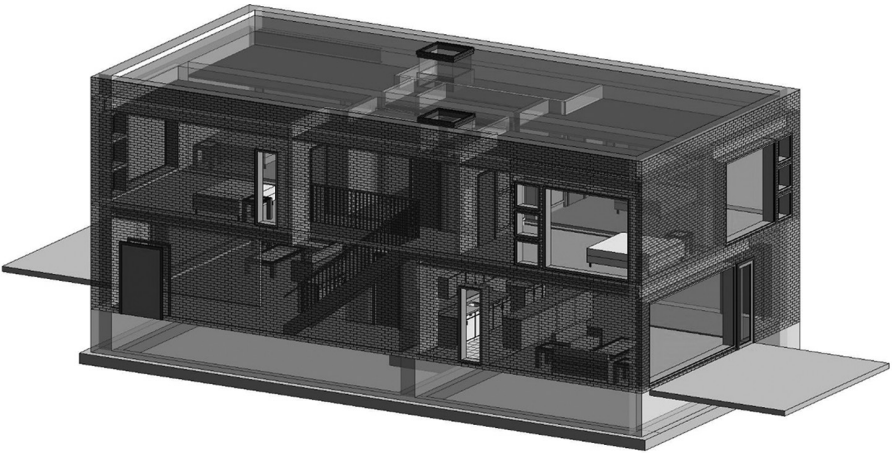


FIGURE 5.19 Building model used for code compliance checking.

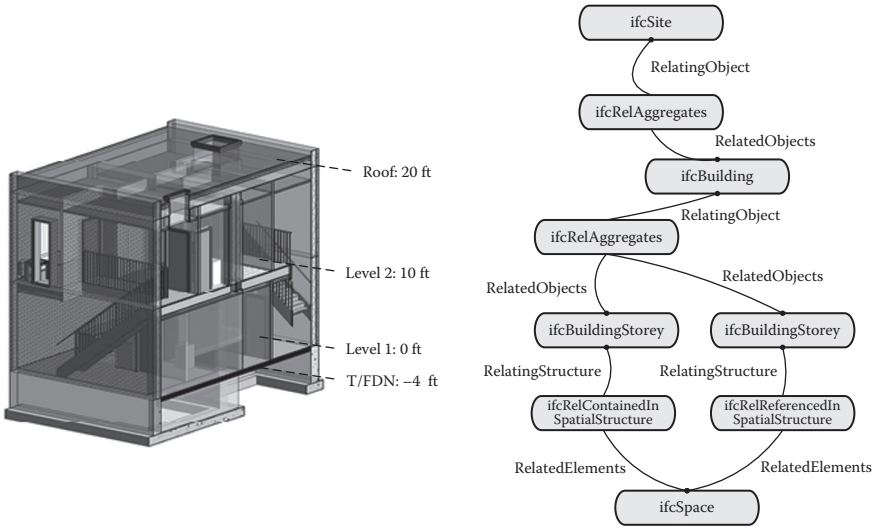


FIGURE 5.20 Relevant IFC objects for code conformance checking.

The building used for this example is the duplex apartment model, which is from the BuildingSMART Alliance website (this BIM model was originally created by a student who developed this building as part of a design competition). The original model has been modified to simplify the dimensions and areas for this example (Figure 5.19). Part of the ifcXML file for this model is illustrated in Figure 5.20.

To understand IFC for this part of the code checking, the main objects required are depicted in Figure 5.20 and Table 5.6. The figure gives an overview of the IFC objects that span different storeys in a building. This will aid in making the code-checking process more efficient when using the proposed framework. By identifying

TABLE 5.6
Quantity Use Definition for ifcSpace (buildingSMART International, IFC2×3)

Name	Value Type	Description
NominalHeight	ifcQuantityLength	Floor height to ceiling height (without suspended ceiling) for this space (measured from top of slab of this space to bottom of slab of space above); the average shall be taken if room form is not prismatic.
ClearHeight	ifcQuantityLength	Clear height between floor level (including finish) and ceiling level (including finish and subconstruction) of this space; the average shall be taken if room shape is not prismatic.
GrossPerimeter	ifcQuantityLength	Computed gross perimeter at the floor level of this space. It is all sides of the space, including those parts of the perimeter that are created by virtual boundaries and openings. The exact definition and calculation rules depend on the method of measurement used.
NetPerimeter	ifcQuantityLength	Calculated net perimeter at the floor level of this space. It normally excludes those parts of the perimeter that are created by virtual boundaries and openings. The exact definition and calculation rules depend on the method of measurement used.
GrossFloorArea	ifcQuantityArea	Calculated sum of all floor areas covered by the space. It normally includes the area covered by elements inside the space (columns, inner walls, etc.). The exact definition and calculation rules depend on the method of measurement used.
NetFloorArea	ifcQuantityArea	Calculated sum of all usable floor areas covered by the space. It normally excludes the area covered by elements inside the space (columns, inner walls, etc.), floor openings, or other protruding elements. Special rules apply for areas that have a low headroom. The exact definition and calculation rules depend on the method of measurement used.
GrossWallArea	ifcQuantityArea	Calculated sum of all wall areas bounded by the space. It normally includes the covered area by elements inside the wall area (doors, windows, other openings, etc.). The exact definition and calculation rules depend on the method of measurement used.
NetWallArea	ifcQuantityArea	Calculated sum of all wall areas bounded by the space. It normally excludes the area covered by elements inside the wall area (doors, windows, other openings, etc.). Special rules apply for areas that have a low headroom. The exact definition and calculation rules depend on the method of measurement used.
GrossVolume	ifcQuantityVolume	Calculated gross volume of all areas enclosed by the space (normally including the volume of construction elements inside the space). The exact definition and calculation rules depend on the method of measurement used.
NetVolume	ifcQuantityVolume	Calculated net volume of all areas enclosed by the space (normally excluding the volume of construction elements inside the space). The exact definition and calculation rules depend on the method of measurement used.

the objects needed for the LINQ-to-XML approach, the code conformance checking can be accomplished in a more direct manner.

In this example, the *ifcSpace* functions as the main project breakdown and is essential to building the spatial structure of a building. The spatial structure components are connected together by utilizing the object relationship *ifcRelAggregates*. Furthermore, it provides additional functions such as serving as the spatial container for space-related elements. A space is (if specified) associated to a building storey (or, in the case of exterior spaces, to a site). A space can cover several connected spaces. Thus, a space group provides a collection of spaces

```
<IfcSite id="i28040">
      <GlobalId>1xS3BCk291UvhgP2a6eflN</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
      </OwnerHistory>
      <Name>Default</Name>
      <ObjectType></ObjectType>
      <ObjectPlacement>
            <IfcLocalPlacement xsi:nil="true" ref="i28039"/>
      </ObjectPlacement>
      <CompositionType>element</CompositionType>
      <RefLatitude exp:cType="list">
            <exp:integer-wrapper pos="0">41</exp:integer-wrapper>
            <exp:integer-wrapper pos="1">52</exp:integer-wrapper>
            <exp:integer-wrapper pos="2">27</exp:integer-wrapper>
            <exp:integer-wrapper pos="3">839999</exp:integer-wrapper>
      </RefLatitude>
      <RefLongitude exp:cType="list">
            <exp:integer-wrapper pos="0">-87</exp:integer-wrapper>
            <exp:integer-wrapper pos="1">-38</exp:integer-wrapper>
            <exp:integer-wrapper pos="2">-21</exp:integer-wrapper>
            <exp:integer-wrapper pos="3">-839999</exp:integer-wrapper>
      </RefLongitude>
      <RefElevation>0.</RefElevation>
</IfcSite>
<IfcBuildingStorey id="i2042">
      <GlobalId>1xS3BCk291UvhgP2dvNMKI</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
      </OwnerHistory>
      <Name>Level 1</Name>
      <ObjectPlacement>
            <IfcLocalPlacement xsi:nil="true" ref="i2041"/>
      </ObjectPlacement>
      <LongName>Level 1</LongName>
      <CompositionType>element</CompositionType>
      <Elevation>0.</Elevation>
</IfcBuildingStorey>
<IfcBuildingStorey id="i2048">
      <GlobalId>1xS3BCk291UvhgP2dvNMQJ</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
      </OwnerHistory>
      <Name>Level 2</Name>
      <ObjectPlacement>
            <IfcLocalPlacement xsi:nil="true" ref="i2047"/>
      </ObjectPlacement>
      <LongName>Level 2</LongName>
      <CompositionType>element</CompositionType>
      <Elevation>10.0</Elevation>
</IfcBuildingStorey>
```

FIGURE 5.21 (a–c) Parts of the ifcXML for the building used for code checking.

```
<IfcPropertySingleValue id="i10463">
      <Name>Level</Name>
      <NominalValue>
            <IfcLabel-wrapper>Level: Level 2</IfcLabel-wrapper>
      </NominalValue>
</IfcPropertySingleValue>
<IfcPropertySingleValue id="i10464">
      <Name>Area</Name>
      <NominalValue>
            <IfcAreaMeasure-wrapper>59.2290176</IfcAreaMeasure-wrapper>
      </NominalValue>
</IfcPropertySingleValue>
<IfcPropertySingleValue id="i10465">
      <Name>Elevation at Bottom</Name>
      <NominalValue>
            <IfcLengthMeasure-wrapper>2.795</IfcLengthMeasure-wrapper>
      </NominalValue>
</IfcPropertySingleValue>
<IfcPropertySingleValue id="i10466">
      <Name>Elevation at Top</Name>
      <NominalValue>
            <IfcLengthMeasure-wrapper>3.1</IfcLengthMeasure-wrapper>
      </NominalValue>
</IfcPropertySingleValue>
<IfcPropertySingleValue id="i10467">
      <Name>Perimeter</Name>
      <NominalValue>
            <IfcLengthMeasure-wrapper>43.4568</IfcLengthMeasure-wrapper>
      </NominalValue>
</IfcPropertySingleValue>
```

(a)

FIGURE 5.21 (Continued)

included in a building storey. A space may also be decomposed into components, where each component defines a partial space. This is defined by the composition type attribute of the supertype *ifcSpatialStructureElement*, which is understood as follows:

COMPLEX = space group; ELEMENT = space; PARTIAL = partial space

The various quantities relating to the *ifcSpace* are defined by the *ifcElementQuantity* and attached by the *ifcRelDefinesByProperties*. Table 5.6 lists some of these quantities along with their definitions.

Other property sets related to the *ifcSpace* object are described by the *ifcPropertySet* and connected by the *ifcRelDefinesByProperties* relationship. According to IFC 2 × 3, the following property set definitions are specific to the *ifcSpace* (buildingSMART, 2014).

- *Pset_SpaceCommon*: Common property set for all types of spaces
- *Pset_SpaceParking*: Specific property set for only those spaces that are used to define parking spaces by *ObjectType = "Parking"*
- *Pset_SpaceParkingAisle*: Specific property set for only those spaces that are used to define parking aisle by *ObjectType = "ParkingAisle"*
- *Pset_SpaceFireSafetyRequirements*: Common property set for all types of spaces to capture the fire safety requirements.

```
<IfcRelAggregates id="i28249">
	<GlobalId>15Z0v90RiHrPC200A6FoKR</GlobalId>
	<OwnerHistory>
		<IfcOwnerHistory xsi:nil="true" ref="i1944"/>
	</OwnerHistory>
	<RelatingObject>
		<IfcBuildingStorey xsi:nil="true" ref="i2048"/>
	</RelatingObject>
	<RelatedObjects exp:cType="set">
		<IfcSpace xsi:nil="true" ref="i2103"/>
		<IfcSpace xsi:nil="true" ref="i2512"/>
		<IfcSpace xsi:nil="true" ref="i2777"/>
		<IfcSpace xsi:nil="true" ref="i3182"/>
		<IfcSpace xsi:nil="true" ref="i3531"/>
		<IfcSpace xsi:nil="true" ref="i3868"/>
		<IfcSpace xsi:nil="true" ref="i4117"/>
		<IfcSpace xsi:nil="true" ref="i4520"/>
		<IfcSpace xsi:nil="true" ref="i4854"/>
		<IfcSpace xsi:nil="true" ref="i5088"/>
	</RelatedObjects>
</IfcRelAggregates>
<IfcRelDefinesByProperties id="i2366">
	<GlobalId>3CQ2dBLlr0rwxF4txggh1g</GlobalId>
	<OwnerHistory>
		<IfcOwnerHistory xsi:nil="true" ref="i1944"/>
	</OwnerHistory>
	<RelatedObjects exp:cType="set">
		<IfcSpace xsi:nil="true" ref="i2103"/>
	</RelatedObjects>
	<RelatingPropertyDefinition>
		<IfcElementQuantity xsi:nil="true" ref="i2364"/>
	</RelatingPropertyDefinition>
</IfcRelDefinesByProperties>
<IfcElementQuantity id="i2364">
	<GlobalId>1hkBuNnSP3rQUYesUq5kwH</GlobalId>
	<OwnerHistory>
		<IfcOwnerHistory xsi:nil="true" ref="i1944"/>
	</OwnerHistory>
	<Name>BaseQuantities</Name>
	<Description></Description>
	<Quantities exp:cType="set">
		<IfcQuantityArea xsi:nil="true" ref="i2358"/>
		<IfcQuantityLength xsi:nil="true" ref="i2361"/>
		<IfcQuantityArea xsi:nil="true" ref="i2362"/>
		<IfcQuantityVolume xsi:nil="true" ref="i2363"/>
	</Quantities>
</IfcElementQuantity>
```

(b)

FIGURE 5.21 (Continued)

- *Pset_SpaceLightingRequirements*: Common property set for all types of spaces to capture the lighting requirements.
- *Pset_SpaceOccupancyRequirements*: Common property set for all types of spaces to capture the occupancy requirements.
- *Pset_SpaceThermalRequirements*: Common property set for all types of spaces to capture the thermal requirements.
- *Pset_SpaceThermalDesign*: Common property set for all types of spaces to capture building service design values.

```
<IfcSpace id="i2103">
      <GlobalId>0BTBFw6f90Nfh9rP1dlXri</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
      </OwnerHistory>
      <Name>A201</Name>
      <ObjectPlacement>
            <IfcLocalPlacement xsi:nil="true" ref="i2057"/>
      </ObjectPlacement>
      <Representation>
            <IfcProductDefinitionShape xsi:nil="true" ref="i2097"/>
      </Representation>
      <LongName>Hallway</LongName>
      <CompositionType>element</CompositionType>
      <PredefinedType>space</PredefinedType>
</IfcSpace>
<IfcRelDefinesByProperties id="i2355">
      <GlobalId>0lZ50ubpT0599sXUXSXdM1</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
      </OwnerHistory>
      <RelatedObjects exp:cType="set">
            <IfcSpace xsi:nil="true" ref="i2103"/>
      </RelatedObjects>
      <RelatingPropertyDefinition>
            <IfcPropertySet xsi:nil="true" ref="i2346"/>
      </RelatingPropertyDefinition>
</IfcRelDefinesByProperties>
<IfcQuantityArea id="i2358">
      <Name>NetFloorArea</Name>
      <Description>area measured in geometry</Description>
      <AreaValue>83.9585</AreaValue>
</IfcQuantityArea>
<IfcQuantityLength id="i2361">
      <Name>Height</Name>
      <Description>length measured in geometry</Description>
      <LengthValue>10.0</LengthValue>
      </IfcQuantityLength>
<IfcQuantityArea id="i2362">
      <Name>GrossFloorArea</Name>
      <Description>area measured in geometry</Description>
      <AreaValue>83.9585</AreaValue>
</IfcQuantityArea>
```

(c)

FIGURE 5.21 (Continued)

Figure 5.21 illustrates part of the ifcXML data file (IfcXMLFile1.xml) for the building being examined in this example. The part shown focuses on the IFC objects required to perform the code conformance checking for FBC 2014, Chapter 5. On the other hand, Figure 5.22 depicts part of the ifcXML data file (IfcXMLFile2.xml) for the building regulations rules of FBC 2014, Chapter 5.

The LINQ-to-ifcXML code in Figure 5.23 accesses the ifcXML file and reads the encoded provisions given by FBC 2014, Chapter 5, using the Microsoft C# language. The first section of the code evidently illustrates the power of LINQ to extract information from the ifcXML data of the building in effective and flexible expressions. The query searches the ifcXM data file of the

sample building (Figure 5.23) and reads the information about the total number of storeys and the total height (line 19–23 in Figure 5.23) using the proposed LINQ-to-ifcXML framework. From line 24 to line 46, the query searches for each storey and then determines the spaces within that storey. All the properties of a space within a storey are read and saved to the *SpaceProperties* object. This object has parameters that define areas, height, volume, and so on (line 5–8 in Figure 5.23). A list object is then created to save all the spaces within a storey. Then, this list object is utilized to determine the properties needed for

```
<IfcBuildingCode id="i100000">
      <GlobalId>1xS3BCk291UvhgP2a6eflM</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i1100"/>
      </OwnerHistory>
      <Year>2014</Year>
      <Version>2014</Version>
      <IfcCodeName id="i1102">
            <Name>2014</Name>
            <Description>2014</Description>
      </IfcCodeName>
   <IfcRelAggregates id="i28249">
      <GlobalId>15Z0v90RiHrPC200A6FoKP</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i2944"/>
      </OwnerHistory>
      <RelatingObject>
            <IfcRegAttributes xsi:nil="true" ref="i3048"/>
      </RelatingObject>
      <RelatedObjects exp:cType="set">
            <IfcSectionNumber xsi:nil="true" ref="i4103"/>
            <IfcSectionNumber xsi:nil="true" ref="i4512"/>
            <IfcSectionNumber xsi:nil="true" ref="i4777"/>
            <IfcSectionNumber xsi:nil="true" ref="i4182"/>
            <IfcSectionNumber xsi:nil="true" ref="i4531"/>
            <IfcSectionNumber xsi:nil="true" ref="i4868"/>
            <IfcSectionNumber xsi:nil="true" ref="i6117"/>
            <IfcSectionNumber xsi:nil="true" ref="i6520"/>
            <IfcSectionNumber xsi:nil="true" ref="i7854"/>
            <IfcSectionNumber xsi:nil="true" ref="i7088"/>
      </RelatedObjects>
    </IfcRelAggregates>
   <IfcSectionNumber id="i4103">
      <GlobalId>0BTBFw6f90Nfh9rP1dlXsj</GlobalId>
      <OwnerHistory>
            <IfcOwnerHistory xsi:nil="true" ref="i2944"/>
      </OwnerHistory>
      <Chapter>5</Chapter>
      <SectionTitle>General Building Height and Area Limitations</SectionTitle>
      <SectionNumber>503</SectionNumber>
      <ClassificationType>Content</ClassificationType>
      <Clause>Table 503: Allowable Building Height and Areas</Clause>
   </IfcSectionNumber>
   <IfcRegAttributes id="i3048">
      <IfcTypeOfConstruction id="i4001">
        <Name>TypeIA</Name>
        <HeightLimit>Unlimited</HeightLimit>
        <NumberOfStories>Unlimited</NumberOfStories>
        <AreaLimit>Unlimited</AreaLimit>
      </IfcTypeOfConstruction>
```

FIGURE 5.22 Part of the ifcXML data of FBC 2014, chapter 5.

```
   <IfcRegAttributes id="i3049">
      <IfcTypeOfConstruction id="i4002">
         <Name>TypeIB</Name>
         <IfcOccupancyAndUse id="i5001">
            <Description>A-1</Description>
            <HeightLimit>160.0</HeightLimit>
            <AreaLimit>Unlimited</AreaLimit>
            <NumberOfStories>5</NumberOfStories>
         </IfcOccupancyAndUse>
      <IfcOccupancyAndUse id="i5002">
            <Description>A-2</Description>
            <HeightLimit>160.0</HeightLimit>
            <AreaLimit>Unlimited</AreaLimit>
            <NumberOfStories>11</NumberOfStories>
         </IfcOccupancyAndUse>
      <IfcOccupancyAndUse id="i5003">
            <Description>A-3</Description>
            <HeightLimit>160.0</HeightLimit>
            <AreaLimit>Unlimited</AreaLimit>
            <NumberOfStories>11</NumberOfStories>
         </IfcOccupancyAndUse>
      <IfcOccupancyAndUse id="i5004">
            <Description>A-4</Description>
            <HeightLimit>160.0</HeightLimit>
            <AreaLimit>Unlimited</AreaLimit>
            <NumberOfStories>11</NumberOfStories>
         </IfcOccupancyAndUse>
      <IfcOccupancyAndUse id="i5005">
            <Description>A-5</Description>
            <HeightLimit>160.0</HeightLimit>
            <AreaLimit>Unlimited</AreaLimit>
            <NumberOfStories>Unlimited</NumberOfStories>
         </IfcOccupancyAndUse>
      <IfcOccupancyAndUse id="i5006">
            <Description>B</Description>
            <HeightLimit>160.0</HeightLimit>
            <AreaLimit>Unlimited</AreaLimit>
            <NumberOfStories>11</NumberOfStories>
         </IfcOccupancyAndUse>
      <IfcOccupancyAndUse id="i5007">
            <Description>E</Description>
            <HeightLimit>160.0</HeightLimit>
            <AreaLimit>Unlimited</AreaLimit>
            <NumberOfStories>5</NumberOfStories>
         </IfcOccupancyAndUse>
      </IfcTypeOfConstruction>
   </IfcRegAttributes>
</IfcBuildingCode>
```

FIGURE 5.22 (Continued)

code checking for each storey and save them to the *StoreyProperties* object (line 37–45 in Figure 5.23).

The second section of the code (line 60–131 in Figure 5.23) queries FBC 2014, Chapter 5, Section 503, for the desired height and areas limitations and then verifies them against the information extracted from the ifcXML data file (Figure 5.21) of the sample building used. The proposed framework on line 66–71 in Figure 5.23 implements LINQ to BIM via ifcXML to check the compliance of the gross area of each storey. Similarly, the compliance with building height is verified in line 72–75 and the number of storeys are checked in line 76–75 in Figure 5.23.

```
XElement CodeCheck = XElement.Load("C:\BIM\Books\IfcXMLFile1.xml");
int NoOfStoreies = 0;
decimal tHeight= 0.0; decimal totalgrossArea=0.0;
decimal totalGrossArea =0.0; decimal totalNetArea =0.0;
public struct SpaceProperties {
      public int StoreyID, SpaceID;
      public decimal  NetFloorArea, GrossFloorArea;
      public decimal  Height,Volume;
}
public struct StoreyProperties {
      public int StoreyID;
      public decimal  netFloorArea, grossFloorArea;
      public decimal  Height;
}
 SpaceProperties spaceProp;  StoreyProperties levelProp;
 var bsID = new List<int>();  var spaceID = new List<int>();
 var sProp = new List<SpaceProperties>();
 var storeyProp = new List< StoreyProperties>();
 foreach (XElement x in CodeCheck.Elements("IfcBuildingStorey")
 {
    int NoOfStoreies +=1;
    tHeight += (decimal) x.Element("Elevation");
    bsID.Add((int) x.Attribute("id"));
 }
 foreach (int j in bsID) {
   IEnumerable<XElement> ifcReAgg =
    from el in CodeCheck.Elements("IfcRelAggregates")
    where (int)el.Descendants("IfcBuildingStorey").Attribute("ref") == j
     select el;
   foreach (XElement e in ifcReAgg) {
         int sID = e.Descendants("IfcSpace").Attribute("ref")
         spaceID.Add(sID);
         int elQuatityID = (int) (from el in
         CodeCheck.Element("IfcRelDefinesByProperties")
     where (int)el.Descendants("IfcElementQuantity").Attribute("ref")
              == sID
     select el);
     SpaceQuantities (elQuantityID; grArea, netArea);
     SpaceProp.StoreyID= j;  SpaceProp.SpaceID=sID;
     SpaceProp.NetFloorArea=netArea;  SpaceProp.GrossFloorArea=grArea;
     sProp.Add(spaceProp);
     totalGrossArea += grArea; totalNetArea += netArea;
   }
   levelProp.StoreyID = j; levelProp.grossFloorArea =
   levelProp.netFloorArea= totalNetArea;
```

FIGURE 5.23 LINQ-to-ifcXML code for automated building regulations checking.

This example illustrates the potential application of the proposed framework to automate an unlimited range of building rules and regulations, including unlimited nested conditions and the branching of alternative contexts within a specified building code and standard.

LIST OF ABBREVIATIONS

AEC: architecture, engineering, and construction. This term is used in an inclusive way and includes facility management, operator, and owner, in addition to related public government.

```
45.    storeyProp.Add(levelProp);
45.    totalGrossArea =0.0; totalNetArea = 0.0;
46.  }
47.  string ConstType =  from btype in CodeCheck.Elements("IfcSite")
48.    select (string) btype.Element("ConstructionType");
49.  string ConstGroup =  from btype in CodeCheck.Elements("IfcSite")
50.    select (string) btype.Element("Group");
46. static void SpaceQuantities(int y, ref decimal A1, ref decimal A2)
47. {
48.    from el in CodeCheck.Elements("IfcElementQuantity")
49.    where (int)el.Attribute("id") == y
50.    select el;
51.    Switch  el.Elements("Name"){
52.          Case "NetFloorArea":
53.            A1 = (decimal) el.Element("AreaValue");
54.            Break;
55.          Case "GrossFloorArea":
56.            A2 = (decimal) el.Element("AreaValue");
57.            Break;
58.    }
59. }
60.  string VerifigyArea, VerifyHeight, VerifyNoOfStories;
60.  XElement bCode = XElement.Load("C:\BIM\Books\IfcXMLbCode.xml");
61.  from node in bCode.Elements("IfcSectionNumber")
62.  where (string)node.Attribute("SectionNumber") == "503"
63.  select node;
64.  String sectionName = (string) node.Element("SectionTitle")
65.  String Clause = (string) node.Element("Clause")
66.  foreach (var story in storeyProp) {
67.    int id = (int) storey.StoreyID;
68.    decimial gArea = (decimal) story.StoreyID;
69.    decimial nArea = (decimal) stoey.StoreyID;
70.    if CheckArea (id, gA, ConstType, OccuType) {
71.         VerifyArea= "PASS";
72.    }
73.    Else {
74.      VerifyArea= "FAIL";
75.    }
71.  }
72.   if CheckHeight(tHight, ConstType, OccuType);
73.         VerifyHeight = "PASS";
74.   Else
75.         VerifyHeight = "FAIL";
```

FIGURE 5.23 (Continued)

BIM: In this book, the term is used to denote *building information model*, *modeling*, or *workflow*.
IDM: Information Delivery Manual
IFC: Industry Foundation Classes
LIM: life-cycle information model
NLP: natural language processing
OWL: Web Ontology Language
RDF: Resource Description Framework
XML: Extensible Markup Language

```
76.  if CheckNoOfStories( NoOfStoreies, ConstType, OccuType)
77.          VerifyNoOfStories = "PASS";
78.  Else
79.          VerifyNoOfStories = "FAIL";
80.  public bool CheckArea ( int j, decimal A1, string, coType, string
81.                ocType)
82.  {
83.     from el in bCode.Elements("IfcRegAttributes")
84.     where (string)el. Descendant s("Name") == coType  &&
85.           (string)el. Descendants ("Description") == ocType
86.     select el;
87.     If (string) el.Descendants("AreaLimit") == "Unlimited " {
88.          return true;
89.     }
90.     else if (decimal) el.Descendants("AreaLimit ") >= A1 {
91.          return true;
92.    }
93.    else {
94.        return false;
95.    }
96.  }
97.    public bool CheckHeight(decimal h1, string, coType, string
98.                ocType)
99.  {
100.    from el in bCode.Elements("IfcRegAttributes")
101.     where (string)el. Descendants("Name") == coType &&
102.          (string)el.Descendants(" Description") == ocType
103.     select el;
104.    If (string) el.Descendants("HeightLimit") == "Unlimited" {
105.         return true;
106.     }
107.    else if (decimal) el.Descendants("HeightLimit") >= h1 {
108.         return true;
109.    }
110.    else {
112.       return false;
113.    }
114.  }
```

```
115.   public bool CheckNoOfStories(int n1, string, coType, string
116.                ocType)
117.  {
118.      from el in bCode.Elements("IfcRegAttributes")
119.       where (string)el. Descendants("Name") == coType  &&
120.           (string)el. Descendants("Description") == ocType
121.      select el;
122.     If (string) el.Descendants("NumberOfStories") == "Unlimited" {
123.        return true;
124.      }
125.     else if (int) el.Descendants("NumberOfStories") >= n1 {
126.        return true;
127.     }
128.     else {
129.        return false;
130.     }
131.  }
```

FIGURE 5.23 (Continued)

REFERENCES

ASCE7-10 (2010). Minimum Design Loads for Buildings and Other Structures. American Society of Civil Engineers, ISBN (print): 978078441291.

buildingSMART International. (2014). Model Support Group of buildingSMART International. http://www.buildingsmart-tech.org/about-us/msg. (accessed 10/29/2016).

FBC, (2014). Florida Building Code 5th Edition. International Code Council (ICC).

Liebich, T. and Wise, M. (2012). Simple ifcXML. http://iug.buildingsmart.org/resources/iug-meetings-2012-oslo/process-room-workshop-20-march-2012/Simple%20ifcXML%20presentation%20v2.0.pdf (accessed 5/19/2017).

Microsoft (2017a). Process XML Data Using LINQ to XML. https://docs.microsoft.com/en-us/dotnet/standard/data/xml/process-xml-data-using-linq-to-xml (accessed 2/17/2017).

Microsoft (2017b). NET Language-Integrated Query for XML Data. https://msdn.microsoft.com/en-us/library/bb308960.aspx?f=255&MSPPError=-2147217396 (accessed 3/14/2017).

Nawari, N.O. (2012a). BIM standard in off-site construction. *Journal of Architectural Engineering*, 18(2), 1–8, June 1, 2012. ASCE, ISSN 1076-0431/2012/2-107.

Nawari, N.O. (2012b). The challenge of computerizing building codes in a BIM environment. *Proceedings of the International Conference on Computing in Civil Engineering*, ASCE, Clearwater Beach, FL, June 17–20, 2012.

Nawari, N.O. (2012c). Automated code checking. Journal of Architectural Engineering, 18(4), 315–323, December 1, 2012. ASCE, ISSN 1076-0431/2012/4-315e323.

Nawari, N.O. (2012d). Automated code checking in BIM environment. *Proceedings of the 14th International Conference on Computing in Civil and Building Engineering*, Moscow, Russia, 27–29 June, 2012.

Nawari, N.O. (2014). BIM standard: Tensile structures data modeling. *Proceedings of the 15th International Conference on Computing in Civil and Building Engineering*, Orlando, FL, June 22–25, 2014.

Nawari, N.O. and Alsaffar, A. (2015). Understanding computable building codes. *Journal of Civil Engineering and Architecture*, 3(6), 163–172.

Index