Automation in Construction 82 (2017) 43–58

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Computer representation of building codes for automated compliance checking

Sibel Macit İlal[a,*], H. Murat Günaydın[b]

[a] İzmir Democracy University, Faculty of Architecture, Department of Architecture, İzmir, Turkey
[b] İstanbul Technical University, Faculty of Architecture, Department of Architecture, İstanbul, Turkey

ARTICLE INFO

Keywords:
Building code representation
Automated compliance checking
Semantic modeling
BIM
IFC

ABSTRACT

Development of automated building code compliance checking systems requires appropriate representations for building codes. Building codes are complex documents written in natural languages, and the development of computable representations is challenging. This paper presents a new model and an accompanying modeling methodology for the representation of building codes that may be utilized in the development of future automated compliance checking systems.

The new model combines the semantic modeling approach of the SMARTcodes project with the theoretical foundations established by Nyman and Fenves, namely the four level representation. This hybrid model organizes the representation in four levels and allows for separate modeling of domain concepts, individual rule statements, relationships between rules, and the overall organization of the building code.

The model is evaluated with a case study. The İzmir Municipality Housing and Zoning Code is chosen as it is representative of complex building codes that are in effect throughout Turkey. The formalizable rules in the section of Izmir code that apply to all types of buildings, are represented in computer implementable format based on the new model. The research presented in this paper shows that decomposing a building code into four levels and modeling rules based on the semantic-oriented paradigm is an effective modeling strategy for representing building codes in a computable form.

1. Introduction

In the Architecture, Engineering, and Construction (AEC) industry, building projects must be checked against numerous building codes for compliance. They are allowed to be executed only when compliance with all applicable rules of the building code have been guaranteed. Compliance checking is a major task for both architects and building certifiers often involving ambiguities and inconsistencies in assessment, leading to delays in the overall construction process [1,2]. Failure to correctly assess projects for compliance can also have negative effects on building performance and allow errors that are expensive to correct. Today, although every building project is modeled in a digital environment, compliance checking is a manual process increasing the delays as well as the risk for errors in evaluation [3].

Automated compliance checking has long been an area of research that aims to provide computational support for accurate compliance checking of building projects against applicable building codes in a time and cost effective way. Research into developing automated compliance checking systems has focused mostly on three areas: Representation of building codes in computational format [4–6] definition of building model views [7,8], and compliance checking algorithms and reporting [1,9,10].

Automated compliance checking systems are expected to retrieve a set of building codes from related authorities and conduct compliance checking on submitted building projects. Compliance checking systems primarily require appropriate computer-based models of both building codes and building designs. Advances in BIM tools have finally established a standardized representation for building designs, even if it is currently deemed unsatisfactory. However, a standard representation for building codes is still not available.

The goal of the research presented in this paper is to develop a new computer representation for building codes that can be used in the development of future automated compliance checking systems. The aim is to develop a formal model that supports the creation of digital representations of build codes by following a corresponding methodology. To achieve this aim, the research has been conducted in the following stages: 1) Exploring and evaluating previous modeling approaches; 2) Analyzing building codes to understand the various types of information contained in them and identifying the components of rule statements as well as the organization of the documents; 3)

* Corresponding author.
E-mail addresses: sibelmacit@gmail.com, sibel.macit@idu.edu.tr (S. Macit İlal), gunaydinh@itu.edu.tr (H.M. Günaydın).

http://dx.doi.org/10.1016/j.autcon.2017.06.018
Received 4 May 2016; Received in revised form 9 April 2017; Accepted 11 June 2017
Available online 29 June 2017

Developing a formal representation model for building codes; 4) Defining a building code modeling methodology for utilizing the representation model to build digital versions of existing building codes; 5) Modeling of an actual building code as a case study to demonstrate the feasibility, benefits, and limitations of the representation model; 6) Implementing a prototype system to demonstrate an application and to test the validity of the new model.

This research focuses on the issue of building code representations and its scope is limited to it. Other issues impeding the development of automated compliance checking systems exist, such as building model extensions for code checking, and efficient querying and checking methods but they are beyond the scope of this work. Additionally, the scope is also limited to representing rules that are formalizable, i.e. can be evaluated computationally. Semi-formalizable rules that contain ambiguous statements or based on relative judgment (e.g. enough, adequate, etc.) as well as non-formalizable rules that require qualitative evaluations (e.g. aesthetic judgment) are not considered.

As a conclusion, this research proposes a formal model for building code representation based on the analysis of building codes and theories established in literature. The theory embedded in this proposed model is evaluated through the development of an actual building code and a prototype implementation.

2. Related work

There has been an extensive amount of research conducted internationally over the last four decades in the area of representing building codes in a computable format for automated compliance checking. The introduction of decision tables by Fenves [11] is the initial effort on building code representation. In this effort, building code provisions are represented in a precise and unambiguous decision table form. A decision table is a concise tabular representation of the conditions applicable in a given situation and of the appropriate actions to be taken as a result of the values of the conditions [12]. A follow-up project by the same group of researchers investigated the restructuring of the AISC (American Institute of Steel Construction) Specification [13]. In this project, the content of the building code is modeled in four levels. This abstract model of the logical structure of building codes is used as a modeling methodology in a software system called SASE (Standard Analysis, Synthesis and Expression), to represent individual provisions, relationships among provisions, and the organization of the building code [14]. This system aimed to provide tools for creating decision tables and for structuring building code representations.

Several researchers [15–18] have proposed methods based on a rule-based modeling approach for representing building codes as rules/clauses in processing systems. In these models, the clauses of the building code are represented as a set of rules in the form of IF [condition] THEN [action] statements instead of decision tables. Later works focused on structuring building codes in a predicate logic structure [19,20]. A commercial expert system was developed in Australia by the Commonwealth Scientific and Industrial Research Organization (CSIRO) called BCAider [21]. This system was available between 1991 and 2005.

Many other models for representing building codes in a computable format have been proposed. Garrett and Hakim [22] developed an object-oriented model of building codes, which allows organizing a building code around building objects pertinent to the building code. Waard [23] offered another object-oriented approach to building code processing. In this study, an object model for residential buildings and another object model for building codes were developed, and the two models were linked for compliance checking. Yabuki and Law [24] combined first order predicate logic and object-oriented modeling approaches to represent and process building codes. Kiliccote and Garrett [25] developed a context-oriented model for representing building codes. This model uses the object-oriented modeling approach and organizes building code around “contexts” which are a collection of subclasses used to define conditional parts of the provisions for which they are applicable. Given a predicate logic structure, Kerrigan and Law [26] developed the REGNET application to determine the applicability of various codes under given building conditions, based on a question-and-answer user interface.

Previous building code representation research efforts mainly focused on the hard-coding approach. The main disadvantage to this approach is that it requires a high-level of expertise in computer programming to define, write and maintain building codes. To overcome the deficiencies of hard-coded representation approaches, recently, attention has been directed towards the study of semantic modeling approach, which is a relatively new method for knowledge representation. The SMARTcodes project [27] is a semantic approach which proposes to mark-up building codes in such a way that rules are dynamically generated in a computable format. The project provides a protocol and a software program (SMARTcodes Builder) for creating smart versions (tagged representations) of actual building code texts that reflects building codes with schema and tags used for automated compliance checking applications [28].

Recently, the application of an ontology-based approach has been investigated as a possible computable framework for building code representation. Yurchyshyna et al. [29] developed a formal ontology-based approach for the formalization and semantic organization of building codes. Ontology-based building code representation using semantic web technologies has also been explored by researchers [30]. Dimyadi et al. [31] developed a regulatory knowledge model and a higher level query language for designers' access to this model. Beach et al. [32] developed a rule-based semantic methodology for the specification of a regulatory compliance checking system. There have been also some projects using semantic modeling approach and the application of industry specific taxonomies and ontologies in combination with Artificial Intelligence (AI) and Natural Language Processing (NLP) techniques to allow systems to interpret building code by automated or semi-automated data extraction [33–35].

Exploration of building code compliance checking systems for building models began following the development of the Industry Foundation Classes (IFC) in the 1990s. The Singapore Construction and Real Estate NETwork (CORENET) project is the earliest production of building code compliance checking effort initiated in 1995 [36]. Initial work was based on electronic 2D drawings, but later on IFC [37] was used. The CORENET project developed the FORNAX platform to capture needed building code information. Another effort is the DesignCheck system from Australia, initiated in 2006 [38]. In this effort, the EXPRESS Data Manager (EDM) [39] platform was used for encoding barrier-free accessibility rules. A more recent effort, led by USA International Code Council (ICC), developed SMARTcodes Model Checking System [40,41]. It is a platform providing methods of translation from written, natural language rules to computer code. The platform targets energy conservation rules. USA General Services Administration (GSA) have supported development of a rule checking system for circulation and security validation of U.S. Court houses [42].

Although several researches have already proposed various building code models, and software environments to operate on these models and check building projects for compliance with building codes, for a variety of reasons, employing these environments in AEC industry practice has been limited. Literature review reveals that the reasons for this failure are related to the building code models used in these environments [43,44]. Previous building code models do have several limitations. One limitation is not being comprehensive enough compared to the complex nature of building codes and thus lacking the capability to represent all of the various types of information in building codes. A second limitation is that some building codes are hard-coded into the systems, thus lacking flexibility, maintainability, and user control (i.e. non-programmer users cannot add/modify the rules embedded in the system). Furthermore, any change in the building code necessitates changes in all such systems. A third

limitation is that there is no direct mapping between the building code documents and building code models, making consistency checking between actual building code and code model difficult. A fourth limitation is focusing only on individual rule representation ignoring the overall building code and thus lacking capability to prevent contradictions among rules.

The motivation for the research presented in this paper stems from these limitations of previous building code models. A new model for representing building codes is needed to improve on existing models. After investigating previous models and identifying their limitations, the following requirements were established for the new representation model: *Independence* - keeping the representation of building codes independent of the compliance checking system and the design system; *Conciseness* - avoiding redundancy in the representation of building codes; *Consistency* - preventing ambiguities as well as contradictions among rules; *Comprehensiveness* - representing all of the various types of information in building codes (i.e. concepts, requirements, applicability conditions, etc.); *Maintainability* - allowing creation and addition of new rules and modification of existing ones. The proposed building code representation model aims to meet these requirements.

3. The new building code representation

The new representation model adopts the four-level representation paradigm [45] as a theoretical base and uses the semantic modeling approach for developing the building code representation.

3.1. Four-level representation

The four-level paradigm is derived on the basis of an abstract model of the logical structure of building codes identified by Nyman and Fenves [46] who investigated alternatives for restructuring building codes. According to this paradigm, the content of the building code is examined in the following four levels:

1. The top level (organizational network) that represents outlines and overall organization of the building code.
2. The intermediate level (information network) that represents the dependency relationships among provisions of the building code.
3. The detailed level that represents the individual provisions in the form of decision tables.
4. The lowest level that consists of the basic *data items* referred to in the provisions.

Identifying the nature of building codes and the hierarchy of information in them is important for the development of successful models for building codes. Nyman's research provides a solid foundation for modeling building codes and outlines the process of modeling building codes. This general organizational structure was initially used for the SASE Model as a representation framework by Fenves et al. [14]. Subsequent studies on building code modeling are also based on this approach [19,20]. Although this approach was based on a solid theoretical foundation, its application in the field was not practical and models based on this theory were not widely adopted in AEC industry. Literature review reveals two important reasons for this failure. First reason is the lack of domain models that identify domain specific objects with their attributes and relationships. The lack of an industry standard in building modeling led to a high number of idiosyncratic data item definitions and increased the complexity of the building code models [47]. Second reason is related to the representation methods used in the modeling of building code information. Decision tables and programming languages used for representation quickly became hard to maintain and build. The complexities involved in actual building codes proved too difficult for these methods of representation [43]. In summary, the four-level representation paradigm was not adopted in practice, mainly due to the fact that information technologies for knowledge representation were not mature enough at the time and since the late 90s have not received any attention.

The new representation model developed in this research adopts the four-level paradigm as a theoretical base for representing building codes but addresses the above issues on knowledge representation methods by utilizing the relatively recent semantic modeling method.

3.2. Semantic representation – RASE model

Representation methods in the modeling of building code information were inefficient and not easily understandable by non-programmer users due to the fact that most of them were based on hard-coding approaches. Ideally, the representation should be independent of compliance checking systems. It should also be adaptable to continuing building code amendments. The key is to make it possible for domain experts, who generally do not have any programming knowledge to manage the representations themselves. The semantic modeling approach, which is a relatively new method for knowledge representation, aims to meet the above requirements. SMARTcodes, can be referenced as a good example of the use of this approach [28]. SMARTCodes project ended in 2010 but the mark-up concept and method developed for this project is currently maintained by AEC3 (UK) Ltd. [27]. The SMARTcodes project aims to define computable rules using simple tools that enable non-programmers to create representations by tagging actual building code texts. Rules are modeled based a new methodology called RASE (Requirement, Applicability, Selection, Exception). RASE defines four common constructs that make up a rule. It states that building code rules can be broken down into four constructs: **R**equirement – defining the condition that must be satisfied by one or more aspects of a building; **A**pplicability – defining which aspect of the building the requirements apply to; **S**election – indicating conditions if the rule is for specified cases among applicable elements; **E**xception – identifying the conditions under which the check is not applicable to the building elements.

The RASE model utilizes these four constructs to identify the building code essence from the actual text of the code. Each of these four constructs has attributes such as a property, a comparator and a target value with a unit. Building code authors are able to markup these indicators that appear in the actual text of the code using SMARTcodes Builder software which creates an XML formatted version of the code [41]..

The RASE model provides an easy to understand, simple method for deconstructing rule sentences. It also accommodates a scheme where code authors are able to build and maintain building code representations. In order to evaluate the capability of the RASE to model real building codes, a pilot study was carried out in the early phase of the research presented in this paper [48]. Izmir Municipality Housing and Zoning Code (IMHZCode) rules have been modeled as requirement, applicability, selection, and exception objects. Some examples of how rules are modeled are shown in Table 1.

The experiences gained in the pilot study showed that the RASE methodology offers an ease of use for non-programmers and could be adapted to represent IMHZCode, however, serious modifications would be required to overcome three shortcomings that were identified. The first shortcoming is the unnecessary repetitions that occur due to the independent modeling of individual rule statements. Representations of the same concepts, referenced by multiple rules, are repeated many times for each *applicability* or *selection* construct they are a part of. The RASE model integrates representation of the code requirements and the domain specific concepts and entities in a single rule representation. While this simplifies conversion of building code texts into a computable building code model, it requires defining the same concepts and entities multiple times for every rule where they are referenced. This creates redundancies and may lead to inconsistencies especially when the concept or entity definitions require updates. In the SMARTcodes project, an external dictionary is used to prevent inconsistencies.

Table 1
Examples of rules from IMHZcode modeled according to RASE.

Id	Rule Text	Requirement					Applicability				
		topic	property	comparator	value	unit	topic	property	comparator	value	unit
1	Clear height of doors shall be at least 2.10 m.	door	Height	≥	2.10	m	Building element	subType	=	door	–
2	Clear width of entrance doors of independent unit shall be at least 1.00 m.	door	Width	≥	1.00	m	Building element	subType	=	door	–
3	Clear width of rooms, kitchen, and bathroom doors shall be at least 0.90 m.	door	Width	≥	0.90	m	Building element	subType	=	door	–
4	Buildings shall have at least one non-wood staircase.	building stair	#stair material	≥ !=	1 wood	– –	Building	–	–	–	–
5	The minimum width" of a flight and a landing shall be 1.20 m.	stair stair	flight Width landingWidth	≥ ≥	1.20 1.20	m m	Building Element	subType	=	stair	–
6	Roofs in general must remain within 33% sloping height, except duplex houses.	roof	pitch	≤	33	%	building Element	subType	=	roof	–

Id	Rule Text	Selection					Exception				
		topic	property	comparator	value	unit	topic	property	comparator	value	unit
1	Clear height of doors shall be at least 2.10 m.						–	–	–	–	–
2	Clear width of entrance doors of independent unit shall be at least 1.00 m.	door door	type contained	= =	entrance iUnit	– –	–	–	–	–	–
3	Clear width of rooms, kitchen, and bathroom doors shall be at least 0.90 m.	door door door door	type type type type	= = = =	room kitchen bathrom wc	– – – –	–	–	–	–	–
4	Buildings shall have at least one non-wood staircase.	–	–	–	–	–	–	–	–	–	–
5	The minimum width" of a flight and a landing shall be 1.20 m.										
6	Roofs in general must remain within 33% sloping height, except duplex houses.	–	–	–	–	–	Building	Type	=	Duplex house	–

Building codes, in general, have a number of rule statements that indicate different *requirements* about the same concept. This is true especially for IMHZCode. Hence, the results of the pilot study unveiled a high number of redundant definitions especially for applicability constructs. The Domain Level of the new representation model that will be discussed in the next section is introduced to address this issue by creating a lower level library that can be used to define these repeating concepts once.

The second shortcoming is the lack of explicit relationships between individual rule statements. The RASE methodology delegates that responsibility to processing at a higher level within automated compliance checking systems. In the SMARTcodes project, first the original code text is marked-up, then this marked-up text is structured into an XML representation, and in the last stage the XML file is used to create computable rules for the automated checking system. It is in this final stage that the relationships are represented in the form of an IFC constraint hierarchy. This totally independent handling of rule statements simplifies deconstructing rules for especially non-programmer code authors. However, with the relationship representation taking place separately in the automated checking system, it becomes impossible to ensure correctness and consistency for the overall code representation independent of automated checking systems. Moreover, this split organization has a negative impact on the maintainability of the representation as well. It is necessary to represent the relationships independent of automated checking systems. The *Management Level* and the *Rule-set Level* of the new representation model are introduced to address this shortcoming and they will be discussed in the following section.

The third shortcoming is about the Exception construct of RASE. The pilot study revealed that it is unnecessary to represent separate selection and exception information for individual rule statements. The simple exceptions in identifying which objects a rule applies to can be integrated with the selection criteria. On the other hand, when a rule itself is an exception to another rule it can be handled by creating multiple rules that are connected with an OR relationship. Therefore, in the new hybrid model that allows representing such relationships at the *Rule-set Level*, the exception constructs are eliminated.

In summary, the semantic modeling approach of the SMARTcodes project was adopted for representing building code rule statements, but it was modified into a four-level representation. The modifications improve it by eliminating redundancies and adding logical relationships. The new representation model is completely independent of actual checking systems and thus should be easier to maintain. The new representation model is discussed in detail in the next section.

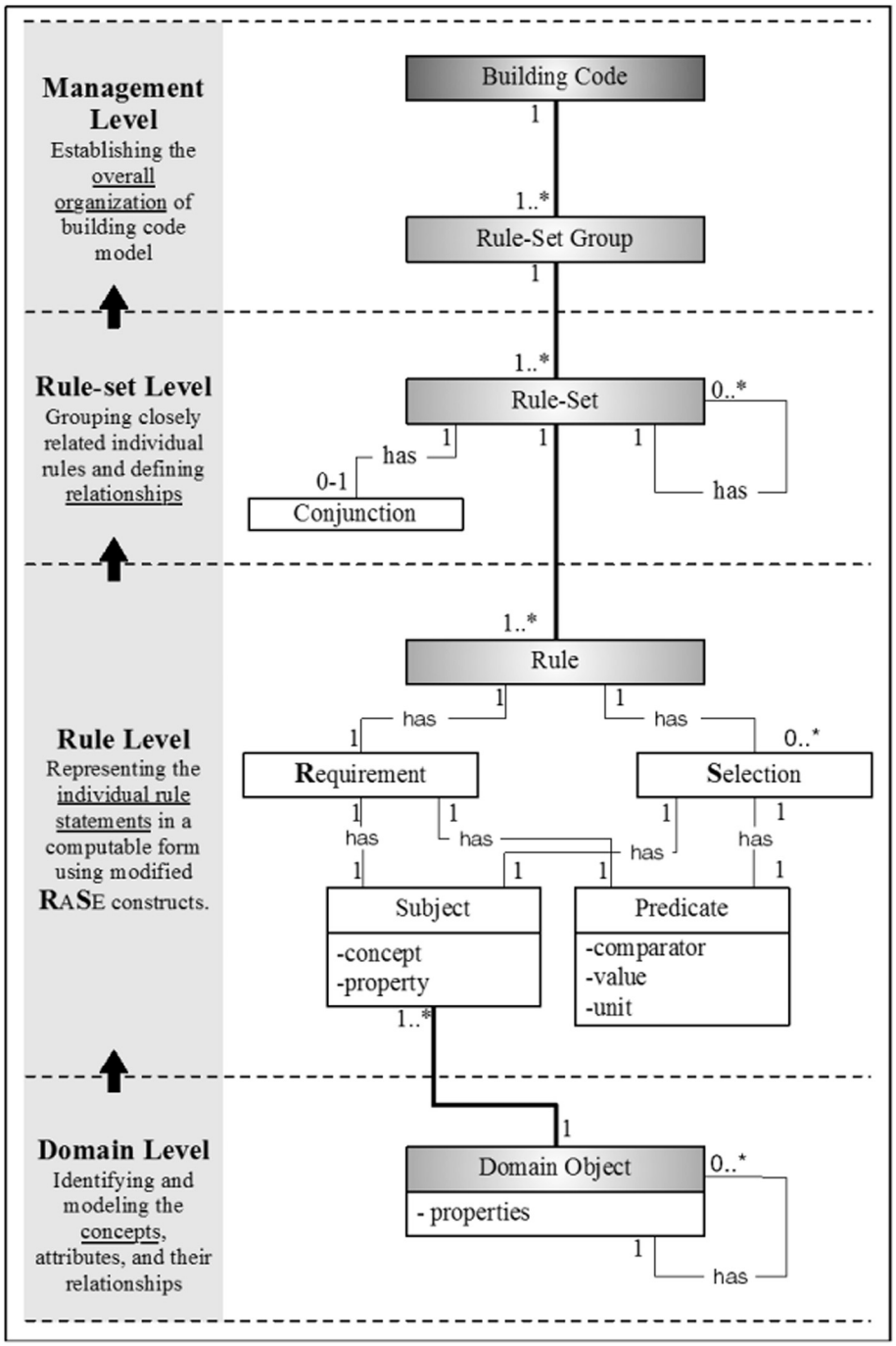

Fig. 1. Overall structure of the new hybrid building code representation.

3.3. The new representation

The hybrid model organizes the representation in four levels to provide a systematic structure for building codes in computational form. These four levels are (Fig. 1):

1. The *domain level* which models the concepts that are mentioned in the original building code text with their attributes and relationships.
2. The *rule level* where individual rule statements of the building code are represented in a structured format, utilizing the concepts modeled at the domain level. The rules are modeled based on modified RASE constructs.
3. The *rule-set level* where relationships between rule-objects are defined forming the rule-sets.
4. The *management level* which reflects the overall organization of the building code model by connecting and categorizing the rule-sets.

The new hybrid approach that modifies the RASE methodology following the four-level paradigm aims to:

- establish a building code representation *independent* of checking systems,
- preserve the high level of *maintainability* in the RASE model,
- minimize redundancies by introducing a hierarchical structure across four levels and improve on *conciseness*,
- offer a level in which rule relationships are modeled and monitored so that *consistency* of a building code model can be ensured.

3.3.1. Domain level

The lowest level of the hybrid model is the *domain level* for representing concepts and entities that appear in building code documents as object classes. Building codes refer to concepts specific to the domain which the codes are meant for (e.g. fire safety, accessibility) as well as entities that correspond to various aspects of the building project such as physical building components, spaces and relations (e.g. building, independent unit, storey, etc.). Automated compliance checking systems assess building projects after mapping these concepts and entities in the build code representation to objects that constitute a building project. The mapping process will benefit from modeling of these domain specific concepts and entities independent of the rules with hierarchical relationships that are similar to the ones in the building information model.

By first defining the domain specific concepts and entities with their attributes and relationships, independent of the rules, a domain model is created. The domain model acts as a library of objects that are utilized during the modeling of individual rule statements. The library objects can be used as building blocks during modeling and maintenance of the rules by code authors with no programming background. For example, the applicability constructs in the RASE methodology are filled by selecting from this library of domain objects. The domain level eliminates redundancies which occur when modeling multiple rules that refer to the same concept many times. Moreover, this domain model also helps expose the domain specific information and how it can map to building models in external systems. It is aimed at facilitating communication and interoperability among building information models, building code models, and automated compliance checking systems. For example, once the domain model of any code document is established, domain concepts can be mapped to IFC (preferably utilizing MVDs) allowing systems utilizing the new code representation to access and process IFC based project data.

3.3.2. Rule level

The second level of the hybrid model is the *rule level* for representing the individual rule statements in computable format. Building codes include a set of rule statements that a building project must satisfy. Building projects are checked against the requirements and/or conditions, indicated by these rule statements.

In this *rule level*, individual rule statements are represented as rule objects in the form of structured data based on the modified RASE constructs. Rule statement semantics are captured in these rule objects. Each rule object determines a single requirement for a specific attribute of a set of domain model objects that meet specific criteria. Every requirement is specified by a specific value and a method for comparison. Structure of the rule object is shown in Fig. 1.

In general, rule statements only have requirement information that indicates a quality requirement that must be satisfied by a domain concept. In some cases, rule statements also have selection information, if the requirement is for specified cases among applicable objects. Separate requirement and selection objects are modeled to capture this. Every rule object must have a single requirement object and may have zero or more selection objects.

Both requirement and selection objects have two constructs, a “subject” and a “predicate”. The subject has a simple structure consisting of two basic elements: a concept, and a property (e.g., *door - height*). Concept comes from the *domain level* and may be a physical building component such as wall, door, slab, or an abstract concept such as space (living room) or zone (independent unit). Properties are attributes of interest belonging to the concept. The predicate consists of a comparator, a value, and a unit. The comparator is one of the relational operators (e.g. greater than, less than, equal to). The value is the specific value that is found in the code, whether numeric, descriptive, or Boolean. The unit simply specifies the unit of measure for the value.

In the modified RASE, rules appear to be made up of only requirement and selection constructs, without the applicability and exception constructs. However, the applicability and exception information also exist. Applicability information is modeled as the Subject object under the Requirement object as shown in Fig. 1. Exceptions are handled in either of two ways depending on how the rules are stated. If the exception is about which set of objects the rule is applicable, the exception is simply included in the selection object modeled by adding a “!=” or “excludes” comparator. If the exception is about which rule applies to a set of objects (exception states a modification of the rule), a separate rule object is created and the exception of applicability is handled in the rule-set level by joining the two rules with an OR conjunction.

Every rule object indicates only a single requirement. However, building codes contain many complex rules which indicate multiple requirements of a subject or a requirement related to multiple concepts. These rules need to be broken down into multiple rules each indicating a single requirement and then connected at the rules-set level depending on the logic embedded in the requirement of the original complex rule statement. It can be stated that “selection logic” is represented by the rule objects and “requirement logic” is embedded in the rule-set objects.

3.3.3. Rule-set level

The third level of the hybrid model is the *rule-set level* for defining the relationships between individual rule objects. In the lower *rule level* each rule statement gets modeled with only one requirement for a specific property of a concept or entity and with selection information that clarifies the conditions under which the requirement applies to the concept or entity. However, in most cases an entity is subject to multiple requirements that vary according to the conditions. Multiple rule statements are used in order to specify and clarify conditions and requirements for a property of a concept or entity. Rules need to be connected representing the logical relationships that exist implicitly or explicitly within the semantics of a clause. Rules may be stand-alone, stating a requirement that is unrelated to other rules. However, for the most part, rules depend on each other. They either modify requirements or introduce additional requirements depending on the conditions. Rules can be joined with an OR conjunction when modifying the requirements and an AND conjunction when adding new requirements.

In the new model related rule objects are collected together into computable rule-sets by using logical conjunctions. AND conjunction is used for combining rule objects that indicate different values to be satisfied by a particular property of a concept simultaneously. OR conjunction is used for a relation between rules that indicate alternative values to be satisfied by a particular property of a concept depending on specified conditions. While all rule objects that are combined with an AND conjunction must be satisfied by the related concept, only one of the rule objects that are combined with an OR conjunction should be satisfied.

The rule-set level is a collection of rule-set objects. This rule-set model is the logical combination of distinct rule objects. Rule objects are grouped into rule-sets, when they are all addressing the same subject (a property of a concept) that is being constrained. Rules are connected by logical conjunctions and form a tree where the root is the rule-set. The leaf nodes are the rules that have been modeled at the lower level. The structure of the rule-set object is shown in Fig. 1.

3.3.4. Management level

Building codes are useful only if users (checkers or designers) can determine which portions of the building code pertain to their problem. To facilitate this, building codes are organized into chapters, sections, and paragraphs, with corresponding tables of contents and indexes. The user of a building code model should also be able to identify which rules of the building code apply for a given design situation. The building code model, therefore, needs to be organized in a systematic manner such that individual rules can be accessed easily. An organizational system can also be used to develop an outline to arrange the rules and

to define the scope for the building code. In the new model, the fourth level addresses issues related to the organization of the rule-set objects modeled in the *rule-set level*.

One natural method of organization is the one that reflects the original building code text. Rule-sets can be grouped to reflect the clauses and clauses can be ordered under sections following the order in the actual code document. However, it is beneficial to allow for alternative organization schemes to exist simultaneously. One such alternative organization is to group rule-sets according to the concepts they impose requirements for. In the actual code document requirements on a single concept can span across multiple clauses making it difficult to recognize inconsistencies. To overcome this shortcoming a concept-based organization is preferable. The concept-based organization is also helpful for automated compliance checking algorithms in identifying all rules that need to be processed for a given concept. The *management level* is included to allow alternative networks of linked rule-sets to co-exist. Moreover, this top level of the new model allows various building codes (such as Fire Safety Code, High-rise code) to be aggregated exposing possible conflicting provisions on concepts.

These organizing networks are modeled using rule-set group objects that form a tree. The root of the tree represents the overall code while the leaves are the rule-sets that are defined at the third level. Any number of intermediary nodes can be defined and they represent headings and sub-headings (sections and clauses in the actual document organization). The overall structure of the new building code representation is shown in Fig. 1.

4. Building code modeling methodology

The modeling process with which the new hybrid representation will be utilized is crucial for successful implementation. A clear, transparent, and well defined process is required. To this end, an accompanying methodology is proposed which comprises three process stages. Below are the recommended process stages to develop building code representations based on the proposed model.

- Stage 1: Analysis of the building code to define what should be represented explicitly for the purposes of automated compliance checking and to document how much of the building code can be modeled reliably.
- Stage 2: Representation of the building code by utilizing the developed representation model.
- Stage 3: Implementation of the building code model within a compliance checking application.

In the next section these stages are explained in detail. Fig. 2 illustrates the stages of the building code representation methodology.

4.1. Analysis stage

It is essential to document the various types of information contained in building codes as well as the organization of the codes in order to develop a building code representation. Thus, analysis of the building code is the first stage in the proposed building code modeling methodology. This stage covers the following steps; (1) determination of scope, (2) decomposition of the building code, (3) classification of the rule statements.

4.1.1. Determination of the scope

The building codes, chapters, and clauses that will be included in the representation should be clearly specified and documented. The output from this step will be a simple text document listing the parts of the code to be modeled, defining the scope of the representation. The actors of this step are the building code domain experts appointed by the authority.

4.1.2. Decomposition of the building code

In the second step, all clauses are decomposed into a list of statements and all statement types that exist are determined. Clauses are composed of different types of statements. While some of these statements are informative such as clarifications or applicability conditions, others relate to the actual rules which all building projects must satisfy. Decomposition of the building code should identify the various types of statements and extract the rule statements. The output of this step will be a human readable text document listing statements and identifying types (e.g. rules, clarifications, applicability conditions). The actors of this step are the building code domain experts appointed by the authority.

4.1.3. Classification of the rule statements

In the last step of the analysis stage, rule statements are classified in order to document how much of the code as well as which types of rules can be modeled. Building codes may include rules that are open to interpretation, uncertain, sometimes even contradictory and impossible for modeling. It is needed to document how much of the code can benefit from automated code compliance checking. Classification of rules according to their formalizability will help to assess potential coverage of building code representations. In addition to the formalizability issue, building codes have a complicated structure. They contain closely related rules that are making exceptions, modifications, or clarifications to other rules as well as stand-alone rules that are unrelated to other rules. It is important to understand the relationship between rules in order to model them correctly. Classification of rules according to their self-containedness will help to figure out relationships between them. The output of this step is a simple table listing all rules that can be represented in computer implementable format. Although the logic embedded in the rules will be analyzed afterwards during the representation stage, this step still needs interdisciplinary knowledge in determining which concepts can be represented in computer implementable format. Cross-disciplinary collaboration should be provided and the building code domain experts should work with software engineers.

4.2. Representation stage

Representation of the building code is the second stage in the proposed building code modeling methodology. In this stage that focuses on modeling, one important concern is about the structure of the building code representation. In the literature, there are two approaches about how the structure of the building code representation should be. Han et al. [1] suggest that the structure of the building code representation should be similar to the structure of the building information model. On the other hand, Nisbet et al. [41] believes the structure of the building code representation should be similar to the structure of the building code. While Han's approach allows for fast rule execution, Nisbet's approach allows for easier code generation and enables higher level of maintainability. In this research Nisbet's approach has been adopted because the main focus is to represent building code rules in a computable format independent of compliance checking systems. If system performance proves to be a serious issue for future compliance checking systems, such systems should be able to employ their own representations of the code that can be derived from a digital representation which is implementation neutral.

After the structure of the building code representation has been determined, the building code is modeled based on the developed representation model. The developed representation model consists of four levels and the representation stages cover respectively these levels as modeling steps.

4.2.1. Representation of domain concepts

The first step in the representation stage is the modeling of the building code domain concepts as object classes that form the domain

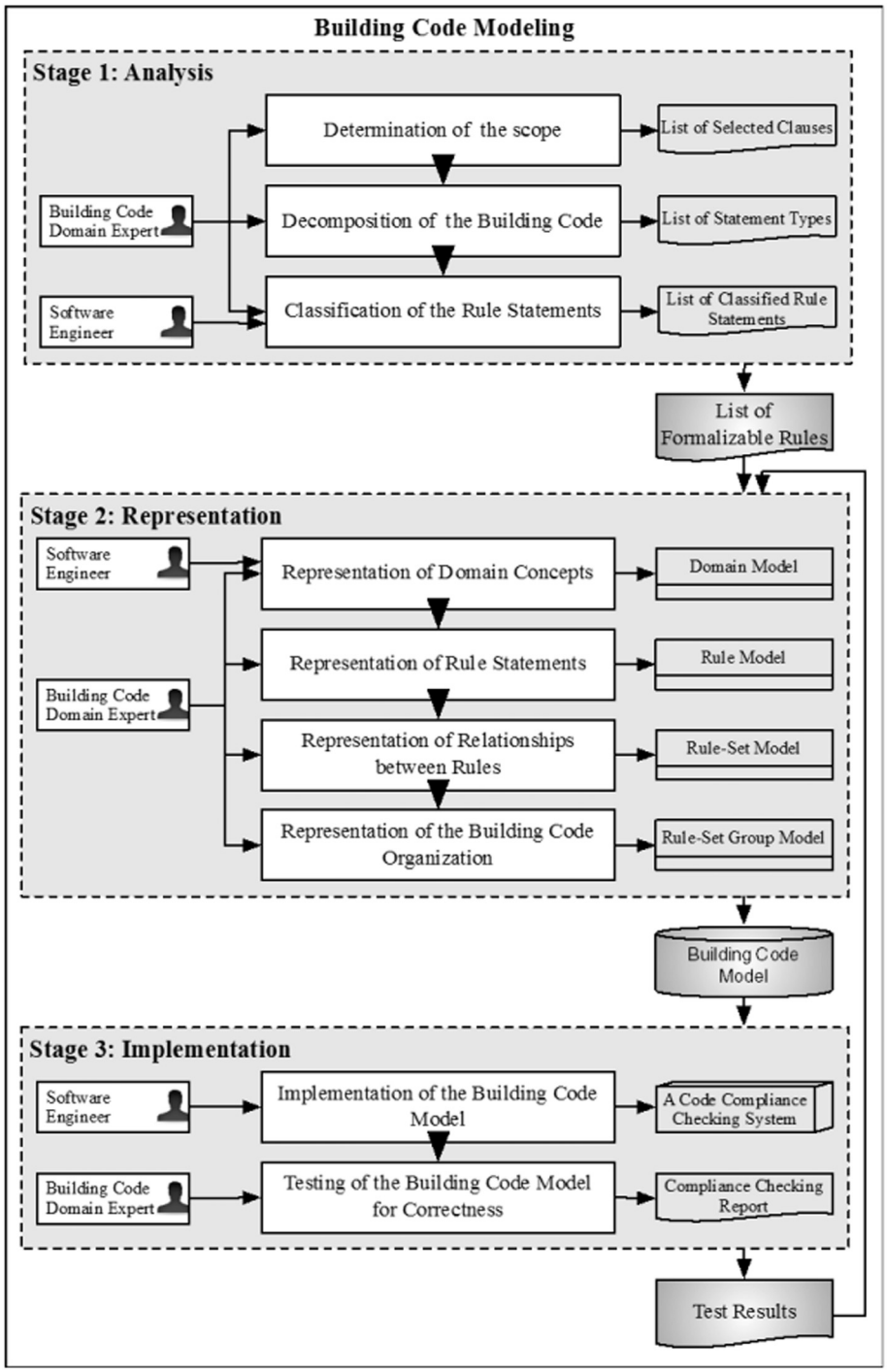

Fig. 2. Stages of the methodology for building code representation.

level. For creating domain object classes, concepts in the building code document are identified and modeled with their attributes and their relationships to each other. In this step, building code domain experts and programming experts work together. This process is undertaken in three stages:

1. Extracting and listing the concepts and entities referenced in the building code document.
2. Identifying the required attributes of the objects and determining the relationships between them.
3. Implementing the representation as a library of objects in a computer-based form. (Existing frameworks such as the IFC may be utilized in this implementation stage.)

The output of this step is a domain model, which will be utilized when modeling rules in the second step. It should be noted that two possible modeling approaches exist. The first involves modeling every concept of building code in a class hierarchy. In this approach specialized concepts are represented as sub-classes of the general concept classes. (e.g., a “kitchen door” can be modeled as a subclass of “door”.) However, this approach tends to increase the complexity of the domain model. The second involves, instead of modeling every concept as a class, the creation of a concept-mapping table. This table lists all concepts of the building code and determines how each concept is represented; either as a class or a filtered set of instances within a class. (e.g., a “kitchen door” can map to all instances of “door” objects with the “relatedSpace” attribute value of “kitchen”.) While the first

approach can be useful when modeling simpler code documents, for modeling building codes that include a high number of concepts with complex relationships, the second approach should be adopted since a high number of specialized sub-classes can be avoided.

4.2.2. Representation of rule statements

The second step in the representation stage involves modeling individual rule statements of the building code as rule objects in the form of structured data. The rules are modeled using the rule model schema. In this step, rule statement semantics of the building code are captured in rule objects. These rule objects form the rule level. This process is carried out by building code domain experts and involves breaking down of the building code rule statements into its constructs and modeling rules by utilizing the concepts modeled in the lower domain level. The *subject* properties of both Requirement and Selection constructs have to refer to either a class in the domain model or a concept in the concept mapping table. Selection constructs allow filtering of objects based on a comparator (e.g. < , > , ! =, etc.) The output of this step is a rule model that includes all rule objects, each indicating a single requirement.

4.2.3. Representation of relationships between rules

The third step in the representation stage of the building code modeling methodology involves defining relationships among rule objects modeled in the second step. In this step, related rule objects that are associated with the same subject are collected together into computable rule-sets by using logical conjunctions. These rule-sets form the rule-set level. This process is handled by building code domain experts and involves using two logical conjunctions “AND” and “OR” for connecting rule objects. The output of this step is the rule-set model covering a collection of rule-set objects.

4.2.4. Representation of the building code organization

The fourth step in the representation stage of the building code modeling methodology is the modeling of alternative organizations of the overall building code representation by categorizing rule-set objects modeled in the third step. Multiple categorizations of rule-set objects are possible. One natural method of categorizing rule-set objects is the one reflecting the structure of the original building code document. One alternative method that is appropriate for use by automated code compliance checking systems is grouping of rule-sets according to the concept they are related to. This allows automated checking to easily access all rules that apply to a given object. There may be many other possibilities in grouping rule-sets appropriate to the goal of the system being developed. It is possible to have multiple classifications exist independently at this level. The output of this step is the rule-set classification objects. Building code domain experts carries out this step.

4.3. Implementation stage

During the second stage, which is the *representation stage,* the building code model is prepared. Implementation of this building code model within a compliance checking application is the third and final stage. This stage consists of two steps: 1) Implementation of the building code model and 2) testing and validation.

4.3.1. Implementation of the building code model

It is important to actually implement the building code representation in compliance checking applications as part of the development process. Through the implementation process, several ambiguities, unclear points, missing definitions (concept, rule, relationship), and insufficient scope definitions can be revealed. Answers to questions on data availability can only be validated through implementation.

One important issue during the implementation of the building code representation in a compliance checking application is related to the identification of building information modeling requirements for the building code domain. Building information models created by a typical BIM platform, to date, do not include the level of detail needed for most building codes. Modeling requirements for building code domains should be identified and used to enhance the BIM standard (IFC) to allow proper data exchange. The domain model which is developed as the first level of the representation, in fact, embodies the modeling requirements for building code domains. How much of the required data can be obtained from the building information models should be analyzed in the implementation stage.

This stage is carried out by software developers and the output of this step is a running system that is able to execute compliance checking of building projects modeled in BIM environments against the building code representation.

4.3.2. Testing and validation

The second step in the implementation stage is testing and validation of the building code model. The Building code model should be evaluated in terms of the three requisite properties found in literature (completeness, uniqueness, and correctness). These requisites are used to evaluate whether building code models are appropriately represented in computational format by most of the research on representation of building codes. These requisite properties are defined as follows [14]:

- Completeness, meaning that the code model can be applied to all possible situations (conditions) within its scope;
- Uniqueness, meaning that the model has no redundant rules and has no contradicting rules and generates the same unique result every time, when applied under a given set of conditions;
- Correctness (clarity), meaning that the result of applying the model must be consistent with the objective of the building code.

Fenves states that completeness and uniqueness are syntactic properties that are related to the organization of the code, while correctness is a semantic property that is more related to the meaning. The new building code representation is based on the RASE constructs and each rule statement is modeled individually. This makes guaranteeing completeness simply a matter of counting the modeled statements and is a major strength of the approach.

Uniqueness ensures that only one rule is applicable for any given situation. Uniqueness can also be defined as the lack of redundancy and lack of contradiction. A rule object is said to be redundant if its applicability conditions are guaranteed to be superseded by other rules. A set of rules is said to be in contradiction when they are all applicable for a given situation (condition). The four-level structure of the new representation follows Nyman's proposed structure for building code representations and the third level which is the rule-set level of the new building code representation is designed to expose the relationships between the rules. Each rule-set deals with a single property of a single domain object and all rules related to the property are collected under a single tree based on the applicability conditions of each rule. Only one rule from the tree is selected as applicable and thus contradictions are not possible. The explicit modeling of conditions in the rule-set tree makes it simple to ensure that there exists a set of conditions for selecting each rule and thus redundancies are avoided. These features of the new representation ensure uniqueness.

Correctness ensures that rule objects represent the meaning, intentions, and implications of the corresponding rule sentences correctly. Completeness and uniqueness are syntactic properties and the representation can guarantee them, but correctness is semantic. The developed building code model should be tested for correctness. The results need to checked and validated by building code domain experts preferably by ones outside the core committee developing the building code representation. Validation should be done using the running compliance checking system built in the first step of the implementation stage. The testing of the building code model should be carried out

using specially prepared test cases.

5. Case study (implementation and evaluation)

In order to provide a proof-of-concept implementation for the new hybrid model, a case study has been conducted. The case study focused on modeling an actual building code and illustrating the use of this model within future compliance checking applications. For the case study, İzmir Municipality Housing and Zoning Code (IMHZCode) has been chosen. IMHZCode is representative of codes that are in effect throughout Turkey. From IMHZCode, the subset of all clauses pertinent to buildings has been modeled. This implementation illustrates the process for representing an existing building code. The case study has been carried out in 3 stages following the proposed building code modeling methodology. Next sections explain these steps in detail.

5.1. Analysis of IMHZCode

In order for the new model to be applicable to as wide a range of code documents as possible, the case study needed to focus on a complex building code with a large set of rules. To determine which building code will be modeled in the case study, current building codes in Turkish Architecture, Engineering and Construction (AEC) industry have been examined. In Turkey, every building project is checked against primarily the housing and zoning code of municipality where the building will be built. The municipalities' housing and zoning codes include rules defined by the ministry documents and add further specifications. Being based on the ministry documents, all housing and zoning codes contain similar rules with few exceptions. Building codes get tested primarily in municipalities of large cities where unforeseen cases and situations come up and force clarifications of code. İzmir is the third most populous city in Turkey and its housing and zoning code is representative of codes that are in effect throughout Turkey. For this reason, İzmir Municipality Housing and Zoning Code (IMHZCode) has been chosen for the case study.

IMHZCode is the legal document that specifies minimum conditions that need to be satisfied by settlements and construction operations within the İzmir Metropolitan Municipality and its environs. Its main structure is divided into six parts. The rules related to buildings are covered by the clauses that are included in part III whereas the rest of the building code is either informative or unrelated to buildings. Part III also includes clauses related to other subjects. Each of the clauses pertinent to buildings consists of several rules defining constraints relating to specific concepts such as roofs, windows, doors, staircases etc. For the case study, IMHZCode's clauses that include rules pertinent to buildings are modeled based on the developed representation model.

After determining the scope, IMHZCode's clauses that include rules pertinent to buildings are extracted. 26 clauses are found on buildings and these clauses are decomposed into a list of statements. As a result of the decomposition study, the statement list containing all 297 individual statements that form the clauses related to buildings is obtained. Afterwards, the type of each statement is determined as being one of “clarification”, “applicability condition”, or “rule”. 258 rule statements are found and they are classified based on the types of rules that have been identified through this analysis. The two classifications have been done; one classification has been based on the structure of the document, and the second classification is based on the rules' formalizability.

Classification of rules based on the code structure is needed for understanding higher-order relationships between rule statements. The analysis of IMHZCode structure has revealed two types of rules; self-contained rules, and linked explanatory rules. *Self-contained rules* indicate how something will be, must be, should be, or can be. *Linked-explanatory rules* are clarifications, exceptions, exemptions, or modifications of other rules. Classification of rules according to their formalizability is necessary to assess the potential coverage of the IMHZCode representation. Three additional types of rules have been identified; formalizable rules, semi-formalizable rules, and non-formalizable rules. *Formalizable rules* are straightforward and can be clearly represented in a computer implementable format. They can be modeled in a single step by the selected representation method. These types of rules allow for automated compliance checking without any ambiguities. *Semi-formalizable rules* contain ambiguous or fuzzy concepts that require human interpretation (e.g. enough, easily, nearly, appropriate, and approximately). These rules require clarification of the concepts involved either during modeling of the rule or later during compliance checking. The required clarifications of concepts are possible by employing objective metrics such as minimum or maximum distances. *Non-formalizable rules* rely on qualitative evaluations such as ones based on aesthetics or characteristics as well as evaluations where local authority is allowed to use initiative. These rules are impossible to represent in computable format and necessitate manual compliance checking under all conditions.

Table 2
Results of the classification of IMHZCode rules.

	Formalizable	Semi-formalizable	Non-formalizable
Self-contained	58% (149)	7% (17)	4% (12)
Linked-explanatory	21% (55)	6% (14)	4% (11)
Total	79% (204)	13% (31)	8% (23)

The classification study revealed that 58% of the 258 rules that are found are self-contained and formalizable and 21% are explicative and formalizable. As indicated in Table 2, 79% of IMHZcode rules on residential buildings can be represented in computer implementable format.

5.2. Representation of IMHZCode

In the second stage of the case study implementation, IMHZCode's all formalizable rule statements on buildings have been modeled based on the new building code representation.

5.2.1. Domain objects and concept mapping list

For creating the IMHZCode domain objects, first, the IMHZCode was scanned manually, statement by statement, concepts and entities in the text are identified (e.g. building, story, space, door, etc.), and all related terms were extracted from it. For example, “construction technique” is a term that is mentioned in IMHZCode and it is an attribute of the “building” concept. There are also terms like height, width, etc. which are mostly used to define requirements. After all terms are extracted, domain objects that represent the identified concepts and entities were determined and modeled as classes with required attributes and relationships to other classes. These classes, which are for utilization by multiple rule objects in the level above, along with the relationships between them form the domain model. When modeling rules, domain classes are used for building applicability and selection constructs. The UML diagram for the resulting IMHZCode domain model is given in Fig. 3.

While concepts generally correspond to a class in the domain model, some concepts correspond to a subset of objects that belong to a class. All objects in the subset are required to hold a specified value for a certain property. For example, a “door” concept is directly represented as a *Door* object with attributes such as *height*, *width*, *relatedSpace*, *allowAirTransfer*. A “bathroom door”, on the other hand is a *Door* with “bathroom” as its *relatedSpace*. These specialized concepts can naturally be modeled through inheritance following the normal object oriented paradigm. However, extending the class hierarchy only for selecting specific subsets may quickly and needlessly increase the complexity of the domain model. All possible variations of doors (“bathroom door”, “kitchen door”, “entrance door”, etc.) should not need to force the

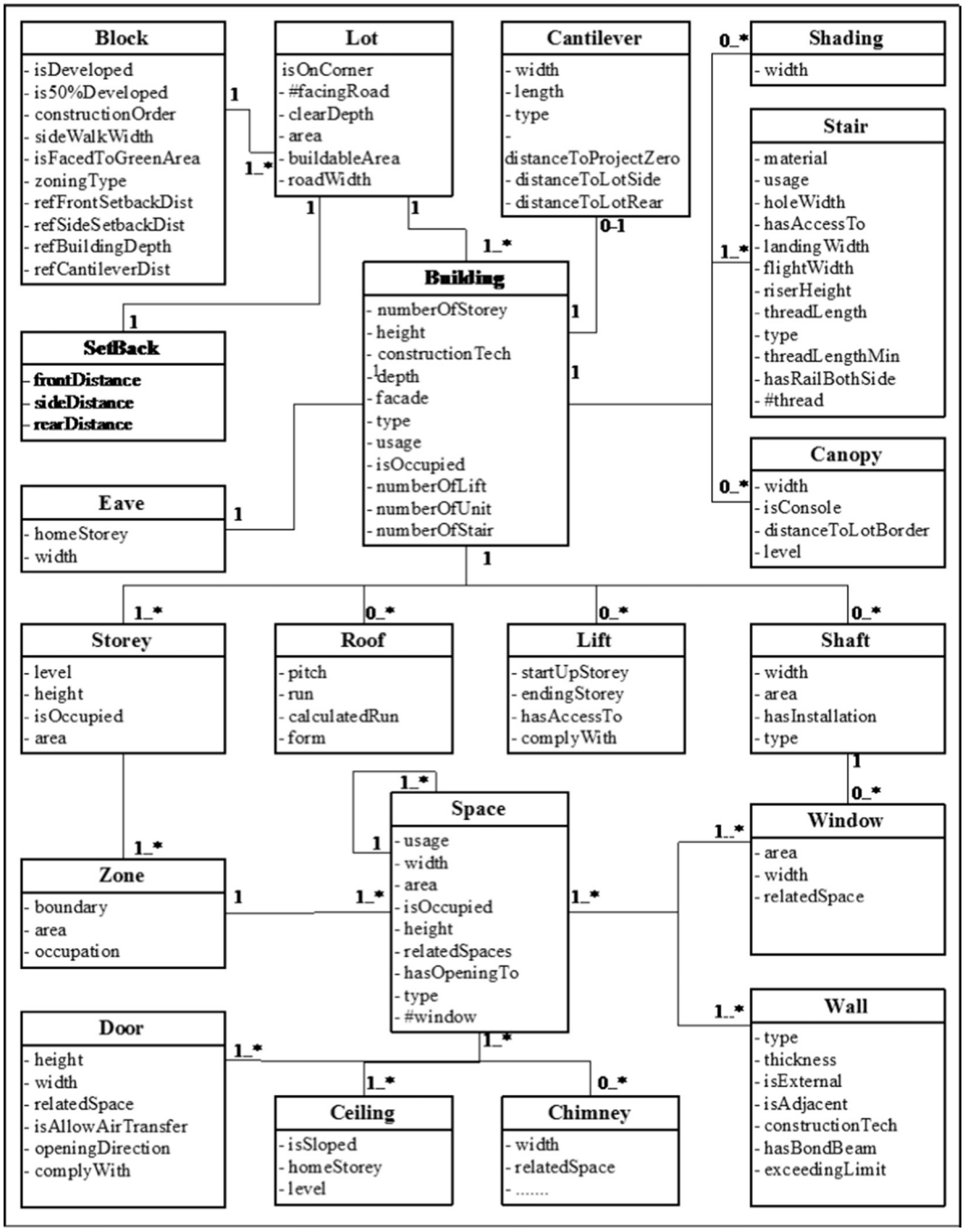

Fig. 3. Domain model of IMHZCode.

modeling of individual classes. The proposed model includes a mapping list for such concepts that are required for the selection of subsets. Instead of modeling every concept as a class, a concept-mapping table is created that defines how a concept maps to a filtered set of objects. The concept mapping table for IMHZCode is shown in Table 3. By associating concepts with a set of selection criteria, the need for a high number of specialized sub-classes derived from the main domain classes was eliminated.

5.2.2. Rule objects

Individual rule statements in 26 clauses of the IMHZCode have been structured using the developed rule model schema as a "semantic rule object". Each rule object has a "requirement" construct that describes the required specification in a concept. Some rule objects also have "selection" constructs describing the specific cases where the requirement is applicable. Both of these constructs have identical structures. They both have the following attributes: A concept, a property, a comparator, a value, and a unit.

The *concept* is a description of the subject to which the rule applies and the *property* is an attribute of the *concept*. The concepts and their properties are defined in the domain model. The requirement constructs must refer to concepts modeled as classes in the domain model. The selection constructs on the other hand may additionally make use of the specialized concepts in the concept mapping table. The *comparator* is a numeric comparison operator such as "≥", "≤", "=", if the value is numeric. If the value is Boolean, then only the "boolean" comparator is used. If the value is descriptive, then the "equal" or "!equal" comparators are used. If the value represents a set of concepts, then the comparator is any of the set comparison operators such as "includes", excludes". The *value* is the specific value that is found in the code,

Table 3
Concept-mapping table for IMHZCode.

Concept	Selection filter			
	Class	Property	Comparator	Value
“attic”	Zone	occupation	equal	attic
“independentUnit”	Zone	occupation	equal	independent unit
“liftShaft”	Shaft	type	equal	lift shaft
“airShaft”	Shaft	type	equal	air shaft
“coalCellar”	Space	usage	equal	coal cellar
“roofTerrace”	Zone	occupation	equal	roof terrace
“stairwell”	Shaft	type	equal	stair shaft
“gableWall”	Wall	type	equal	gable
“mainEntranceDoor”	Door	relatedSpace	equal	mainEntrance
“bathroomDoor”	Door	relatedSpace	equal	bathroom
“kitchenDoor”	Door	relatedSpace	equal	kitchen
“entranceDoor”	Door	relatedSpace	equal	entrance
“roomDoor”	Door	relatedSpace	equal	room
“cellarDoor”	Door	relatedSpace	equal	cellar
“livingRoom”	Space	usage	equal	livingRoom
“kitchen”	Space	usage	equal	livingRoom
“bedroom”	Space	usage	equal	livingRoom
“bathroom”	Space	usage	equal	livingRoom
“basement”	Storey	level	equal	basement
“dwelling”	Zone	occupation	equal	dwellingUnit

whether numeric, descriptive, or Boolean. There are two different kinds of values: Explicit (literal value) and derived (an expression). While explicit value is a constant, derived value is either a reference to another concept's property or the result of a mathematical expression. Curly brackets “{}” are used for specifying references to concepts, and parentheses “()” are used to specify expressions. The *unit* specifies the unit of measure for numeric values. If the value is not numeric the unit is blank.

The rule models of Clause-27 are given in Table 4 as an example for illustrating how rule statements are modeled.

In the code document, individual rule statements generally indicate a single requirement, which has a single subject and a single predicate, associated with a concept. However, some individual rule statements of IMHZCode indicate multiple requirements of a concept or a requirement related to multiple concepts. When modeling, these types of rule statements need to be separated into multiple statements, each targeting a single requirement related to the same concept. Each rule object thus indicates a single requirement and is associated with a single property of a single concept defined in the domain model. The relationships among these individual rule statements are modeled in the levels above.

Table 5
Rule-set objects of IMHZCode Clause-27.

Id	Subject		Set
	Concept	Property	
RS27.A	Setback	frontDistance	(||: R27.1, R27.2, R27.3, R27.4)
RS27.B	Setback	sideDistance	(&: (||: R27.5, R27.6), R27.7)
RS27.C	Setback	rearDistance	(||: R27.8, (&: R27.9, (||: R27.10, R27.11, R27.12)))

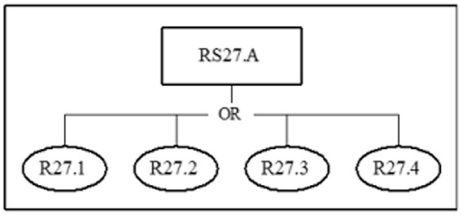

Fig. 4. Tree representation of the rule-set RS27.A.

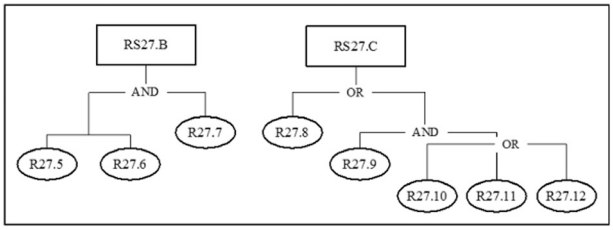

Fig. 5. Tree representation of rule-set RS27.B and RS27.C.

5.2.3. Rule-set objects

IMHZCode is composed of various clauses that include closely related individual rule statements as well as the implicit or explicit information on the relationship among the statements. The rule-set level of the new representation model defines the relationships among individual rule statements. Related rule objects that are associated with the same property of the same concept are collected together and modeled as nested rule-sets using two logical conjunctions: AND, OR. Each top-level rule-set object is given an id, and defines the related

Table 4
Structured rule objects of IMHZCode Clause-27.

Rule	Requirement					Selection			
Id	Concept	Property	C.	Value	U.	Concept	Property	Comp.	Value
R27.1	Setback	frontDistance	≥	5	m				
R27.2	Setback	frontDistance	=	{Block_referencedFrontSetbackDistance}	m	Block	constructionOrder	equal	semiDetached
						Block	hasExistingBuilding	boolean	true
R27.3	Setback	frontDistance	=	{Block_referencedFrontSetbackDistance}	m	Block	constructionOrder	equal	plannedUnit
						Block	hasExistingBuilding	equal	true
R27.4	Setback	frontDistance	=	{Block_referencedFrontSetbackDistance}	m	Block	constructionOrder	equal	attached
						Block	is50%Developed	equal	true
R27.5	Setback	sideDistance	=	3	m				
R27.6	Setback	sideDistance	=	(3 + ((:{Building_numberofStorey}:-4)/2))	m	Building	numberofStorey	≥	4
R27.7	Setback	side Distance	≥	5	m	Building	constTechnique	equal	timberFramed
R27.8	Setback	rearDistance	=	(:{Building_height}:/2)	m				
R27.9	Setback	rearDistance	≥	3	m	Block	hasExistingBuilding	boolean	true
R27.10	Setback	rearDistance	=	{Block_referencedRearSetbackDistance}	m	Block	constructionOrder	equal	semiDetached
						Block	hasExistingBuilding	boolean	true
R27.11	Setback	rearDistance	=	{Block_referencedRearSetbackDistance}	m	Block	constructionOrder	equal	plannedUnit
						Block	hasExistingBuilding	boolean	true
R27.12	Setback	rearDistance	=	{Block_referencedRearSetbackDistance}	m	Block	constructionOrder	equal	attached
						Block	is50%Developed	boolean	true

Table 6
Classification of rule-sets related to setback concept.

Part	Concept	Property - Rule-set	Rule
III - Rules related to buildings and land readjustment			
	Setback		
		frontDistance – [RS27.A = (||: R27.1, R27.2, R27.3, R27.4)]	
			R27.1
			R27.2
			R27.3
			R27.4
		sideDistance – [RS27.B = (&: (||: R27.5, R27.6), R27.7)]	
			R27.5
			R27.6
			R27.7
		rearDistance – [RS27.C = (||: R27.8, (&: R27.9, (||: R27.10, R27.11, R27.12)))]	
			R27.8
			R27.9
			R27.10
			R27.11
			R27.12

concept and property associated with all rules in the set. A rule-set is defined for each concept property that is subject to a requirement in the code document even if there is a single rule object in the set. The rule-set objects of Clause-27 is given in Table 5 as an example.

Some rule sets have a simple, one level relation between rule objects. The rule-set RS27.A is the collection of rules specifying constraints for the *frontDistance* property of the *Setback* concept and is an example for this type of rule-sets (Fig. 4).

Some rule sets have multilevel relations between rule objects. The nested sets of rules form a hierarchical tree-structure. Rule-sets RS27.B, and RS27.C are collections of rules specifying *sideDistance*, and *rearDistance* properties of the *Setback* concept, and are examples for this type of complex rule-sets. (Fig. 5).

5.2.4. Rule-set group objects

The final fourth level (*management level*) of the new representation model, allows for grouping of rule-sets. While a building project must simply be compliant with all rule-sets defined in the third level (*rule-set level*) regardless of how they are grouped, this level allows for modeling the structure of the code document itself as well as the relationships among rule-sets based on any aspect. There may be multiple methods of grouping rule-sets each representing a different sorting scheme. One obvious grouping method is the structure of the code document itself. Rule-sets can be grouped representing the heading and sub-heading based structure of the document. Many other types of relationships and similarities also exist among rules and rule-sets that can be used as criteria for grouping them. Alternative methods of grouping rule-sets are allowed to co-exist. For this case study, rule-sets are grouped based on the concept (corresponds to a class in the domain model) they are related to. In the IMHZCode there are instances where more than one clause is related with the same concept. For example, rules related with the *building* concept are distributed into three clauses. When checking the building objects for compliance it is more efficient to process all applicable rules before moving on to other classes of objects. Table 6 illustrates classification of rule-sets of *setback* concept as an example.

5.3. Implementation

During the representation stage, IMHZCode's all formalizable rule statements on buildings have been modeled based on the new building code representation. Afterwards, the implementation of IMHZCode model and its utilization within a compliance checking application was carried out. A rudimentary code compliance checking system has been implemented as a prototype for the purpose of demonstration and evaluation of the developed model for the representation of building codes. This proof-of-concept prototype demonstrates the feasibility of the developed model.

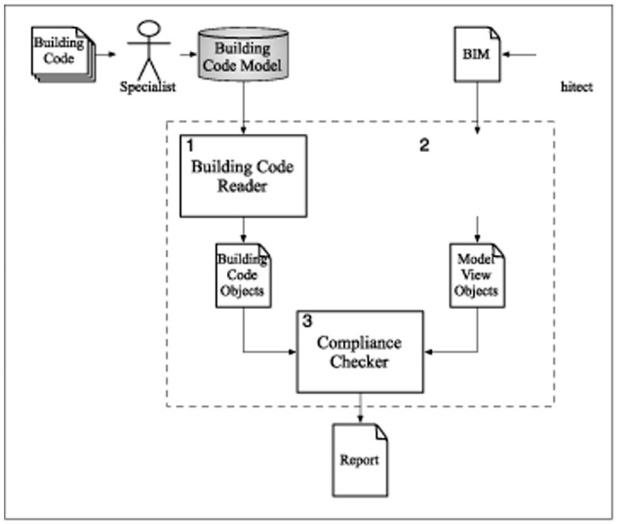

Fig. 6. Conceptual framework for the compliance checking system.

5.3.1. Prototype

The prototype system has been implemented using the Java language and consists of three main components: Building Code Reader, Model View Builder, and Compliance Checker. Building Code Reader reads the building code model from the database and instantiates rule objects for checking, Model View Builder extracts objects and properties of interest from BIM data, populates the domain model objects and thus derives a model view to be checked. Compliance Checker ensures that the building project meets all requirements by applying all rules to related domain concepts and compiles a report. Fig. 6 illustrates the conceptual framework for the compliance checking system.

In this prototype, the building code model is stored in a database. A relational database application is used as a tool to create and store the code model in computational format. Fig. 7 illustrates tables and relations in the IMHZCode model database. Building code authors, without assistance and independent of the checking systems, are able to create new rules and update existing ones. Exchange of the building code can happen through various methods. Future code checking systems may establish live connections and retrieve latest codes from the appropriate authority and the code can be in a number of formats that can represent object oriented data, such as XML.

In the prototype, the Building Code Reader component connects to the database (via a standard JDBC-ODBC bridge) where the building code models are stored, reads from it and instantiates the necessary objects (rules, rule-sets, rule-set groups, etc.).

While rules are stored in a database, the project data to be checked for compliance with the code is in the form of a building information model (BIM) file. In the prototype, the BIM model is required to be an Industry Foundation Classes (IFC2x3) file. The IFC format is utilized by most major research efforts in compliance checking. IFC is currently considered to be the most suitable schema for improving information exchange and interoperability in the construction industry.

Model View Builder component of the prototype accesses and extracts the BIM data that is required during compliance checking. The prototype makes use of JSDAI (Java Standard Data Access Interface) application programming interface for parsing STEP (Standard for the Exchange of Product Model Data – ISO 10303) files. IFC2x3 files exported from the BIM application are parsed using the JSDAI library and IFC objects are created. The IFC objects and their properties of interest are mapped to domain objects and information is copied over to the

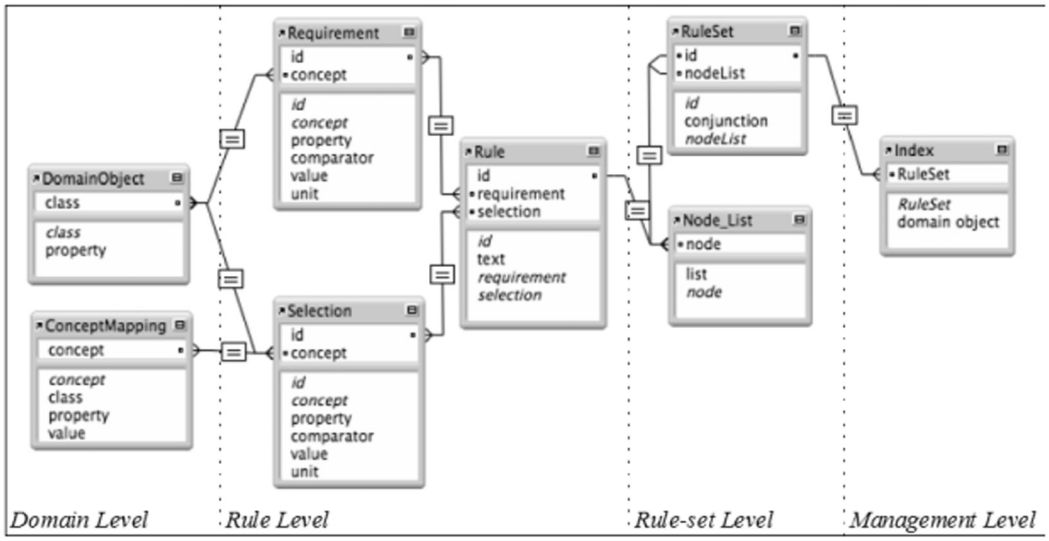

Fig. 7. Database structure for the IMHZCode model.

domain objects and thus the required model view is derived for the Compliance Checker.

The prototype's third major component is the Compliance Checker. It takes the list of "rule-set group objects" that groups rule-sets according to the concept they constrain. For each group it applies all rule-sets to all instances of the concept found in the model view. Finally, the Compliance Checker reports compliant and non-compliant instances of concepts and related rules.

5.3.2. Testing

The successful implementation of the prototype is validation of *completeness* and *uniqueness* of the model but the *correctness* requires testing of the prototype. Correctness ensures that rule objects represent the meaning, intentions, and implications of the corresponding rule sentences correctly. The implementation has been tested on a range of different building projects exported by a BIM tool. Simple building models in IFC format have been prepared and imported to the prototype in order to test the results of compliance checking against the sample IMHZCode model. Results indicate correct modeling of the building code. Fig. 8 is a screenshot where compliance checking of the sample building against IMHZCode clauses related to setback, door and building concepts takes place. In this example, the door object D006 is not valid. It does not pass the check on rule-set RS.47.B (related to door widths) because of rule R.47.06. There are seven rules restricting minimum door width. Because the D006 is a bathroom door, the system applies R.47.06, which is the correct rule for bathroom doors.

6. Conclusion

The research presented in this paper was aimed at developing a building code representation that is independent of checking systems, concise (avoids redundancies), consistent (does not allow ambiguities and contradictions), comprehensive (able to represent all types of information) and easy to maintain (can be updated by non-programmer code-authors). To achieve this goal, a new hybrid model and an accompanying methodology has been developed. The hybrid model takes the RASE constructs from the SMARTcodes project's [27] semantic-oriented representation and adapts them to work within a four level structure theoretically grounded in Nyman and Fenves [46]'s work on an abstract model of the logical structure of building codes.

Fenves and Nyman's work provides a solid theoretical foundation for representation of building codes. The four-level structure they

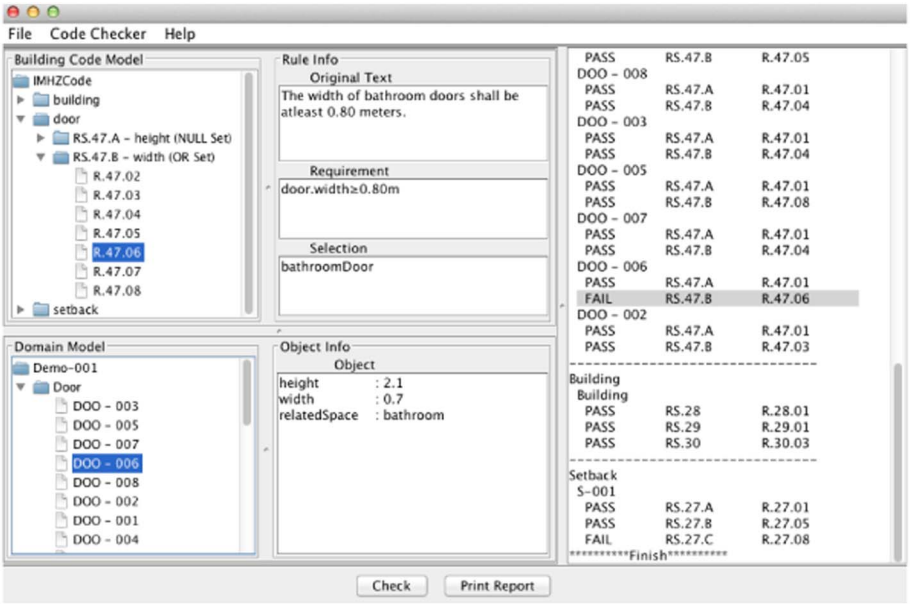

Fig. 8. Screenshot of the testing application.

introduced is still applicable and has proven to be a robust method for decomposing building codes. However, previous building code models based on this theory were not widely adopted in AEC industry mainly due to the immature information technologies for knowledge representation at the time. The hybrid model adopts the four-level paradigm and combines it with the relatively recent semantic modeling method.

SMARTcodes' semantic-oriented representation approach, which is a relatively new method for knowledge representation, provides an easy to understand, elegant method for modeling rule statements. It utilizes a simple scheme (RASE constructs) that is applicable for all types of rule statements, and with it, non-programmer users are able to build and maintain building code models. A pilot study carried out in the early phase of the research showed that it can be utilized for representing building codes, however, several shortcomings of the RASE methodology were identified. First, it is prone to inconsistencies and creates redundancies arising from modeling the same concepts multiple times for each applicability and selection construct when the concepts are referenced by multiple rules. This shortcoming was addressed by the first level of the new representation model creating domain objects of interest and concept mapping table that are used to define repeating concepts once. Second shortcoming is the lack of explicit relationships between individual rule statements. In SMARTcodes' approach, relationship representation (hierarchy within rules) is handled separately in the automated compliance checking system. This makes it difficult to ensure correctness and consistency for the overall code representation, independent of automated checking systems. The third level of the new model addresses this shortcoming by allowing to build the logical hierarchy relationships between rules. As a result, while SMARTcode's approach was adopted, it was modified into a four level representation that improves it by eliminating redundancies and adding logical relationships.

A modeling methodology is also defined for creating digital versions of building codes, based on the new hybrid representation. The methodology is comprised of three process stages; (1) analysis of the building code and determining scope, (2) representation of the building code, (3) implementation and testing of the building code model within a compliance checking application.

This research shows that decomposing a building code into four levels and modeling rules based on the semantic-oriented paradigm is an effective modeling strategy for representing building codes in a computable format independent of automated compliance checking systems. The four levels are: The domain level, the rule level, the ruleset level and the management level. Since concepts, individual rule statements, relations between the rule statements and organization of the building code are separately represented, required changes to the code model due to revisions of the building code can be localized and handled without affecting future automated checking systems that will be utilizing this hybrid building code representation.

Handling rules related to geometric and spatial relationships is another important issue. The four level approach allows representing and processing geometry and graphs at the domain level to implement the “compute once, use many times” strategy. The advantages of this strategy is described by Solihin and Eastman [49]. Computation may take place during the initial processing of the BIM model to populate the domain objects which the rules are built on. The domain objects may include concepts based on a specific set of objects, geometric relationships and even performance indicators which the rules refer to.

One limitation of the research is the fact that testing has been carried out only with the IMHZCode. Modeling of other code documents is necessary. Yet, it should be noted that IMHZCode, as a code that is actually in effect and belonging to a metropolis, is comprehensive and includes rule statements that have a high level of complexity. Thus, it is representative of codes that are hard to represent in computational format.

Another limitation is that only formalizable rules have been considered in this study. Formalizable rules constitute 79% of all rules in IMHZCode. This is considerable but future research needs to look into the handling of semi-formalizable and even non-formalizable rules. It should be noted that even the formalizable rules cannot all be automatically checked due to the fact that current BIM definitions do not hold all of the required data. The above figure of 79% is strictly based on what is theoretically calculable. Other challenging directions for future research include: 1) Investigating the extent to which domain models can be automatically derived from original texts of building codes by applying advanced artificial intelligence and natural language processing techniques, 2) Investigating effective methods of extending building information models to hold information requirements of all sorts of building code domains, and 3) Investigating efficient checking algorithms for the purpose of determining effective building code representations and building information models.

Acknowledgments

We would like to thank Mustafa Emre İlal and Georg Suter for their support and contributions to the research.

The research is supported in part by the Scientific and Technological Research Council of Turkey (TÜBİTAK) via the 2214 - International Research Fellowship Programme (for PhD Students) 2010/2 (Grant No: 1059B141000315).

References

[1] C.S. Han, J.C. Kunz, K.H. Law, Client/server framework for on-line building code checking, J. Comput. Civ. Eng. 12 (4) (1998) 181, http://dx.doi.org/10.1061/(ASCE)0887-3801(1998)12:4(181).
[2] N. Nawari, Smartcodes and BIM, in: B.J. Leshko, J. McHugh (Eds.), Structures Congress, 2013, pp. 928–937, , http://dx.doi.org/10.1061/9780784412848.082 Pittsburgh, Pennsylvania, United States.
[3] C.M. Eastman, J.-m. Lee, Y.-S. Jeong, J.-k. Lee, Automatic rule-based checking of building designs, Autom. Constr. 18 (8) (2009) 1011–1033, http://dx.doi.org/10.1016/j.autcon.2009.07.002.
[4] L. Ding, R. Drogemuller, J. Jupp, M.A. Rosenman, J.S. Gero, Automated code checking, in: K. Hampson (Ed.), Clients Driving Innovation Conference, CRC Construction Innovation, Gold Coast, Queensland, Australia, 2004, http://www.construction-innovation.info/indexe830.html?id=803.
[5] A. Yurchyshyna, A. Zarli, An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction, Autom. Constr. 18 (8) (2009) 1084–1098, http://dx.doi.org/10.1016/j.autcon.2009.07.008.
[6] E. Hjelseth, A. Dikbas, E. Ergen, H. Giritli (Eds.), Foundation for Development of Computable Rules, 26th CIB W78 Conference on Construction IT, 2009, pp. 257–267 Istanbul, Turkey http://itc.scix.net/data/works/att/w78-2009-1-25.pdf.
[7] Q.Z. Yang, X. Xu, Design knowledge modeling and software implementation for building code compliance checking, Build. Environ. 39 (6) (2004) 689–698, http://dx.doi.org/10.1016/j.buildenv.2003.12.004.
[8] T.-H. Nguyen, A.A. Oloufa, K. Nassar, Algorithms for automated deduction of topological information, Autom. Constr. 14 (1) (2005) 59–70, http://dx.doi.org/10.1016/j.autcon.2004.07.015.
[9] X. Tan, A. Hammad, P.E. Paul Fazio, Automated code compliance checking for building envelope design, J. Comput. Civ. Eng. 24 (2) (2010) 203–211, http://dx.doi.org/10.1061/(ASCE)0887-3801(2010)24:2(203).
[10] H.-H. Wang, F. Boukamp, Ontology-based representation and reasoning framework for supporting job hazard analysis, J. Comput. Civ. Eng. 25 (6) (2011) 442–456, http://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000125.
[11] S.J. Fenves, Tabular decision logic for structural design, J. Struct. Div. ASCE 92 (1966) 473–490 http://cedb.asce.org/CEDBsearch/record.jsp?dockey=0014476.
[12] S.J. Fenves, E.H. Gaylord, S.K. Goel, Decision Table Formulation of the 1969 AISC Specification, in Civil Engineering Studies SRS-347. http://hdl.handle.net/2142/14275, (1969).
[13] D.J. Nyman, S.J. Fenves, R.N. Wright, Restructuring study of the AISC specification, Civil Engineering Studies SRS-393, Department of Civil Engineering, University of Illinois Engineering Experiment Station, Urbana-Champaign, 1973, http://hdl.handle.net/2142/13806.
[14] S.J. Fenves, R.N. Wright, F.I. Stahl, K.A. Reed, Introduction to SASE: Standards Analysis, Synthesis and Expression, NBSIR, Editor, National Bureau of Standards, Washington, D.C., 1987, http://dx.doi.org/10.6028/NBS.IR.87-3513.
[15] M.A. Rosenman, J.S. Gero, Design codes as expert systems, Comput. Aided Des. 17 (9) (1985) 399–409, http://dx.doi.org/10.1016/0010-4485(85)90287-8.
[16] C.L. Dym, R.P. Henchey, E.A. Delis, S. Gonick, A knowledge-based system for automated architectural code checking, Comput. Aided Des. 20 (3) (1988) 137–145, http://dx.doi.org/10.1016/0010-4485(88)90021-8.
[17] W. Rasdorf, T. Wang, Generic design standards processing in an expert system environment, J. Comput. Civ. Eng. 2 (1) (1988) 68–87, http://dx.doi.org/10.1061/

(ASCE)0887-3801(1988)2:1(68).
[18] B. Kumar, Knowledge Processing for Structural Design, Computational Mechanics Publications, Southampton, UK, 1995http://hdl.handle.net/1842/7492 ISBN 1853123765.
[19] D. Jain, K.H. Law, H. Krawinkler, On Processing Standards with Predicate Calculus, in: T.D. Barnwell (Ed.), Sixth Conference on Computing in Civil Engineering, ASCE, Atlanta, Georgia, 1989, pp. 259–266 http://cedb.asce.org/CEDBsearch/record.jsp?dockey=0063010.
[20] W.J. Rasdorf, S. Lakmazaheri, Logic-based approach for modeling Organization of Design Standards, J. Comput. Civ. Eng. 4 (2) (1990) 102–123, http://dx.doi.org/10.1061/(ASCE)0887-3801(1990)4:2(102).
[21] R. Sharpe, et al., Bcaider: Pc Software to Help Building Code Users and Developers, CSIRO. Division of Building, Construction and Engineering, 1991, https://books.google.pt/books?id=LCbTZwEACAAJ ISBN 0858255367.
[22] J.H.J. Garrett, M.M. Hakim, Object-oriented model of Engineering design Standards, J. Comput. Civ. Eng. 6 (3) (1992) 323–347 http://dx.doi.org/10.1061/(ASCE)0887-3801(1992)6:3(323.
[23] M. de Waard, Computer aided conformance checking, Computers and Building Standards Workshop, Research Press of the National Research Council of Canada, Montreal, Canada, 1992, https://www.irbnet.de/daten/iconda/CIB12661.pdf.
[24] N. Yabuki, K.H. Law, An object-logic model for the representation and processing of design Standards, Eng. Comput. 9 (3) (1993) 133–159, http://dx.doi.org/10.1007/BF01206345.
[25] H. Kiliccote, J.H. Garrett, Standards modeling language, J. Comput. Civ. Eng. 12 (3) (1998) 129–135, http://dx.doi.org/10.1061/(ASCE)0887-3801(1998)12:3(129).
[26] S. Kerrigan, K.H. Law, Logic-Based Regulation Compliance-Assistance, 9th International Conference on Artificial Intelligence and Law, ACM, Scotland, United Kingdom, 2003, pp. 126–135, , http://dx.doi.org/10.1145/1047788.1047820.
[27] AEC3 International Code Council, [February, 23, 2013], Available from: http://www.aec3.com/en/5/5_013_ICC.htm, (2012).
[28] D. Conover, Method and Apparatus for Automatically Determining Compliance with Building Regulations, Washington, DC, US https://www.google.ch/patents/US20090125283, (2009).
[29] A. Yurchyshyna, C. Faron-Zucker, N.L. Thanh, A. Zarli, Towards an ontology-enabled approach for modeling the process of conformity checking in construction, in: Z. Bellahsene, et al. (Ed.), CAiSE Forum, 2008, pp. 21–24 Montpellier, France http://ceur-ws.org/Vol-344/paper6.pdf.
[30] P. Pauwels, D.V. Deursen, R. Verstraeten, J.D. Roo, R.D. Meyer, R.V.d. Walle, J.V. Campenhout, A semantic rule checking environment for building performance checking, Autom. Constr. 20 (5) (2011) 506–518, http://dx.doi.org/10.1016/j.autcon.2010.11.017.
[31] J. Dimyadi, P. Pauwels, M. Spearpoint, C. Clifton, R. Amor, Jakob Beetz, L.V. Berlo, T. Hartmann, R. Amor (Eds.), Querying a regulatory model for compliant building design audit, 32nd International CIB W78 Conference, 2015, pp. 139–148 Eindhoven, the Netherlands http://hdl.handle.net/1854/LU-6890582.
[32] T.H. Beach, Y. Rezgui, H. Li, T. Kasim, A rule-based semantic approach for automated regulatory compliance in the construction sector, Expert Syst. Appl. 42 (12) (2015) 5219–5231, http://dx.doi.org/10.1016/j.eswa.2015.02.029.
[33] C.P. Cheng, G.T. Lau, K.H. Law, J. Pan, A. Jones, Improving access to and understanding of regulations through taxonomies, Gov. Inf. Q. 26 (2) (2009) 238–245, http://dx.doi.org/10.1016/j.giq.2008.12.008.
[34] D.M. Salama, N.M. El-Gohary, Semantic Modeling for Automated Compliance Checking, Computing in Civil Engineering (2011), (2011), pp. 641–648, http://dx.doi.org/10.1061/41182(416)79.
[35] J. Zhang, N.M. El-Gohary, P. Morand, A. Zarli (Eds.), Automated Information Extraction from Construction-Related Regulatory Documents for Automated Compliance Checking, CIB W78-W102 Conference, 2011, pp. 1–10 Sophia Antipolis, France http://itc.scix.net/cgi-bin/works/Show?w78-2011-Paper-99.
[36] T. Liebich, J. Wix, J. Forester, Z. Qi, Z. Turk, R.J. Scherer (Eds.), Speeding-Up the Building Plan Approval - The Singapore E-Plan Checking Project Offers Automatic Plan Checking Based on IFC, European Conferences on Product and Process Modelling (ECPPM) 2002 - eWork and eBusiness in Architecture, Engineering and Construction, eWork and eBusiness in Architecture, Engineering and Construction, Portoroz, Slovenia, 2002, pp. 467–471 https://books.google.com.tr/books?id=bzFkxrFrYGAC.
[37] buildingSMART, Industry Foundation Classes (IFC) Data Model, [14 May 2014], Available from: http://www.buildingsmart.org/standards/ifc, (2008).
[38] L. Ding, R. Drogemuller, M. Rosenman, D. Marchant, J. Gero, K. Brown, K. Hampson, P. Brandon (Eds.), Automating Code Checking for Building Designs – Designcheck, in Clients Driving Innovation: Moving Ideas into Practice, Cooperative Research Centre (CRC) for Construction Innovation, Gold Coast, Queensland, Australia, 2006, pp. 113–126 http://ro.uow.edu.au/cgi/viewcontent.cgi?article=7773&context=engpapers.
[39] Jotne, Express Data Manager, [14 May 2014], Available from: http://www.epmtech.jotne.com, (1994).
[40] D. Conover, Development and Implementation of Automated Code Compliance Checking in the U.S. International Code Council, 2007, http://projects.buildingsmartalliance.org/files/?artifact_id=1610.
[41] N. Nisbet, J. Wix, D. Conover, The Future of Virtual Construction and Regulation Checking, Virtual Futures for Design, Construction & Procurement, Blackwell Publishing Ltd, 2009, pp. 241–250, http://dx.doi.org/10.1002/9781444302349.ch17.
[42] C. Eastman, Automated assessment of early concept designs, Archit. Des. 79 (2) (2009) 52–57, http://dx.doi.org/10.1002/ad.851.
[43] S.J. Fenves, J.H. Garrett, H. Kiliccote, K.H. Law, K.A. Reed, Computer representations of design standards and building codes: U.S. perspective, Int. J. Constr. Inf. Technol. 3 (1) (1995) 13–34 http://fire.nist.gov/bfrlpubs/build95/PDF/b95012.pdf.
[44] H. Kiliccote, A Standards Processing Framework in Department of Civil and Environmental Engineering, 304 Carnegie Mellon University, Pittsburgh, 1997http://dl.acm.org/citation.cfm?id=266586.
[45] S.J. Fenves, R.N. Wright, The Representation and Use of Design Specifications, in NBS Technical Note 940, National Bureau of Standards, Washington, DC, 1977https://archive.org/details/representationus940fenv.
[46] D.J. Nyman, S.J. Fenves, An organization model for design specifications, J. Struct. Div. ASCE 101 (4) (1975) 697–716 http://cedb.asce.org/CEDBsearch/record.jsp?dockey=0005899.
[47] M.M. Hakim, J.H. Garrett, Issues in modelling and processing design standards, in: D. Vanier, R. Thomas (Eds.), The Joint CIB Workshops on Computers and Information in Construction, 165 CIB Publication, Montreal, Canada, May 1992, pp. 242–257 http://itc.scix.net/cgi-bin/works/Show?w78-1992-242.
[48] S. Macit, M.E. İlal, H.M. Günaydın, G. Suter, G. Suter, Y. Rafiq, P. de Wilde (Eds.), Izmir Municipality Housing and Zoning Code Analysis and Representation for Compliance Checking, 20th Workshop of the European Group for Intelligent Computing in Engineering, 1–3 EG-ICE, Vienna, Austria, July 2013, https://www.researchgate.net/publication/310137845.
[49] W. Solihin, C. Eastman, Classification of rules for automated BIM rule checking development, Autom. Constr. 53 (2015) 69–82, http://dx.doi.org/10.1016/j.autcon.2015.03.003.