WESTERN MICHIGAN UNIVERSITY

Bronco Construction Research Center

Title: **A Holistic Framework to Support Compliance Checking in the Construction Domain**

PI(s): **Wuwei Shen**

Report Number (*use your project number here)*: **17-4**

Date: **1/10/2019**

Project Title: A Holistic Framework to Support Compliance Checking in the Construction Domain

Final Report

Principal Investigators Wuwei Shen

Date: Dec 2018

Abstract Computer use is pervasive in our daily life and the increasing demand for computer applications has penetrated various domains. Construction industry has become one of the domains which is more reliable on the application of computers to implement regulatory compliance checking. Like many safety critical domains, the construction domain has its own set of international building codes on construction projects which must be complied. With the increasing complexity of construction projects, many manual compliance checking techniques have shown some serious issues. First, the manual techniques are error-prone due to human errors. Second, the complexity of a construction project exceeds the human limit to deal with. Third, the evolution of a construction project is inevitable, and the human maintenance of a construction project is almost impossible because either the memory of the original project design has faked away, or some development team members are gone. So, it has become a new trend to employ computers to support automatic regulatory compliance checking in construction industry. In this project, we propose a novel framework to support compliance checking with the emphasis on the foundation of automatic regulatory compliance checking to certify whether a construction project complies with some international building codes.

1 Introduction

Computer use is pervasive in our daily life and the increasing demand for computer applications has penetrated various domains. Construction industry has become one of the latest industries to be more reliable on the application of computer to implement regulatory compliance checking. Like many safety critical domains, the construction domain has its own set of international codes on construction projects which must comply with. With the increasing complexity of construction projects, many manual compliance checking techniques have some serious issues [1]. First, the manual techniques are error-prone due to human errors. Second, the complexity of a construction project exceeds the human limit to deal with. Usually, a construction project should consider a set of international codes for different purposes. For instance, a construction project should comply with ICC 2009 [2] as part of the overall construction requirement and Energy code [3] for the energy conservation purpose. Third, the evolution of a construction project is inevitable, and the human maintenance of a construction project is almost impossible because either the memory of the original project design has faked away, or some development team members are gone. So, it has become a new trend to employ computers to support automatic regulatory compliance checking in construction Industry.

Compliance checking is not new in the computer science community since more and more safety critical industries require compliance checking [4, 5, 6, 7, 8, 9, 10]. One of the most important reasons is that the failure of a project in these industries can have a serious consequence such as loss of life. Central to the compliance checking is that developers must demonstrate that a system or a project indeed complies with the relevant governmental or international standard documents. Thanks to advance in Model Driven Engineering (MDE) [11], the automatic compliance checking has become a feasible means in support of compliance checking in various safety critical domains [12, 13, 14, 15, 16, 17, 18, 19, 20, 21].

Like compliance checking in the safety critical domains, the compliance checking in the construction industry has several obstacles which should be overcome. First, the ambiguity issue must be solved since most international codes in construction industry are written in a natural language like English [2]. Second, heterogeneity of the representation of construction projects from various construction companies when carrying out regulatory compliance checking is another touchy issue. Third, an appropriate integration methodology of a construction project into a set of international codes in construction industry should be sought.

In this project, we propose to apply the conformance checking, which has been widely used in the Model Driven Architecture (MDA) [11], in support of automatic regulatory compliance checking in construction industry. Conformance checking aims to ensure that an instance model conforms to its original model. Due to the four levels of models, conformance checking can be performed at various levels. In the construction industry context, we will apply conformance checking to level zero and one. Namely, a domain model at level one is used to model a set of international codes and represented in a class diagram in the Unified Modeling Language (UML) [22]. An instance model denotes a specific construction project to be checked against international codes. In MDA, a domain model usually includes constraints in the Object Constraint Language (OCL) [23] since many kinds of constraints in a class diagram cannot be represented. The conformance checking in MDA ensures that an instance model is a valid instance of the domain model. In other words, an instance model should satisfy all constraints in OCL as well as constraints given in a class diagram such as the multiplicity restriction if the instance model conforms to a class diagram. In this case, regulatory compliance checking of whether a project complies with a set of international codes is achieved.

Conformance checking at level one and zero is appropriate for conformance checking in construction industry due to the nature of international codes. Unlike some governmental and international standard documents, international codes in construction industry place more specific restrictions on a construction project. For instance, ICC 2009 requires "*Interior spaces intended for human occupancy shall be provided with active or passive heating systems capable of maintaining a minimum indoor temperature of 68 degree at a point 3 feet above the floor on the design heating day"* [2]. Obviously, this requirement can be well denoted by a class diagram, i.e. a domain model, where the constraint on an indoor temperature can be converted to an OCL constraint. Furthermore, a specific construction project can easily be converted to an instance model of a domain model, which models a set of international codes with which the project must comply. As a result, conformance checking leverages the capability of compliance checking in construction industry by ensuring that a specific construction project satisfies all constraints in a class diagram. To lay out the foundation of a framework to support automatic regulatory compliance checking in construction industry, we formalize compliance checking by means of conformance checking in this report.

The report is organized as follows. Section 2 presents some relevant work in support of compliance checking in various domains. Section 3 illustrates an example which demonstrates how conformance checking contributes to regulatory compliance checking in construction industry. Section 4 outlines the

application of the UML class diagram to formalize the international building code 2009 (chapter 19, section 1904) [2]. Section 5 presents the framework to support the compliance checking where two different methods via Eclipse EMF and programming are introduced respectively. In section 6, we lay out the theoretic foundation of automatic regulatory compliance checking by formalization of conformance checking. We finally draw a conclusion and present some future work in Section 7.

2 Related Work

Compliance checking has been widely discussed in the construction community and manual checking of compliance checking has been proved to be time consuming, error prone, and expensive. Instead, automatic compliance checking has been proposed to tackle these problems and researchers in the construction community have presented various techniques in support of automatic compliance checking [24]. Tan et al proposed an integration approach which combines building envelope design with building codes and simulation by means of decision tables [18]. Specifically, a building code is used to produce decision tables while information of a building project is represented as a tree-like structure via an Extended Building Information Model (EBIM) [25] including the building simulation output. Rules in the decision tables are checked for the design facts shown in EBIM to validate whether the building project complies with the building code. Ding et al proposed an approach to represent building codes via object-based rules and design via an Industry Foundation Classes (IFC)-based internal model [26] to support compliance checking according to accessibility regulations. The Construction and Real Estate Network (CONENET) [27] project of Singapore employed a method to use semantic object in the FORNAX library to represent design information while properties and functions in FORNAX objects are used to denote regulatory rules [28]. The SMRTcodes project (International Code Council (ICC) 2012 [29]) of the ICC adopted an approach to represent ICC codes in a tuple format and represent designs using an IFC-based model for automatic compliance checking. An approach given by Zhang et al concentrated on the integration of NLP and logic reasoning for automatic compliance checking [30]. Specifically, they employed NLP to generate a Prolog program to model a building code while converting design information for a construction project to a set of facts so running of the Prolog program can achieve the goal of compliance checking.

On the other hand, conformance checking is not new in the MDE community [31, 32]. The UML specification presents a metamodel for UML diagrams and the

specification includes well-formedness rules to enforce the constraints on UML diagrams back to the first version [33]. Thus, conformance checking has been proposed to ensure that a UML model satisfies the UML metamodel according to the UML specification. Various tools have been implemented to support the syntax checking for a UML model by means of conformance checking which enforces not only multiplicity but also all OCL constraints in the UML metamodel [34].

With the rapid development of MDE, the UML specification allows to define a domain specific language via the UML profile mechanism. The UML profile mechanism facilitates conformance checking in support of a user defined metamodel which extends the UML metamodel [35]. In this way, developers can define a specific language via a UML profile and apply conformance checking to ensure that a specific model satisfies the UML project, i.e. the specific language, according to some purpose. For instance, OMG published a UML testing profile dedicated to Model-based testing. Conformance checking thus ensure whether a specific testing procedure follows the UML testing profile or not. Furthermore, conformance checking has been used to support forward engineering recently. As progress has been made in MDE in the past decade, many tools have the forward engineering feature which translate a design model into some executable skeletal code such as C program and Java program to reduce the implementation time. What programmers do is to fill out the detailed implementation in the skeletal code. Thus, a state captured during an execution time can be checked against its original class diagram via conformance checking. In this way, conformance checking ensures that each valid program state does not violate the class diagram given by the design phase [36].

3 An Illustrative Example of Compliance Checking

One challenging issue in support of regulatory compliance checking is how to denote the regulatory requirements given in a set of international codes. Due to the ambiguity issue in a document written in a natural language, various techniques have been proposed. In this report, we employ a class diagram in UML to formalize the international codes in construction industry. However, before presenting the formalization of conformance checking, we would like to illustrate how a UML class diagram models the text in an international code such as ICC 2009.

To formalize the text in an international code, we need to read carefully the code's text to identify all concepts and their relationships. To systematically

analyze the text, we first label an important noun or noun phrase as a concept and create a definition in a glossary if the noun or noun phrase is first encountered. We also study the relationship between these concepts to create an association relationship. In doing so, we can create a class diagram to model the text of an international code in construction domain.

As an example, we consider the text retrieved from section 1204.1 in ICC 2009, which has the following text: "*Interior spaces intended for human occupancy shall be provided with active or passive heating systems capable of maintaining a minimum indoor temperature of 68 degree at a point 3 feet above the floor on the design heating day"*. In this text, we retrieve the following concepts: interior spaces, heating systems, design heating day, temperature, and above the floor. They are converted to UML classes *InteriorSpace*, *HeatingSystem*, *HeatingDay* and *Temperature* respectively. Specifically, we have attributes *temp* and *locAboveFlr* to denote the temperature at the specific point above the floor in class *Temperature*. Likewise, we have attribute *isHD* to denote whether a heating day is a design heating day or not in class *HeatingDay*. Also, we find an association between classes *InteriorSpace* and *HeatingSystem* to denote "*Interior spaces ... shall be provided with active or passive heating systems."* The association between classes *InteriorSpace* and *Temperature* denotes the temperature at a specific location in an interior space required by the text "*…maintaining a minimum indoor temperature of 68 degree at a point 3 feet above the floor on the design heating day".* Likewise, from the same part of the sentence, we have an association between classes *Temperature* and *HeatingDay* to model the temperature on a heating day. A complete class diagram is illustrated in Figure 1(i) to illustrate the main concepts as well as their relationship extracted from the text from Section 1204.1.

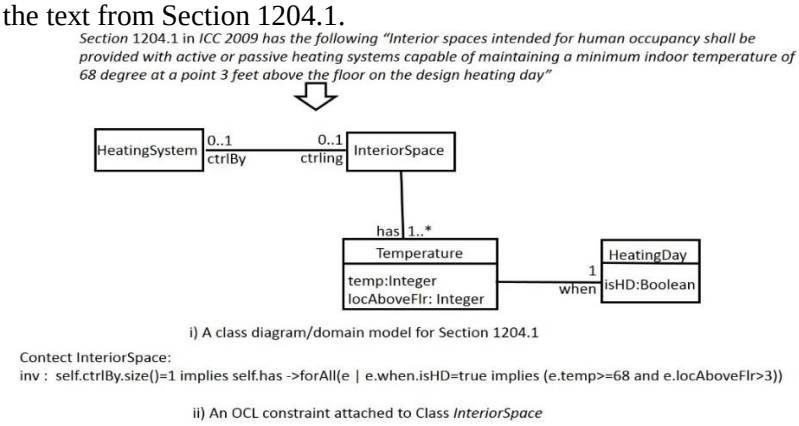

Figure 1 A domain model and an OCL constraint for Section 1204.1

However, some constraints represented in the text of an international code cannot be represented in a UML class diagram. In this case, we need to employ OCL to denote such constraints. For instance, in the 1204.1 section, we cannot represent the restriction on the minimum indoor temperature. Thus, we give an OCL constraint to enforce the restriction which is shown in Figure 1(ii).

Once a class diagram is given based on the text of an international code, we can perform the automatic compliance checking by means of conformance checking in MDE to validate whether a construction project satisfies the code or not. In this way, we convert information from a construction project into an instance diagram and employ the conformance checking to achieve the compliance checking purpose. As an illustrative example, Figure 2(i) shows an example of a construction project which complies with the text of Section 1204.1 while Figure 2(ii) shows an example of another construction not complying with the same section. The former example shows two temperatures measured on two different design heating days and both temperatures satisfies the restriction of Section 1204.1. However, the latter example does not satisfy the OCL constraint since an instance of class *InteriorSpace*, i.e. object *t2*, has 60 degree as the value of the temperature at the point of 4 feet above the floor on a design heating day and this does not satisfy the OCL constraint; while the other temperature has the value of 76 degree at the point of 5 feet above the floor on the same design heating day.

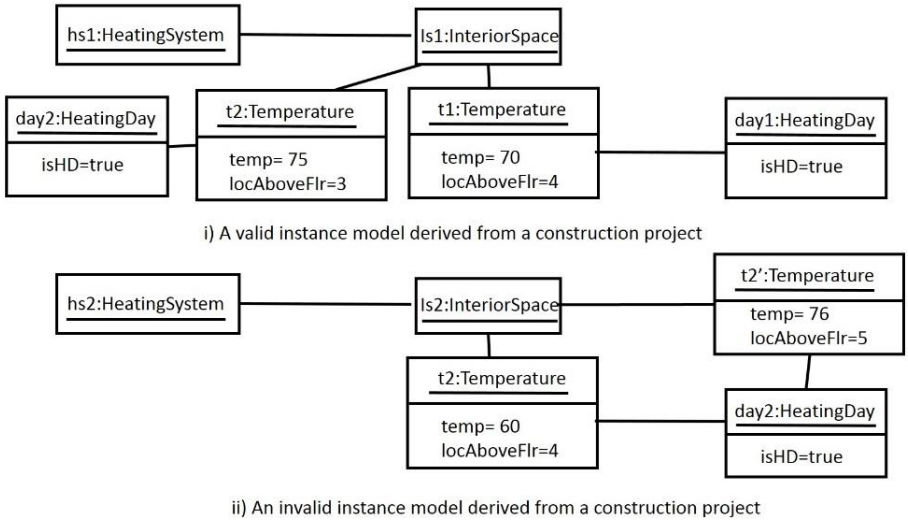

Figure 2 Instance models showing two different construction projects

4 Formalization of International Code By UML Class Diagram

This section gives an example application of a UML diagram to the act of formalizing international building code. To aid construction site managers as they track building construction according to these standards, the mapping of site tools, materials, etc. to the standards as modeled through UML diagrams may be performed. Here we consider the example of Chapter 19 Concrete, section 1904 of [2]. The purpose of chapter 19 is to lay out standards regarding concrete construction. Section 1904 is devoted to regulations on concrete durability requirements. In [2], section 1904, classifications of exposure to chemicals or the environment are defined, along with restrictions on ratios of mixes of cementitious materials to water in concrete and the recommended minimum strengths of concrete mixes according to weathering exposure and location of concrete construction.

A UML model can be useful for relating defined specification rules placed on project material properties and overall design, via a graphical representation. With this representation, instances of the model that represent the resources applied at a construction site can be given values for their properties based on the ratings of their work site counterparts. In the example below, concrete steps are carried out to design a UML model related to section 1904. As a note, the models of each subsection of chapter 19 found able to be modeled as a UML diagram are featured in Appendix A. First, to build a UML model of section 1904, the document section is manually parsed to obtain components and constraints that are translated by the diagram designer. Nouns and pronouns (concepts, enumerations and concept attributes) are translated into interfaces, classes, association classes and class attributes. Nouns containing other identified nouns in their own names (generalizations) are translated and categorized as subclasses. Generalizations are listed in subclass-superclass pairs. Verbs and verb phrases related to noun pairs (relationships) are translated into associations between their classes. Sentences containing words like “shall” and “must” are translated as constraints. Constraints are listed with their related text sections.

To illustrate this process, a sample text is taken from section 1914.3 concrete properties, “Concrete mixtures shall conform to the most restrictive maximum water-cementitious materials ratios and minimum specified concrete compressive strength requirements of ACI 318, Section 4.3, based on the exposure classes assigned in Section 1904.2.” [2]. Here, we can extract concrete, water and cementitious materials as concepts. Given that the water and cementitious materials are related in a mixture ratio that is constrained, we can also consider this ratio as a concept and give it a floating-point number attribute

that represents the ratio of water mixed with cementitious materials. Since this concept is dependent on the concepts of water and cementitious materials, we can define relationships between water-cementitious materials ratio and them. We also see there are references to exposure classes related to the previous subsection of 1904. These classes are given a small number of specified values. Due to this, the classes are considered enumerations and are defined as such in notes used to track concepts and other components to model. There is also a requirement on maximum water-cementitious materials ratios, which can be defined as a constraint that can be later applied to the UML model, for that related concept.

Concepts and attributes are documented and defined in a glossary according to what their purposes are to the section specification. Once all definitions are written, a UML diagram is created to hold the design of the section specifications. For this case, we consider the use of the Papyrus tool in an Eclipse Java development tool. The UML diagram contains classes that represent the key concepts of the section, that are cited. In section 1904, the concept of concrete is considered a class. Any properties of classes that can be enumerated, such as weather exposure severity, are treated as UML enumerations. Relationships between classes where concepts are dependent on other concepts are represented by association line connections from dependent concepts to the concepts on which they depend. Relationships where one or many concepts are a specialization of another concept are represented by generalization lines connected from the specialized concepts to their parent concept. Concept attributes are items applied directly to the classes of the concepts that contain them as properties defined by the specification. The completed UML diagram of this process for section 1904 is displayed in Appendix A, Figure 16.

Once all concepts, attributes and relationships have been represented on the UML diagram, we apply gathered constraints on classes that have properties that are being constrained. When constraints are applied to the diagram, they are written in the syntax of OCL. OCL is required when modeling this section, because it can be used to verify the correctness of a model instance's properties, according to the constraints taken from section 1904. To illustrate the transformation of a textual constraint into an OCL equivalent, we can examine the same text given in the illustration of the process of defining concepts from section 1904. In the initial construction of the UML diagram, we consider constraints first directly defined in section 1904. For constraints applied from related external documents, such as American Concrete Council's ACI 318 as cited in the above text example, the external documents will have those constraints noted and applied to the UML diagram in second phase refinement of

the diagram. With the initial phase of constraint modeling, we can extract an implicit constraint on concrete, requiring its compressive strength to be at least that of its minimum specified compressive strength. This can be translated into the following OCL constraint on concrete and its attributes minimumSpecifiedCompressiveStrength and compressiveStrength:

Context: Concrete
Invariant *MinimumSpecifiedCompressiveStrength:*
self.compressiveStrength >= self.minimumSpecifiedCompressiveStrength;

This translated sentence in OCL syntax states that for a given instance of concrete, its compressive strength must be rated greater than or equal to its minimum specified compressive strength. In the greater scope of modeling a document, this approach is applied to each subsection of each chapter in the document to be modeled, such that one overall model can be constructed from combining features of like-defined concepts between sections and referencing each section's model in the overall model. Formalizing further, additional UML models created from chapter 19 sections of [2] in this initial phase are provided in Appendix A as well.

5 A FRAMEWORK TO SUPPORT COMPLIANCE CHECKING

According to the above example, we propose a new framework to support compliance checking in construction industry. The framework supports the formalization of a set of international building codes using UML class diagram as well as the representation of a construction project via an instance model. First, we employ a UML class diagram to represent an international building code which a construction project should comply with. At the same time, we adopt the Object Constraint Language to denote the constraints in the code which cannot be represented in the class diagram. As mentioned above, to leverage the communication between the designers of a domain model and a construction project, a glossary table is provided to explain how the concepts from an international building code are derived. Second, a construction project is converted to an instance model according to a class diagram. Once these two inputs are provided, conformance checking is carried out and a compliance checking result is returned to designer of a construction project. The conformance checking part in the framework will be built on the UML2 APIs [37], OCL APIs [23], and Eclipse Modeling Framework (EMF) [38] in Eclipse.

A diagram illustrating an overall structure as well as the flow of compliance checking of the framework is shown in Figure 3.

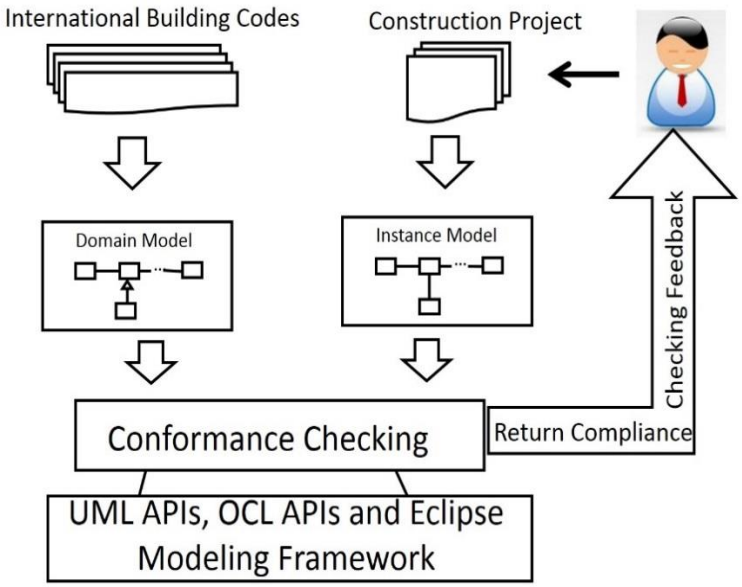

Figure 3 A framework to support compliance checking

In application of this framework, we can take the example from the previous section, after formalizing section 1904 of [2]. We can use two approaches to validate constraints and create instance files, related to the tasks of creating project instances and conducting conformance checking. Figure 4 and Figure 13 highlight the steps taken to use these approaches. In the first approach depicted in Figure 4, we have another tool in Eclipse Java called Eclipse Modeling Framework (EMF), which can be used to create instances of the UML diagram and have them validated. EMF allows the creation of class instance files, that can have values given to their attributes. An instance of the model is represented by the collection of connected class instance files that represent the materials required in a project. A built-in validator in EMF is then used to verify the correctness of the instance model, by comparing its class properties and attributes to the constraints placed upon them according to the UML diagram. The messages are given by the tool according to whether all constraints were passed by the instance or a violation occurred.

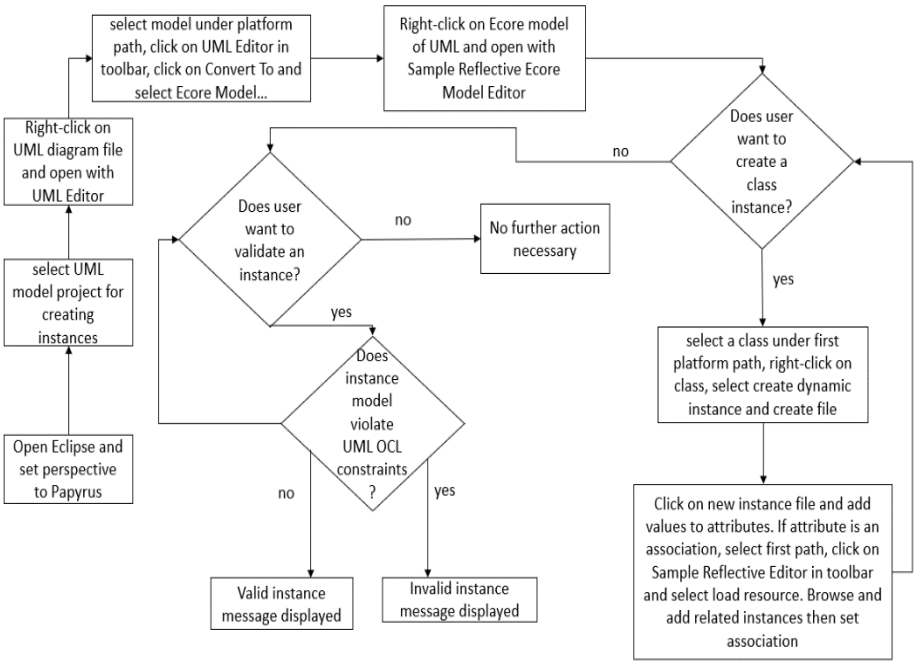

Figure 4. Flowchart depicting steps for using Eclipse Java EMF for creating UML instance diagrams and validating them.

To apply this approach, the user must have Papyrus and Ecore downloaded into their development environments, Eclipse Java for this case. The user must open Eclipse and set their perspective to Papyrus. We do so and search for the UML project created for section 1904 in the file explorer of the environment. If the file doesn't exist the current workspace used by Eclipse, it must be imported. Letting the model be in the directory, we select the UML file, open with a UML Editor, as shown in Figure 5.

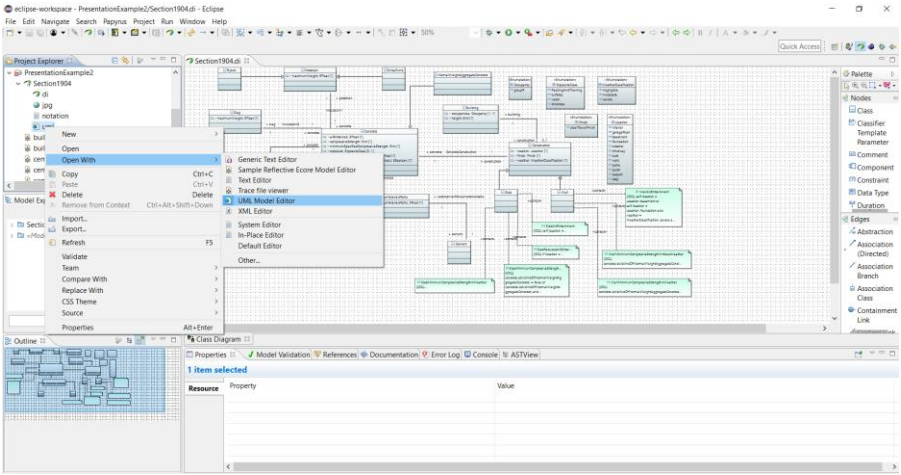

Figure 5. Depiction of how to view a designed document UML diagram in a UML Editor window. The highlighted menu options are those that must be selected in this step.

We then select the model in the editor window, select UML Editor in the toolbar and select Convert To→Ecore Model…, as shown in Figure 6. This allows UML diagram to be saved in a .ecore file format, which can then be used to create dynamic class instances. To do so, we right-click on the converted file, open with Sample Reflective Ecore Model Editor, as shown in Figure 7. We select a class from which an instance will be created, right-click on it and select Create Dynamic Instance, shown in Figure 8. We choose Slab for this case, name the file and save it. We repeat instance creation for each class, for as many instances necessary to represent a project. Once we create the necessary instances, we fill them with data that is representative of the project instance. If an attribute of an instance references other instances, we select the top path in the instance file, go to Sample Reflective Editor→Load Resource..., shown in Figure 9. We browse and enter related instances to each reference and select ok, shown in Figure 10. This is done for each instance, which will then resolve the individual instances into an instance model of the section. To validate these instances, for each instance, we right-click on the instance under the first path listed in its Sample Reflective Editor window, select OCL and then select Validate, as shown in Figure 11. If there is no constraint violation, a success message as shown in Figure 12. Otherwise, a failure message is shown, which explains which constraint is violated. If at least one constraint is violated in the instance model, then the instance model for section 1904 is rejected.

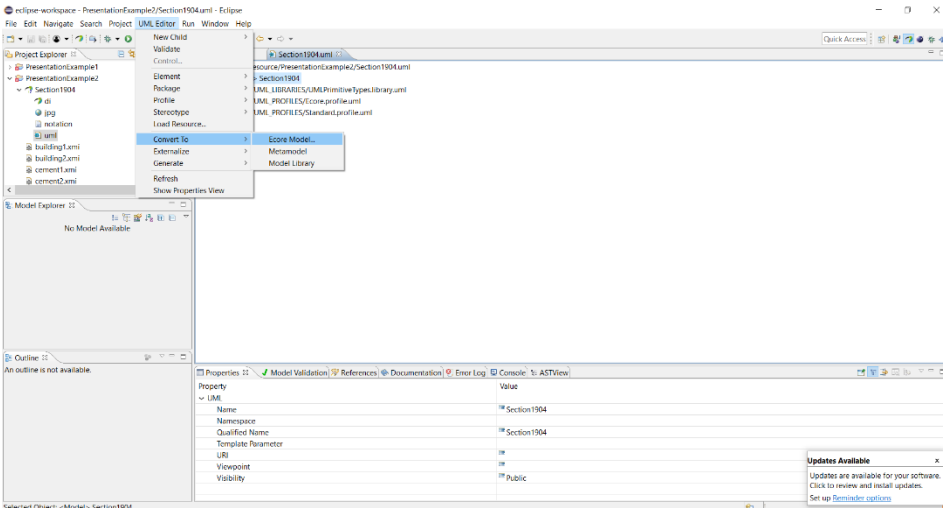

Figure 6. Depiction of how to convert a designed document UML diagram into an Ecore Model. The highlighted menu options are those that must be selected in this step.

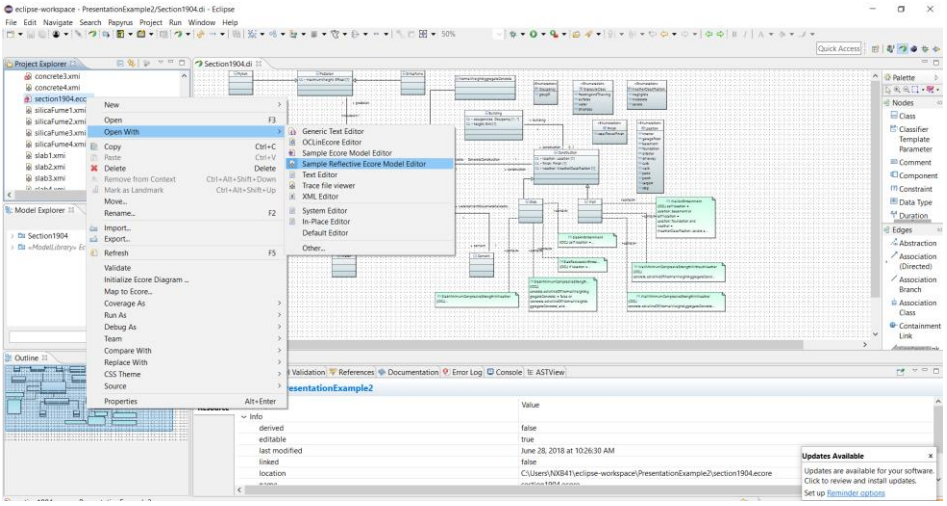

Figure 7. Depiction of step to view converted Ecore model of document UML model for creating project model instances. The highlighted menu options are those that must be selected in this step.

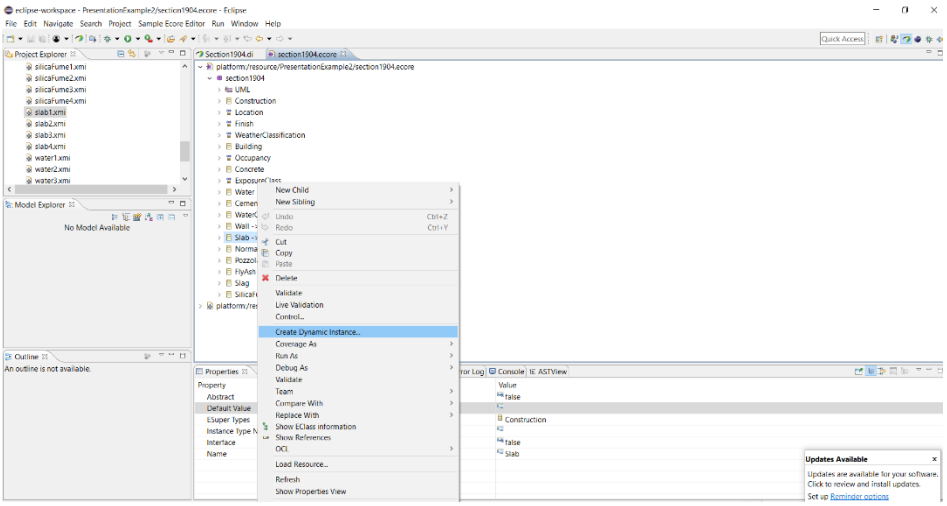

Figure 8. Depiction of step to create a class instance of an Ecore model in Eclipse Java EMF. The highlighted menu options are those that must be selected in this step.

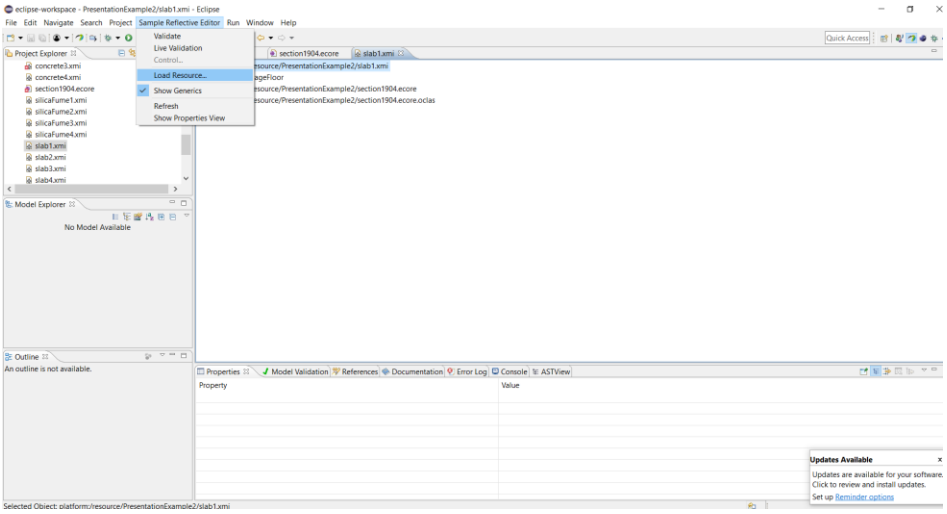

Figure 9. Depiction of step to navigate Sample Reflective Editor menu to load references of class instances into a given model class instance, in Eclipse Java EMF. The highlighted menu options are those that must be selected in this step.

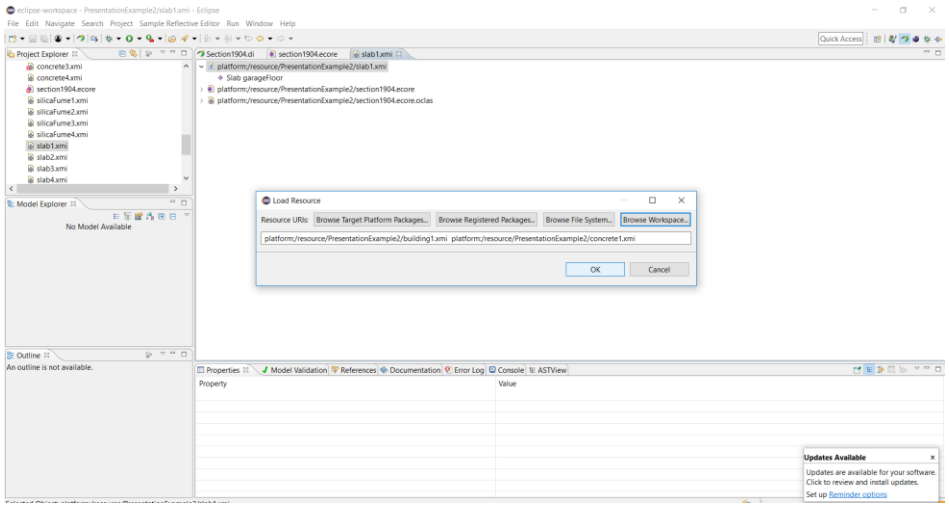

Figure 10. Depiction of step to load references of associated class instances in a model class instance in Eclipse Java EMF. The highlighted menu options are those that must be selected in this step.

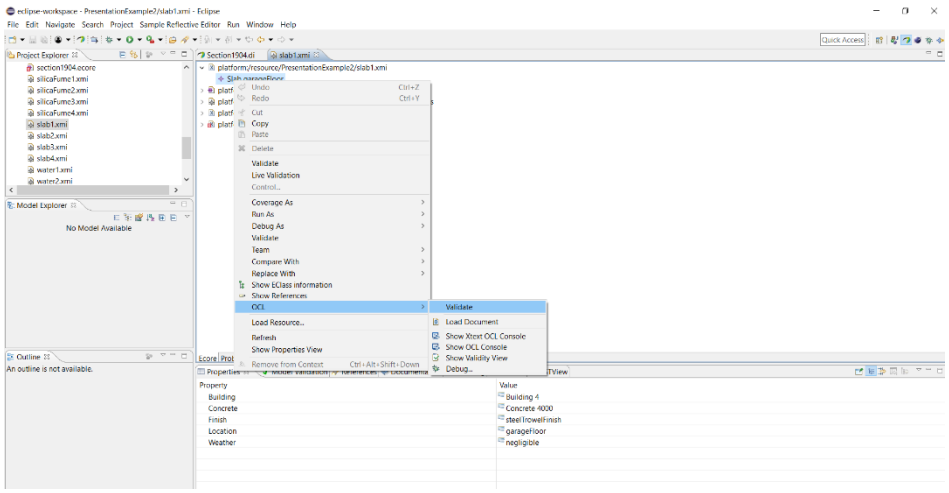

Figure 11. Depiction of step to perform OCL validation on a model instance in Eclipse Java EMF. The highlighted menu options are those that must be selected in this step.

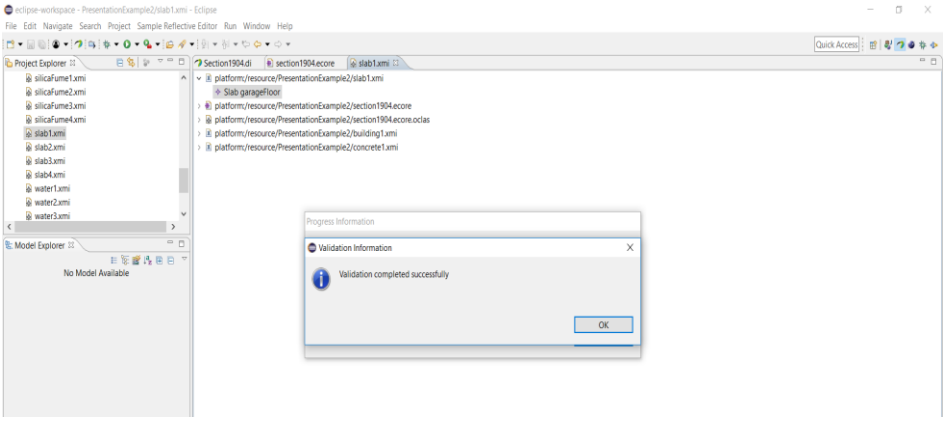

Figure 12. Displayed successful validation of section 1904 class instance.

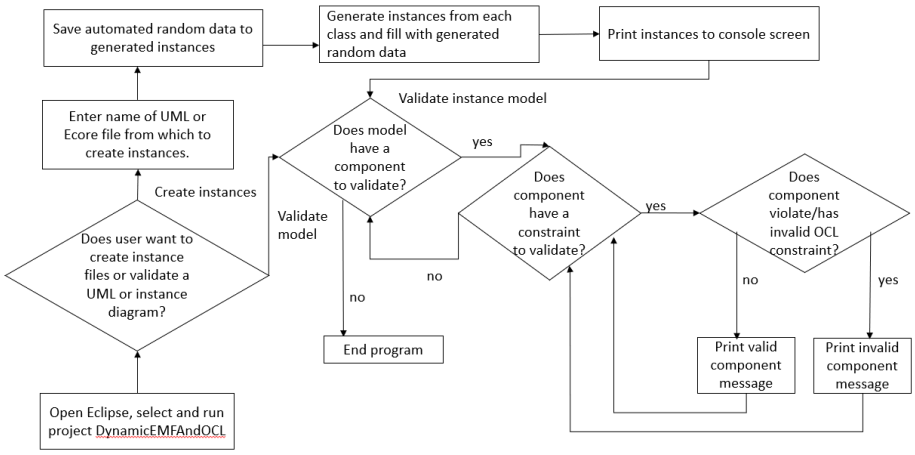

Figure 13. Flowchart depicting steps for using a written Java program for generating instance diagrams and validating them.

In the second approach, a program is written with Java programming language to automate the process of conformance checking and instance model production of for a given UML diagram. The steps performed in this approach are given in Figure 13. When the user runs the program, they have the choice to enter a model file that will have class instances created from it and fed generated random data for constraint testing or can immediately enter an instance model file and its source model to validate. The user must enter a valid model file of type .uml or

a converted serialized version, .ecore in the first case, while instance file types, .xml and .xmi, and related model files in the second case. For this example, we load the .ecore file of section 1904 UML diagram, as shown in Figure 14. The class instances are generated from a dynamic runtime model of the loaded model file from the user and then the random data is given to them. When the instances are generated, not only do they contain the attributes of their source classes, but also annotations and constraints as well. These generated instances are saved to one xml file representing the instance diagram. After generating the instances, each instance's information is printed to the console window for the user to see. An optional step may be taken where additional constraints to be placed on the project at runtime by the user, further constraining instances when such constraints are necessary. Next, the instances generated are taken in context of their source model file and have their constraints validated. For each class instance in the instance model, they are validated by their constraints and messages are printed to console according to whether each constraint has passed validation, as shown in Figure 15. With these features and additional applicable future features, this program can facilitate project model generation and validity testing according to documents upon which they are based.

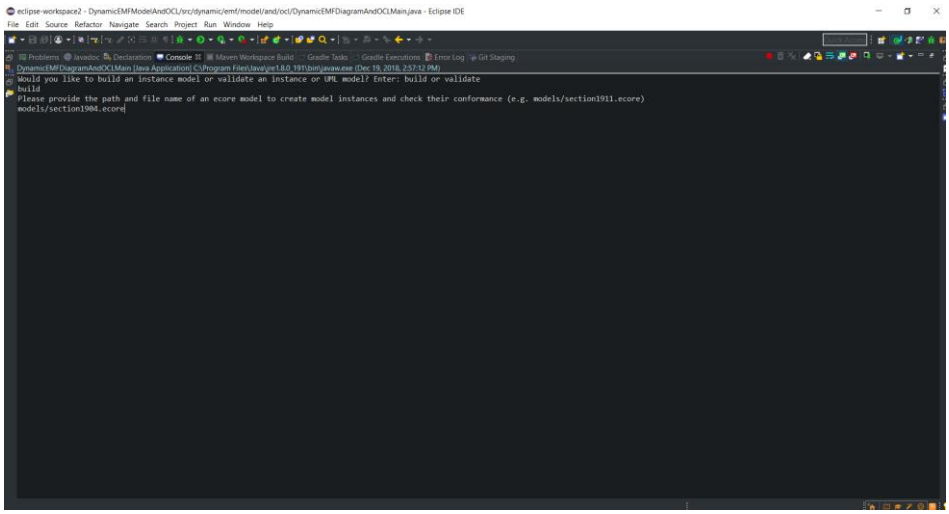

Figure 14. Depiction of step to enter name of the Ecore file representative of the section 1904 UML diagram, from which an instance model will be created by the Java program. The input model file is entered within the console.

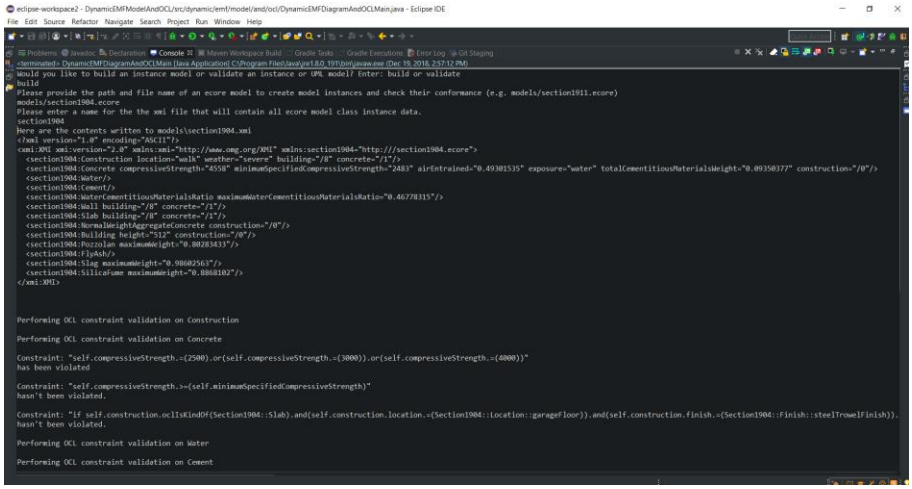

Figure 15. Displayed OCL constraint validation results from automatically generated instance model, from input Ecore file.

6 Formalization of Conformance Checking

In this section, we discuss the formalization of conformance checking for construction industry. As mentioned before, one integral part to conformance checking is a domain model which formally denotes a set of international codes and is given by a UML class diagram. Thus, we first formalize a UML class diagram as follows.

Let set $\mathcal{A}$ be an alphabet and $\mathcal{T}$ *re*presents a set of type names. All string name set can be denoted as set $\mathcal{S} \subseteq \mathcal{A}^+$. Set $\mathcal{T}$ includes all the types in UML such as *Integer* and all UML classes. Set *Classes* $\subseteq \mathcal{S}$ denotes a set of class names. For each $c \in$ *Classes*, we use $t_c \in \mathcal{T}$ to denote the type which is the same as the class name *c*. The attributes of a class $c \in$ *Classes* are defined as a set $Attribute_c$ of signatures a: $t_c \rightarrow t$ where a is the attribute and t_c is the type of the class (name) *c*. A set of associations is given by

i) a finite set of names *Associations* $\subseteq \mathcal{S}$ and

ii) a function *associates*: $\begin{cases} Associations \rightarrow Classes^+ \\ as \rightarrow < c_1,,, c_n > with\ n > 2 \end{cases}$

Let *as* ∈ *Associations* be an association with *associates*(*as*)= $< c_1,,, c_n >$, which denotes an n-ary association *as* connects class $c_1,,, c_n$. Each association has a role name at a class end. Role names are defined by a function

$$roles: \begin{cases} Associations \rightarrow S^+ \\ as \rightarrow < r_1,,, r_n > with\ n > 2 \end{cases}$$

Intuitively, *roles(as)*= $< r_1,,, r_n >$ assigns each class c_i for 1<=i<=n participating in the association *as* a unique role name r_i. Let *as* ∈ *Associations* be an association with *associates*(*as*)= $< c_1,,, c_n >$. The function *multi(as)*=$< M_1,,, M_n >$ assigns each class a non-empty $M_i \subseteq \mathbb{N}$ with $M_i \neq \{0\}$ for all 1⩽ i ⩽ n, where $\mathbb{N}$ denotes the set of all integers.

A generalization hierarchy $\prec$ is a partial order on the set of classes *Classes*. For c_1,c_2 ∈ *Classes* we have $c_1 \prec c_2$, if and only if the class c_1 is a child class of c_2 and c_2 is a parent class of c_1 in UML. In order to collect all parents of a give class, we define the following function:

$$parent: \begin{cases} Classes \rightarrow 2^{Classes} \\ c \rightarrow \{c' | c' \in Classes\ and\ c \prec c'\} \end{cases}$$

Then the full set of attributes of class *c* is given by set $Attribute_c^*$ which contains all inherited attributes along a generation hierarchy as well as those directly defined in *c*.

$$Attribute_c^* = Attribute_c \bigcup_{c' \in parent(c)} Attribute_{c'}$$

Another integral part to conformance checking is an instance model of a domain model. An instance model is used to model a specific construction project which consists of a set of objects as well as a set of links among them. Let I denote a mapping from the domain of all class diagrams, called the UML domain, to the domain of all instance diagrams, called the semantics domain. The semantics domain includes all values, such as *OclVoid*, types defined in OCL, such as *Integer*, and user defined classes. The domain of all instance diagrams includes some values such as $\perp$, an invalid value, ϵ, a null value, $\mathcal{Z}$ (all integer numbers), *true, and false* for OCL predefined values and types. So, I can be defined as *I(Boolean)={true,false}* $\cup$ {ϵ, $\perp$}. Values ϵ and $\perp$ enable the evaluation of an expression which includes undefined and invalid values.

Furthermore, the semantics domain includes a set of instances of a class $c \in$ *Classes* in a class diagram which is denoted by an infinite set *oid(c)*=$\{o_1, o_2, \ldots, o_n\}$. Then the domain of a class c $\in$ $Classes$ is defined as $I_{Classes}(c) = \bigcup\{oid(c') | c' \in Classes\ and\ c' \prec c\}$. Likewise, we can define I on the association. For each association *as* $\in$ Associations with *associates(as)*=$< c_1,,, c_n >$, $I_{Associates}(as) = I_{Classes}(c_1) \times \ldots \times I_{Classes}(c_n)$ denotes all possible links among objects instantiated from classes $c_1,,, c_n$ according to association *as.* Next, an instance model for a class diagram *D* is a structure $\boldsymbol{\sigma}$*(D)=*$\{\sigma_{Classes}, \sigma_{Attributes}, \sigma_{Associations}\}$ which consists of the following three parts:

i. The finite set $\sigma_{Classes}(c) \subseteq oid(c)$ represents all instance of class c in an instance model;
ii. Function $\sigma_{Attributes}(a)$ assigns attribute values to each object: $\sigma_{Attributes}(a)$: $\sigma_{Classes}$ (c) $\rightarrow I(t)$ for each a: $t_c \rightarrow t \in Attribute_c^*$; and
iii. The finite set $\sigma_{Associations}$ *(as)* $\subset I_{Associations}(as)$ denotes all valid links satisfying the multiplicity restriction defined for the association *as*.

A constraint *ocl* in OCL can be evaluated on an instance diagram *ins* derived from a class diagram *d* via the semantics function *I*: Constraints $\times$ Environment $\rightarrow$ {*true,false}*. An environment derived from an instance diagram and denoted as π consists of a state σ and a variable assignment β. The semantics function *I* is applied to the syntax of all OCL structures. For instance, *I* is applied to an if statement in OCL as follows and semantics for the rest OCL structure can be found [23]:

$$I(\textit{if } e_1 \textit{ then } e_2 \textit{ else } e_3 \textit{ endif}, \pi) = \begin{cases} I(e_2, \pi), \text{if } I(e_1, \pi) = true; \\ I(e_3, \pi), \text{if } I(e_1, \pi) = false; \\ \perp,\ otherwise \end{cases}$$

For a specific constraint *l*, an environment π derived from an instance diagram *ins*, which is an instance model of a domain model *d*, and a mapping function *I*, if $I(l, \pi) = true$, then we say the environment π derived from instance diagram *ins* satisfies the domain model *d* based on the constraint *l*, denoted as *ins* $\vDash_{(I,\pi)}$ *l*. If an instance diagram *ins* satisfies a domain

model d based on all the constraints in d, which is $\forall\ l \in d.Constraints$, such that $ins \vDash_{(I,\pi)} l$, then we say the instance diagram conforms to the domain model denoted as $ins \vDash_{(I,\pi)} d$. The conformance relationship $\vDash_{(I,\pi)}$ ensures that an instance diagram is a valid instance of the domain model. Assume a domain model d is derived from a code C and some instance diagram ins is based on a specific project denoted as S. If we have $ins \vDash_{(I,\pi)} d$, then we say the project S comply with the code C.

7 CONCLUSION AND FUTURE WORK

In this report, we introduce how to support the compliance checking in construction industry can be achieved the conformance checking used in safety critical domains by means of MDE. One of the most important reasons we choose conformance checking is that several frameworks are available to carry out conformance checking in the MDE community. We employ the application of UML2 APIs, and OCL APIs in the Eclipse Modeling Framework (EMF) to support the conformance checking as illustrated earlier. EMF has been extensively applied as a framework to support MDA. Various tools and plugins such as IBM Rational Architect [39] and Eclipse Papyrus Plugin have been developed to support the EMF framework. The UML2 APIs are a set of APIs to process a UML model such as a class diagram while OCL APIs provide a set of APIs to aid OCL evaluation in a UML model. The UML2 and OCL APIs provide us with a programmatic method to automatically perform conformance checking. These frameworks have been proved efficient in the execution of conformance checking and we wish the efficiency of these frameworks can leverage the capability of compliance checking in construction industry and shorten design time for a construction project. As a result, our framework is proved to be efficient when applying to several case studies. Furthermore, the formal definition of compliance checking provides the theoretic foundation of application of these frameworks in construction industry, ensuring that conformance checking carried out by the frameworks can be further proved to be correct in the support of compliance checking.

As the future work, we will plan to develop new features in the framework to leverage the capability of developers to estimate project cost. In fact, cost estimation is a daunting task requiring developers' experience and most of cost estimation is done manually. But with a domain model generated by a UML class diagram according to an international building code, it can provide us some benefit to build features such as supporting project cost estimation as future work.

Acknowledgement This PI appreciates the generous support by the Georgeau Construction Research Institute at Western Michigan University.

References

[1] Z. Zhou, Y. M. and L. Shen, "Overview and Analysis of Ontology Studies Supporting Development of the Construction Industry," *Journal of Computing in Civil Engineering,* vol. 30, no. 6, 2016.

[2] International Code Council, "2009 international codes," 2009. [Online]. Available: https://archive.org/stream/gov.law.icc.ibc.2009#page/n177/mode/2up. [Accessed 23 7 2017].

[3] Model national energy code of Canada for houses 19971st Ed, *Table A 3.3.1.1.—Prescriptive requirements—building assemblies,* Ottawa: National Research Council of Canada, 1997.

[4] P. Graydon, I. Habli, R. Hawkins, T. Kelly and K. J., "Arguing Conformance," *IEEE Software,* no. May/June, pp. 50-57, 2012.

[5] T. Kelly, Arguing Safety - A Systematic Approach to Manage Safety Cases, Doctoral Dissertation, Dept. of Compuer Science, Univ. of York ed., 1998.

[6] M. Woehrle, K. Lampka and L. Thiele, "Conformance testing for cyber-physical systems," *ACM Trans. Embed. Comput. Syst.,* vol. 11, no. 4, pp. 1-23, January 2013.

[7] A. A. Hauge and K. Stølen, "A pattern-based method for safe control systems exemplified within nuclear power production," in *Computer Safety, Reliability, and Security, LNCS Volume 7612*, 2012.

[8] M. Pajic, A. Jiang, I. Lee, O. Sokolsky and R. Mangharam, "Safety-critical medical device development using the UPP2SF model translation tool," *ACM Trans. Embed. Comput. Syst.,* vol. 13, no. 4, July 2014.

[9] D. J. Rinehart, J. C. Knight and J. Rowanhill, "Current Practices in Constructing and Evaluating Assurance Cases With Applications to Aviation," NASA/CR–2015-218678, Jan 2016.

[10] E. Denney and G. Pai, "A lightweight methodology for safety case assembly," in *Computer Safety, Reliability, and Security*, Springer, 2012, pp. 1-12.

[11] R. France and B. Rumpe, "Model-Driven Development of Complex Software: A Research Roadmap," in *Proceedings of Future of Software Engineering*, 2007.

[12] J. Dimyadi and R. Amor, "Automated Building Code Compliance Checking–Where is it at.," in *CIB WBC*, 2013.

[13] T. Kasim, "BIM-based smart compliance checking to enhance environmental sustainability," in *Diss. Cardiff University*, 2015.

[14] R. Lipman, M. Palmer and S. Palacios, "Assessment of conformance and interoperability testing methods used for construction industry product models," *Automation in Construction,* vol. 20, no. 4, pp. 418-428, 2011.

[15] N. O. Nawari, "Automating Codes Conformance in Structural Domain," in *International Workshop on Computing in Civil Engineering*, 2011.

[16] W. Solihin and C. Eastman, "Classification of rules for automated BIM rule checking development," *Automation in Construction,* vol. 53, pp. 69-82, 2015.

[17] J.-K. Song, G.-H. Cho and K.-B. Ju, "A study on the rule development for BIM-based automatic checking in a duct system," *Korean Journal of Air-Conditioning and Refrigeration Engineering,* vol. 25, no. 13, pp. 631-639, 2013.

[18] X. Tan, A. Hammad and P. Fazio, "Automated code compliance checking for building envelope design," *J. Comput. Civ. Eng.,,* vol. 24, no. 2, pp. 203-211, 2 2010.

[19] J. K. Yeoh, J. H. Wong and L. Peng, "Integrating Crane Information Models in BIM for Checking the Compliance of Lifting Plan Requirements," in *International Symposium on Automation and Robotics in Construction*, 2016.

[20] A. Yurchyshyna and A. Zarli, "An ontology-based approach for formalisation and semantic organisation of conformance requirements in construction," *Automation in Construction,* vol. 18, no. 8, pp. 1084-1098, 2009.

[21] A. Yurchyshyna1, C. F. Zucker, N. L. Thanh, C. Lima and A. Zarli, "Towards an ontology-based approach for conformance checking modeling in construction," in *PROC. OF «24TH W78 CONFERENCE–BRINGING ITC KNOWLEDGE TO WORK*, 2007.

[22] The Object Management Group (OMG), "Unified Modeling Language (OMG UML), Superstructure Specification (Version 2.4.1)," 2011.

[23] The Object Management Group, "The Object Constraint Language (OCL) version 2.4," 2014.

[24] P. Pauwels, D. Van Deursen, R. Verstraeten, J. De Roo, R. De Meyer, R. Van de Walle and J. Van Campenhout, "A semantic rule checking environment for building performance checking," *Automation in Construction,* vol. 20, no. 5, pp. 506-518, 2011.

[25] C. M. Eastman, C. Eastman, P. Teicholz and R. Sacks, BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors, John Wiley & Sons, 2011.

[26] buildingSMART, "IFC introduction," [Online]. Available: http://buildingsmart.org/ifc/.

[27] F. S. Tien and Q. Zhong, "Construction and real estate NETwork (CORENET)," *Facilities,* vol. 19, no. 11/12, pp. 419-428, 2001.

[28] L. Khemlani, "CORENET e-PlanCheck: Singapore's automated code checking system," AECbytes "Building the Future" Article, 26 10 2005. [Online]. Available: www.novacitynets.com/pdf/aecbytes_20052610.pdf. [Accessed 23 7 2017].

[29] ICC (International Code Council), "International code council AEC3," 26 10 2013. [Online]. Available: http://www.aec3.com/en/5/5_013_ICC.htm. [Accessed 23 7 2017].

[30] J. Zhang and N. El-Gohary, "Automated Information Transformation for Automated Regulatory Compliance Checking in Construction," *J. Comput. Civ. Eng.,* vol. 29, no. 4, 2015.

[31] M. Richters and M. Gogolla, "Validating UML models and OCL constraints," in *Proceedings of Unified Modeling Languages (UML'00)*, York, England, 2000.

[32] R. F. Paige, P. J. Brooke and J. S. Ostroff, "Metamodel-based model conformance and multiview consistency checking," *ACM Transactions on Software Engineering and Methodology (TOSEM),* vol. 16, no. 3, 2007.

[33] The Object Management Group, "Unified Modeling Language (OMG UML), Superstructure Specification (Version 2.4.1)," 2012.

[34] H. M. Chavez, W. Shen, R. B. France, B. A. Mechling and G. Li, "An Approach to Checking Consistency between UML Class Model and Its Java Implementation," *IEEE Transactions on Software Engineering,* vol. 42, no. 4, pp. 322-344, 2016.

[35] E. Fundation. [Online]. Available: http://www.eclipse.org/modeling/mdt/.

[36] D. Steinberg, F. Budinsky, E. Merks and M. Paternostro, EMF: eclipse modeling framework, Pearson Education, 2008.

[37] D. Leroux, M. Nally and K. Hussey, "Rational Software Architect: A tool for domain-specific modeling," *IBM Systems Journal,* vol. 45, no. 3, pp. 555-568, 2006.

[38] A. Ayoub, B. Kim, I. Lee and O. Sokolsky, "A safety case pattern for model-based development approach," in *NASA Formal Methods*, Springer, 2012, pp. 141-146.

[39] W. Hunt, "Modeling, Verification of Cyber-Physical Systems," in *National Workshop on High-Confidence Automotive Cyber-Physical Systems*, 2008.

[40] A. Joshi, M. Heimdahl, S. Miller and M. Whalen, "Model-Based Safety Analysis Final Report," *Contractor report Cecilia Haskins, Nasa Langley Research Center,* 2006.

[41] E. Denney and G. Pai, "Automating the Assembly of Aviation of Safety Cases," *IEEE Transaction on Reliability,* vol. 63, no. 4, Dec 2014.

8 Appendix

A. Figures ICC international code 2009, sample formalized UML diagrams of sections able to be modeled as UML diagrams, under first document parsing phase

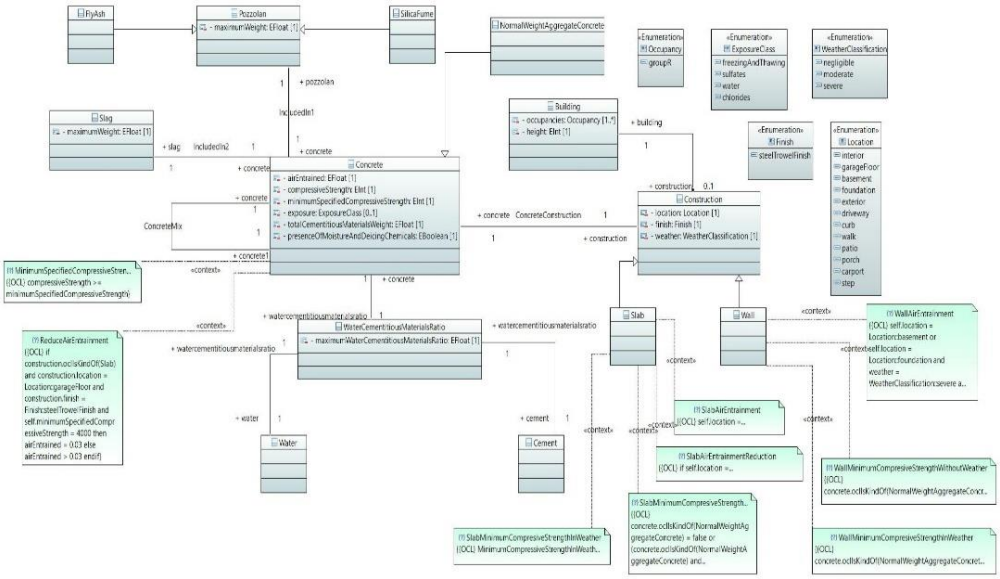

Figure 16. UML diagram of ICC international code 2009, chapter 19 section 1904.

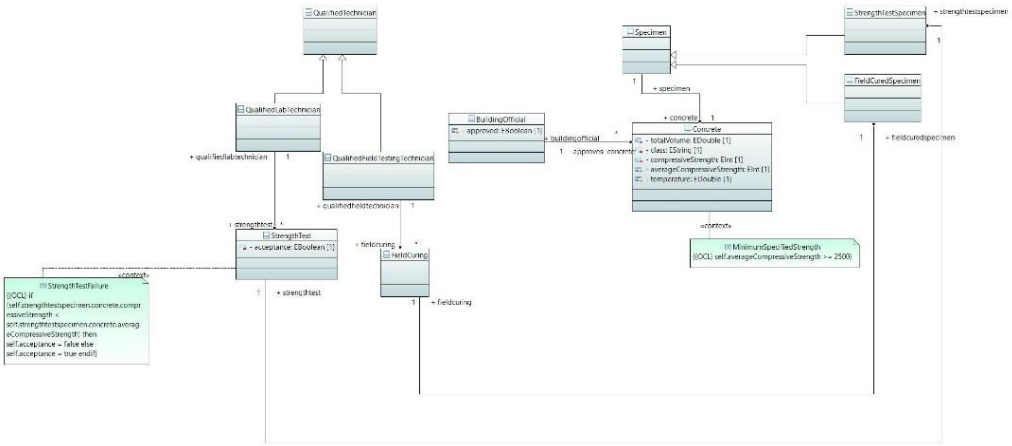

Figure 17. UML diagram of ICC international code 2009, chapter 19 section 1905.

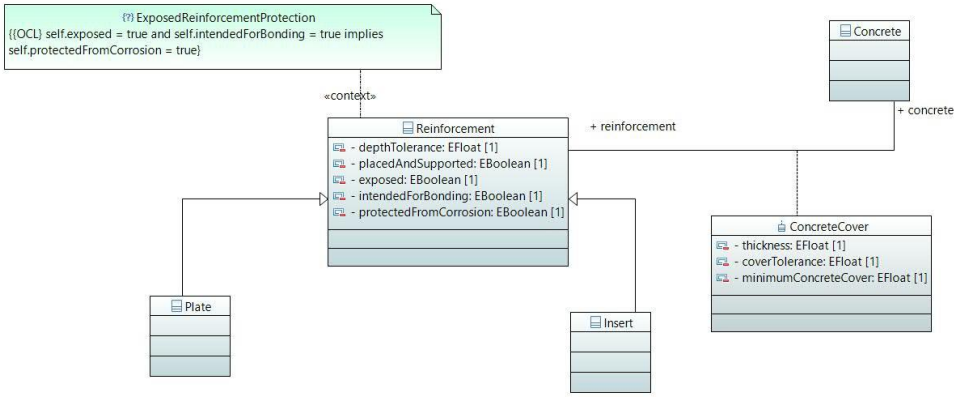

Figure 18. UML diagram of ICC international code 2009, chapter 19 section 1907.

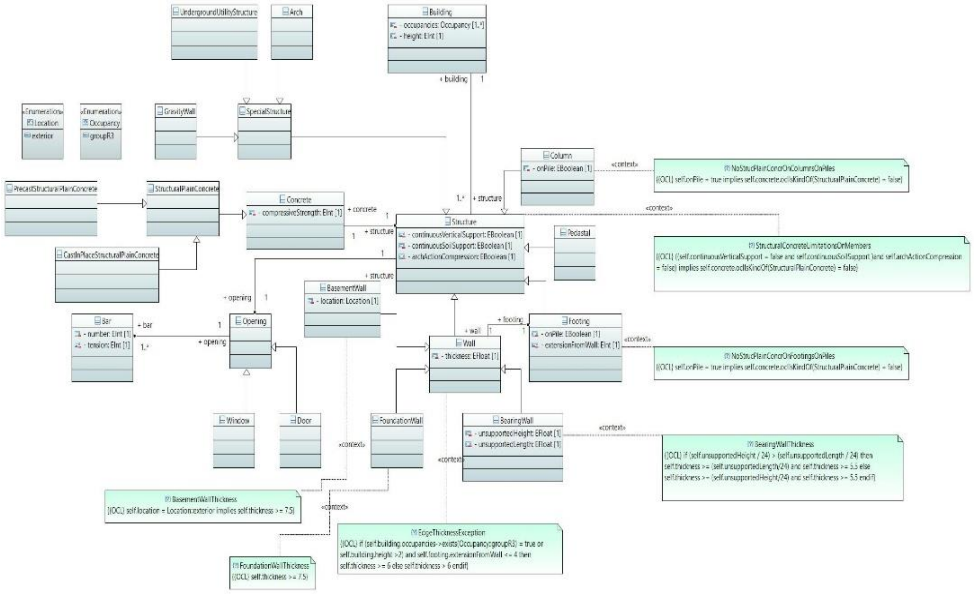

Figure 19. UML diagram of ICC international code 2009, chapter 19 section 1909.

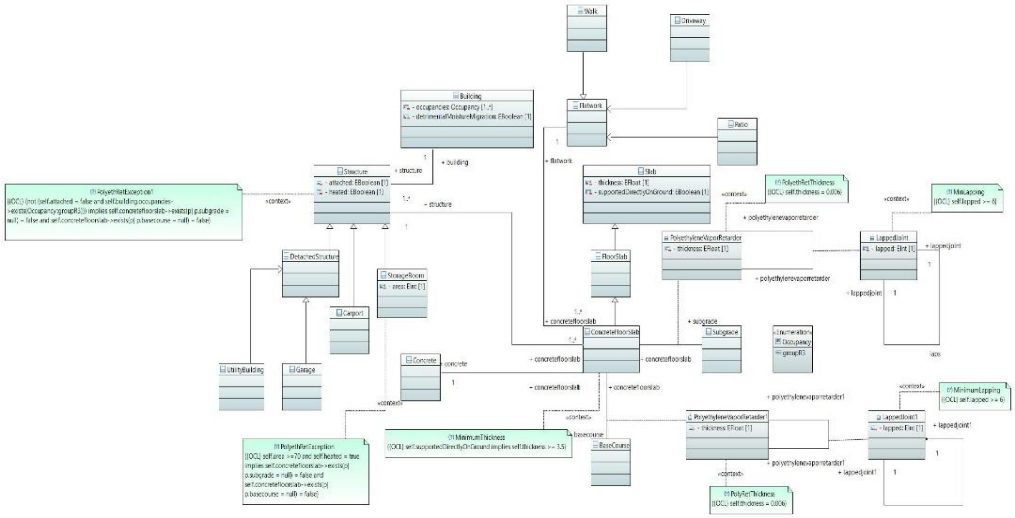

Figure 20. UML diagram of ICC international code 2009, chapter 19 section 1910.

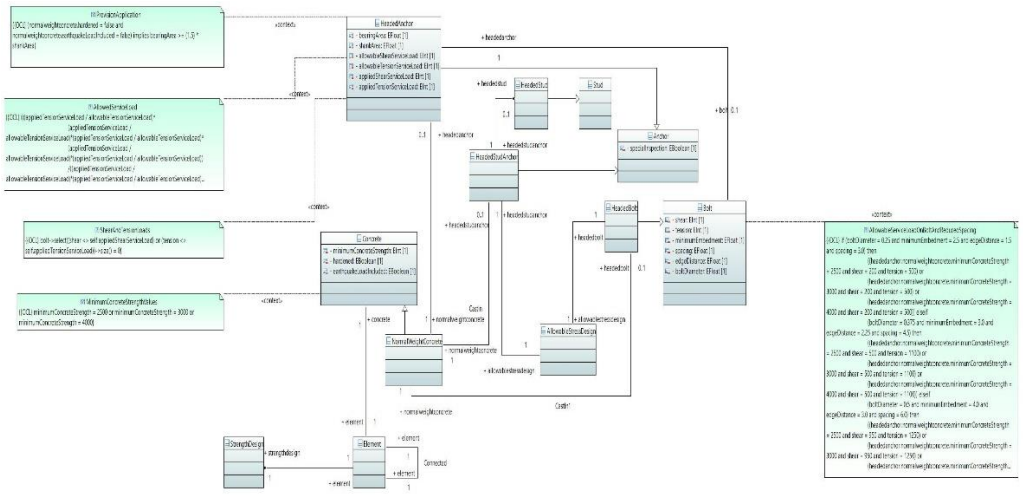

Figure 21. UML diagram of ICC international code 2009, chapter 19 section 1911.

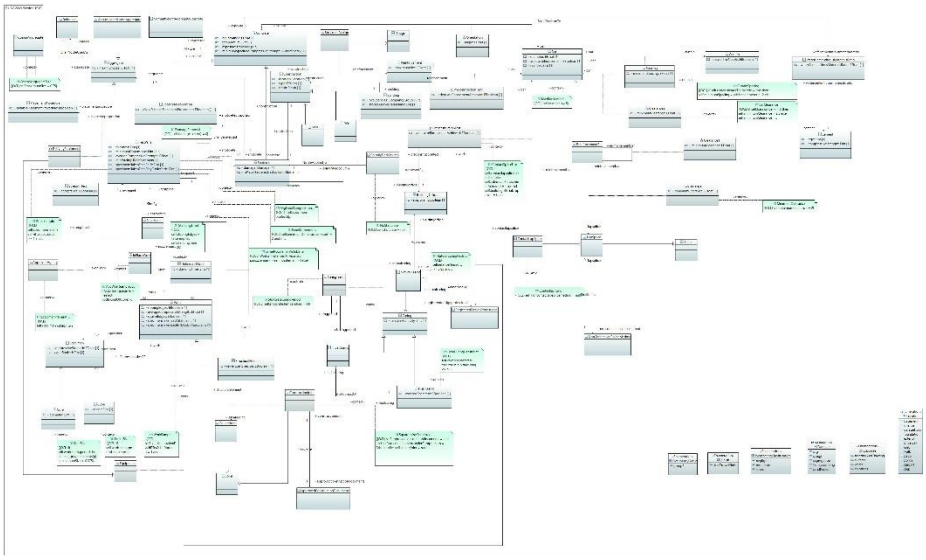

Figure 22. UML diagram of ICC international code 2009, chapter 19 section 1913.

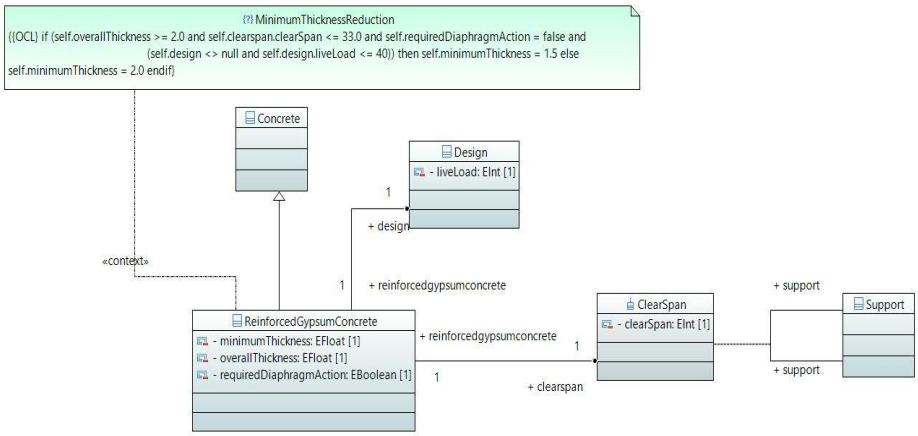

Figure 23. UML diagram of ICC international code 2009, chapter 19 section 1914.

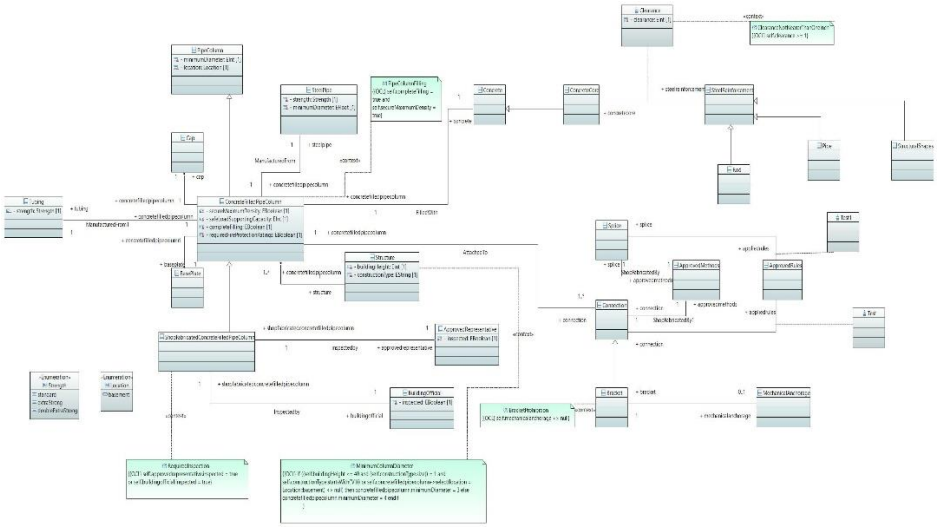

Figure 24. UML diagram of ICC international code 2009, chapter 19 section 1915.