A Knowledge Representation Approach to Capturing BIM Based Rule Checking Requirements Using Conceptual Graph

Wawan Solihin, wsolihin3@gatech.edu
Georgia Institute of Technology, USA

Charles Eastman, charles.eastman@coa.gatech.edu
Georgia Institute of Technology, USA

Abstract
This paper addresses the issue of requirement specifications of BIM based rules, including the more complex building codes, for the purpose of automated rule checking implementation. We propose a standardized way to capture these requirements using a knowledge representation approach. Detailed analysis of the structure of the rules points to the conceptual graph (CG) as a suitable method for this purpose due to its expressiveness, which allows unambiguous description of the requirements that can be understood by all the participants in the implementation efforts. A conceptual graph is a representation that conforms to the first order logic that makes it suitable for the job. The approach bridges the domain knowledge that resides mostly with the rule experts and the technical knowledge possessed by the implementers that are mostly computer scientists and engineers. This paper demonstrates the potential of the approach using selected rules from various building codes. Using the conceptual graph, rules can be broken into their atomic rules, making it necessary to remove or clarify ambiguities that often plagued building codes. It also provides a standardized way to capture and document the model data requirements and the high level checking logic as their functional requirements. With this, a layer that often separates the rule experts and the implementers can be eliminated, resulting in clarity and immediate usefulness for the implementation.

Keywords: automated rule checking, conceptual graph, knowledge representation

1 Introduction

One of the important steps in the rule checking process is rule interpretation (Eastman, Lee, Jeong, & Lee, 2009). Experience in implementing CORENET ePlanCheck in Singapore shows that the interpretation step can take as much as 30% of the total time to implement a rule. Complex rules typically found in building codes are a combination of several aspects that contribute to their complexity, i.e. the language structure, the domain knowledge embedded in the rules that includes hidden assumptions, and their logic structure. Added to these technical aspects of the rules is the human aspect of the interpretation. A study by Fiatech confirmed that when the human interpretation is involved, inconsistencies are expected. Different officers tend to interpret the rules differently, often colored by their experience and locality (Fiatech, 2012). Some rules in CORENET ePlanCheck implementation went through multiple iterations and revisions because of the same reason when multiple reviewers are involved. It does not help that in the typical development, software developers are not the ones directly involved in the rule interpretation.

Rule checking does not stop at the development effort. It also involves the second workflow after the rules have been implemented. Different sets of users are now involved in the process that involves data exchange, which needs to be consistent with the rule implementation. This requires a different form of the current knowledge transfer to the modelers. This aspect of AEC specific workflow has been well defined and a standard process has been proposed and used within the BuildingSMART community using IDM (Information Delivery Manual) and MVD (Model View Definition). The challenge in rule checking implementation is to integrate the knowledge transfer during the interpretation, implementation process and the actual usage of the rule. This should be done with

minimum information loss (Figure 1). Good communication and documentation becomes critical in this process. Unfortunately without a systematic methodology it has proved to be a significant challenge. Experience in implementing an automatic rule checking system in CORENET ePlanCheck shows that the voluminous documentations did not reduce the issue of knowledge loss along the process, and in some cases it added to the problem.

Recognizing the importance of addressing the knowledge gap and reducing the information loss, this study proposes the use of Conceptual Graphs (CG) to capture the rule requirements in form of semantic knowledge representation. The CG is designed to capture knowledge of the rule into its

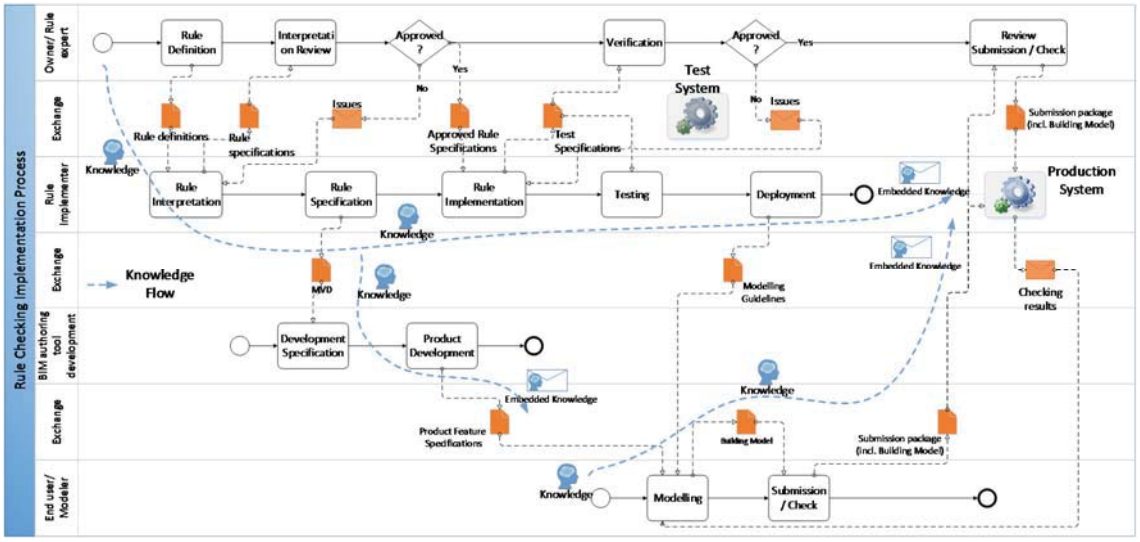

Figure 1 - Typical Rule Implementation Process and the Knowledge Flow

basic logic structure and the data involved. The aim is to enable effective communication of all users and to identify exact data, relationship between data and any required function to encapsulate the complex algorithm involved in solving a rule. One important feature of using the CG for this purpose is a built in mechanism to capture the requirement for derived data and relationships that are vital to rule checking implementation.

2 Semantic Knowledge Representation of Building Rules

In recent years, several approaches have been proposed to capture building related rules into certain forms of knowledge representations. For example, El-Gohary et al proposed the use of Natural Language Processing (NLP) to "interpret" building codes (Salama & El-Gohary, 2013; Zhang & El-Gohary, 2012). Hjelseth and Beach used a methodology called RASE (Requirement, Applicability, Selection, Exception) to tag the building codes into four categories of ideas to drive the computer implementation of automated rule checking (Beach, Kasim, Li, Nisbet, & Rezgui, 2013; Hjelseth & Nisbet, 2011). Both approaches focus on the structure of the rules and have not sufficiently addressed the semantic-interpretation issue of the rules and whether the building representation is sufficient to support the richness of language expression written in the rules. In this paper, the rules are viewed as knowledge assets complete with their association with the building representation that they mean to check. In this study, we only focus on the widely used open standard IFC (Industry Foundation Classes) by BuildingSMART, which is now an ISO standard (ISO, 2013). IFC is supported by all major BIM authoring tools.

The first task needed is to choose the suitable representation to capture the semantic knowledge of the rules. The most suitable method appears to be from the field of Knowledge Base (KB) that is a branch of Artificial Intelligence (AI). But even within KB, there are many different approaches including First Order Logic (FOL), Description Logic (DL), and Conceptual Graph (CG). CG was originally proposed by Sowa in 1976 (John F. Sowa, 1976) and further developed in 1984 (J. F. Sowa, 1984). The Conceptual Graph (CG) offers intuitive, easy to read and suitable to capture semantic

knowledge representation of the rules (Chein & Mugnier, 2008). CG has its semantic foundation in FOL and the basic form of CG can be mapped 1-to-1 directly to FOL.

Thus the goals in using the CG are:

- As an expressive tool to capture knowledge of rules in terms of their requirements for automation that are easily understood by the rule experts, who are typically not familiar with computer programming.
- Ability to capture data requirements of the building objects and their relationships or interactions with other building objects, including constraints.
- Direct mapping of the CG concepts into IFC entities, derived entities and extension functions. The mapping is important for both defining MVDs and for software development efforts.
- Ability to breakdown complex rules into their atomic rules in a systematic and standardized way.

2.1 Conceptual Graph as a Knowledge Representation of Building Rules

With its history in semantic networks, CG defines rectangles to represent concept nodes, ovals to represent conceptual relations, and diamond shapes to represent functions (an extension to CG). The nodes are connected by arcs with an arrowhead pointing to the ellipse. This marks the node as the first argument of the relation. The node with the arrow pointing away from the ellipse marks the last argument (Figure 2).

A Concept node typically represents an object, but it can also be extended to represent a whole atomic rule. This is achieved using a concept called coreference that links two different nodes. Coreferent nodes should be able to be merged into just a single node. They are represented with a dashed line (Figure 3).

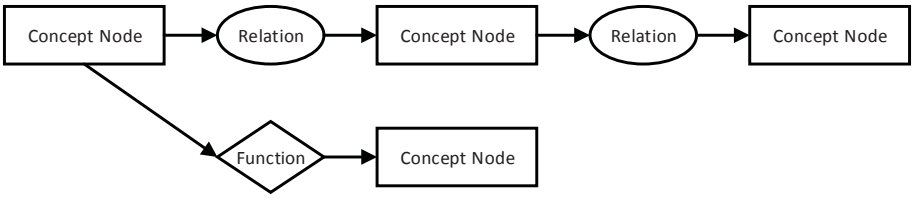

Figure 2 - Basic Definitions of Conceptual Graph

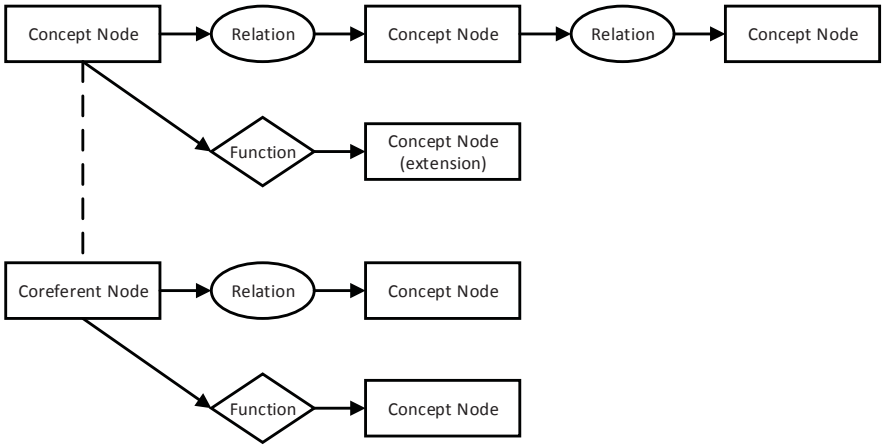

Figure 3 - Coreferent Node

We will use the CG to represent the semantic knowledge of the rules. A rule will be represented by a series of connected graphs as above. An additional notation used for the graph is for Constraint. Constraint is described in the similar manner as a rule, except it uses a different style, i.e. shaded. Constraint can be applied to any node in the graph, which indicates on which concept the constraint applies (Figure 4).

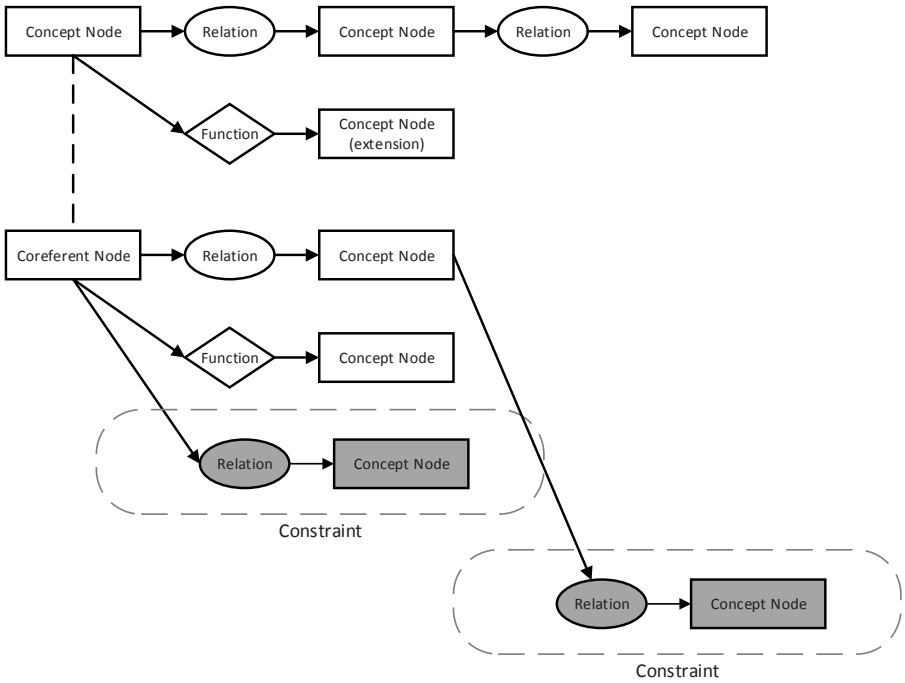

Figure 4 - CG with Constraints

2.2 Notational Extension

Due to the complexity of building rules, there is a necessity to add a few notational extensions into the graph for improved readability of the CG. Figure 5 shows such extensions:

- Nodes with dashed line borders represent a derived concept or concept that will require additional support during the implementation using computer algorithm. The requirement for a derived concept has been identified to address more complex classes of rules (Solihin & Eastman, 2015). The derived concept generally requires support from a computer algorithm that is applied to the basic building model. FORNAX™ , which was developed for CORENET ePlanCheck, used the same concept (Solihin, Shaikh, Rong, & Poh, 2004), and very recently a rule checking effort in the UK took a similar approach (Malsane, Matthews, Lockley, Love, & Greenwood, 2015).
- Specific labels: OR and (NOT) ¬ to represent logical disjunction (∨) and negation (¬). By default, if there is more than one link connecting a node, the operation is a logical conjunction (∧). A thin line box surrounding the negation block is part of the standard CG.
- Thin dashed line with rounded rectangle represents a special block, which could be used to show the constraint block(s) or the exception rule.
- A dotted line connecting a concept node to a double border box indicates a dependency for the concept that is specified in another rule. This is important information that connects a certain concept with a specific rule.

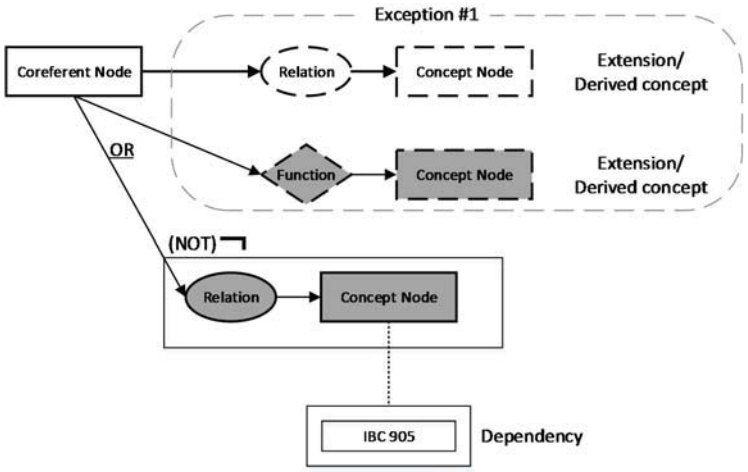

Figure 5 - Notational Extensions for CG

3 Translation of Rules into the Conceptual Graph

Translation of rules into CG is not always straightforward sequential mapping because of the way the rules are written. The following steps are used in performing the translation, which is generally part of the interpretation step:

1) Identify the main concept that the rule is applicable to. This step should identify a concept without any qualification as the minimum. Most of the time it will have some kind of filter or specifications on what type of specific concept it is applied to. For example:
   - "*Spaces must be from agreed list*" (Solibri) will identify the concept applicable to this rule as a "Space" without any further specification. In First Order Logic (FOL) this rule is represented with $\forall x(Space(x))$
   - "*The underground building shall be equipped throughout with a standpipe system …*" (IBC 2009 405.10) will identify that the concept is applicable to a building, specifically an underground building. In FOL it is represented with $\exists x(Building(x), Property(x, type: UNDERGROUND))$

   In some cases the concept is not very explicit. In this case a level of abduction is needed. For example:
   - "*Model should have components*" (Solibri). In this rule, model refers to the building model as a whole because the rule requires any type of entity that can be specified. Since the rule comes from Solibri Model Checker, which provides a template, this rule can only be operational once user assigns what entity type(s) the rule should apply to.
2) Identify atomic sub-rule(s). A rule, especially in building codes, often specifies more than one sub-rule that is relatively independent, except that they are operating on the same entity. In this case, the sub-rules will be defined as separate rules under the same heading. For example:
   - "*Doors, when fully opened, and handrails shall not reduce the required means of egress width by more than 7 inches (178 mm). Doors in any position shall not reduce the required width by more than one-half. Other nonstructural projections such as trim and similar decorative features shall be permitted to project into the required width a maximum of 11/2 inches (38 mm) on each side.*" (ICC, 2009).

     This rule is applicable to doors that open to the egress path. There are two sub-rules, one dealing with reduce of egress width due to the door opening, and the second one deals with the space occupied by the door at the fully open position.
3) Identify atomic constraint(s). The general structure of a building rule is a specification of the main building entity with its specifications, followed by one or many constraints. The constraints are not restricted to the main entity, but can be applicable to other entities

that are related to the main entity or even to an entity within the constraint, i.e. constraint within constraint. This increases the complexity of the rule. For example:

- "*All patient rooms must be visible from the nurse station*"
  This rule has a constraint on the main entity, the nurse station. The constraint specifies that the patient rooms must be within the line of sight from the nurse station.

4) Define the appropriate CG of the rule by connecting the concepts using relations and functions until the consistent semantic is clearly self-describing.

3.1 Applying the Semantic Representation CG to Building Rules

In this section, several rules are selected to represent the range of rules that are applicable to buildings. Several rules are selected from different classifications of rules as defined in (Solihin & Eastman, 2015), mainly for class-1 to class-3. Class-4 does not create new semantics or complexity, but it introduces requirements in terms of algorithmic solutions to present a "proof of solution", and therefore it does require additional semantics compared to the other three classes.

3.1.1 Class-1 rule (rules that require a single or small number of explicit data) example:

"Spaces must be from agreed list" (*Solibri*)

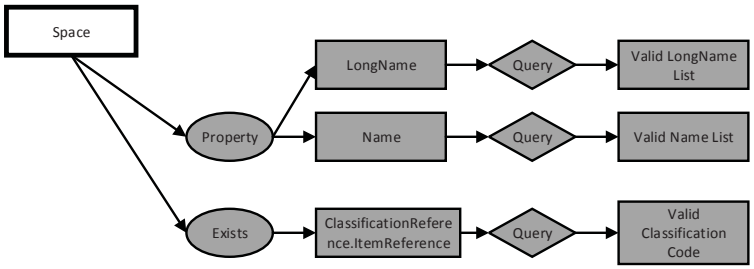

Figure 6 - Class-1 rule example

In FOL, the above rule (Figure 6) can be expressed as:

$$\forall a\big(Space(a)\big)$$
$$\wedge\, \exists a((Property(a, Name) \wedge Query(Name, ValidNameList)) \wedge (Property(a, LongName) \wedge Query(LongName, ValidLongNameList))$$
$$\wedge\, \exists a((ClassificationReference(a, ClassificationReference) \wedge Query(ClassificationReference.ItemReference, ValidClassificationCode)$$

This rule checks the existence of properties Name and LongName, and checks IfcClassificationReference.ItemReference using a simple query function that checks for the existence of a property or a classification. Name and LongName properties are existing properties in IFC schema and IfcClassificationReference is the IFC entity that is used to assign a classification item to an entity, which is expected in this case for an IfcSpace.

3.1.2 Class-2 rule (Rules that require simple derived Attribute Values) example:

(42) 3.2.2 Design Criteria (Singapore)
f) The discharge pipe shall not be located in places where it can cause health and safety hazards such as locating the discharge pipe above any portable water storage tank and electrical transformer/ switchgear.

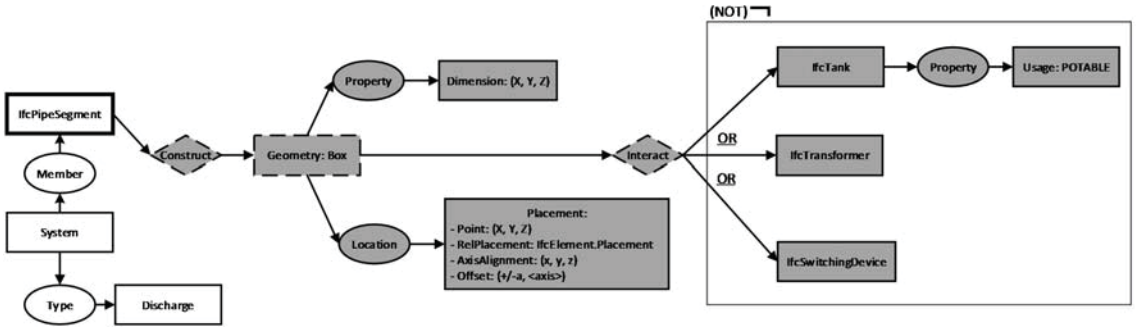

Figure 7 - Class-2 Rule Example

Representation in FOL for the above CG (Figure 7):

$$\exists b \forall a(\forall g(IfcPipeSegment(a) \wedge System(b) \wedge Discharge(g) \wedge Member(b,a) \wedge Type(b,g))$$
$$\wedge (\forall c \forall p \forall d(Box(c) \wedge Placement(p) \wedge Dimension(d) \wedge Location(c,p) \wedge Property(c,p))$$
$$\wedge Construct(a,c)) \wedge \neg(\exists w \exists t \exists m(Interact(c,(\forall u(IfcTank(w)$$
$$\wedge Usage(u,POTABLE) \wedge Property(w,u)) \vee IfcTransformer(t) \vee IfcSwitchingDevice(m))))))$$

This expression requires an extension function to construct a transient Box geometry based on a specified location and dimension. This Box is used to evaluate the existence of any type of objects within the Box that is a type of a potable water tank, a transformer, or a switching device. Their existence within the Box is not allowed.

As shown in Figure 7, in this class-2 rule, we start to see the need of extensions to generate a new concept. Three such extensions are required in here: a simple box geometry, and two functions to construct the box and to perform spatial operation to find the specific object types that interact with the constructed box that is placed below the discharge pipe.

3.1.3 Class-3 rule (Rules that require extended data structure) example:

IBC 1005.2 Door encroachment. Doors, when fully opened, and handrails shall not reduce the required means of egress width by more than 7 inches (178 mm). Doors in any position shall not reduce the required width by more than one-half. Other nonstructural projections such as trim and similar decorative features shall be permitted to project into the required width a maximum of 1-1/2 inches (38 mm) on each side. (ICC, 2009).

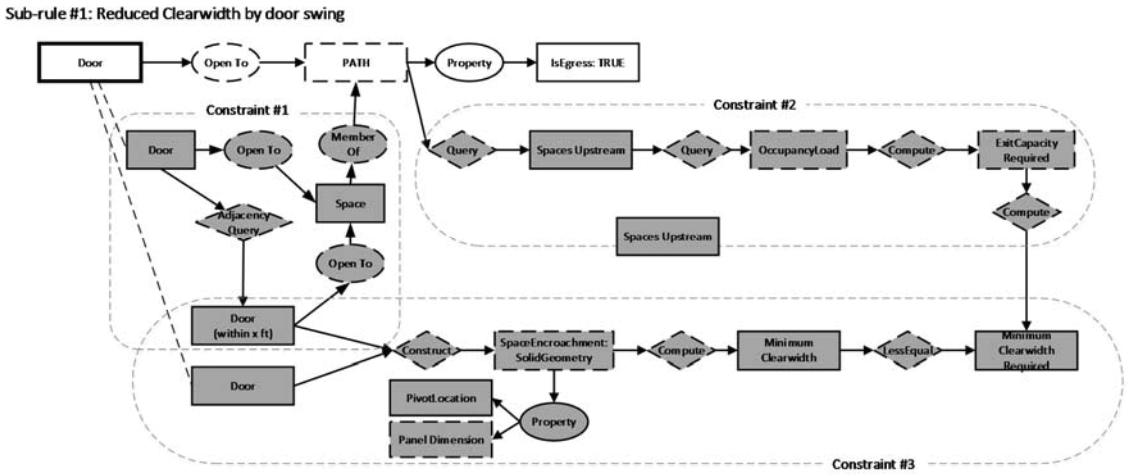

Figure 8 - Class-3 Rule Example – Sub-rule #1

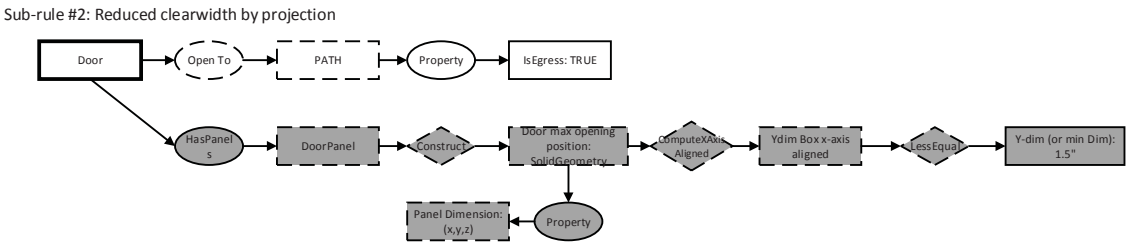

Figure 9 - Class-3 Rule Example – Sub-rule #2

In FOL the above graph for the sub-rule #1 (Figure 8) can be expressed as:

$$\forall d \exists p \Big(\big(Door(d) \wedge OpenTo(d,p) \wedge PATH(p) \wedge Property(p, IsEgress: TRUE) \big) \\ \wedge \exists y \exists x (Space(x) \wedge MemberOf(p,x) \wedge OpenTo(d,x) \\ \wedge DoorList(y) \wedge AdjancencyQuery(d,y,Distance) \wedge OpenTo(y,x)) \\ \wedge \forall s \forall o \forall w \big(Space(s) \wedge QueryUpstream(p,s) \\ \wedge OccupancyLoad(o) \wedge ExitCapacity(c) \wedge Query(s,o) \wedge ClearWidth(w) \\ \wedge Compute(o,c,w) \big) \\ \wedge \forall l \forall m \forall e \forall v \big(Pivot(l) \wedge Panel(m) \wedge Property((d,y),(l,m)) \\ \wedge SpaceEncroachment(e) \wedge Construct(x,(d,y),e) \wedge Property(e,(l,m)) \\ \wedge MinClearwidth(v) \wedge Compute(e,v) \wedge LessEqual(v,w)) \Big)$$

Sub-rule #2 can be expressed in FOL as (Figure 9):

$$\forall d \exists p ((Door(d) \wedge OpenTo(d,p) \\ \wedge PATH(p) \wedge Property(p, IsEgress: TRUE)) \wedge \forall q ((DoorPanel(q) \wedge HasPanels(d,q) \\ \wedge Dimension(r) \wedge Property(q, r: (x,y,z))) \wedge \forall b \forall y (Box(b) \wedge YDim(y) \wedge Construct(q,b) \\ \wedge ComputeXAxisAligned(b,y) \wedge LessEqual(y, minYDim: 1.5")))$$

Class-3 rules requires extensive extensions both to the building model in the form of derived data as well as functions for the purpose of computation, which includes geometry operations. The above rule highlights the nature of the complexity involved. Not only is there the needs for extension, but also nested and branching conditions of the sentence that can occur in any entity within the statement. Sub-rule #1 sentence branches at PATH node into two constraints. Constraint #1 merges into constraint #3 at the beginning while constraint #2 merges at the end of constraint #3.

Sub-rule #2 in this example is an entirely independent rule from the sub-rule #1, except that it applies to the same main entity. Here, it is perfectly fine to separate the rule into two rules. In other cases, sub-rule may serve as an exception of a nested rule inside the main rule. In this case the use of coreferent concept will become handy.

4 Mapping the CG to the MVD

With the well-defined CG, it is possible to create a direct mapping of CG concepts and relations into an IFC MVD as well as to a UML diagram for software development (Figure 10). In the mapping to an MVD, IFC entities represented by Concept and Relation can be directly mapped to the IFC MVD, which includes relevant details such as Types and Properties. Each of the rules can be defined as one exchange requirement within the MVD. In practice, it may be practical to define only one or just a few specific MVDs, in order to consolidate the MVD requirements that often overlap among many of the rules.

Extended Concepts and Functions do not have equivalent mappings to an MVD and are only applicable for mapping to the software development environment, represented using a UML diagram in this example. Figure 10 shows an example of mapping from CG to an MVD and to a UML diagram.

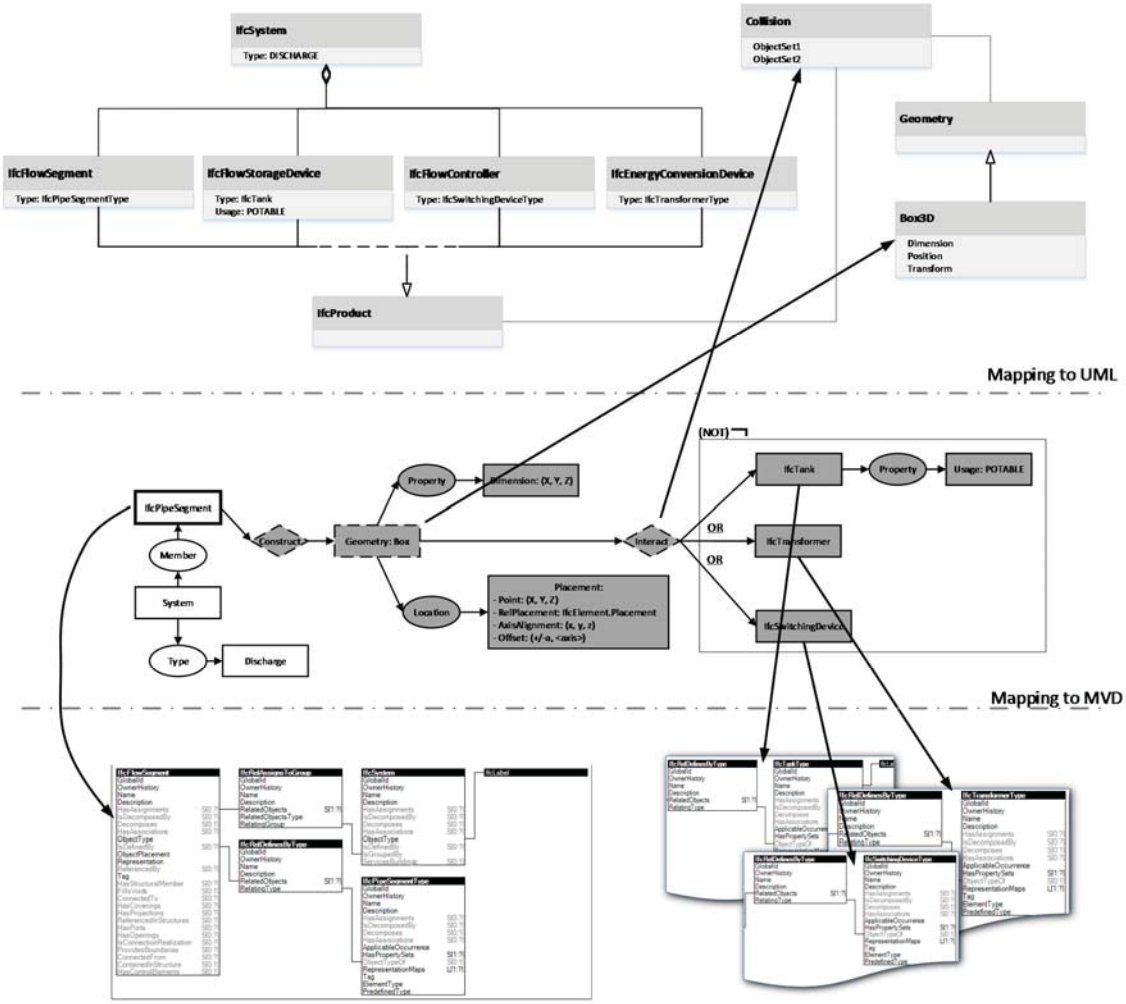

Figure 10 - Mapping CG into an MVD and UML

5 Conclusions and Future Work

We have presented the use of Conceptual Graph as a tool to capture building rule requirements and their general checking logic as a knowledge representation. It has been demonstrated that it is effective in capturing both data requirements and the higher level checking logic in an intuitive format, which can be understood by typical rule experts. The proposed CG also provides a direct mapping to an MVD for model exchange requirement relevant to the rule it describes, and a mapping to software class design such as a UML diagram. The CG provides a systematic mean to capture the knowledge and retain it as the knowledge flows in the rule development process described in Figure 1. The exercise to capture the rule expert knowledge into CG also provides a template for analysis and breaks down a complex rule into atomic rules and constraints. In our research work the method has been applied to more rules of higher level complexity and it is found to be able to describe the rule requirements and logic successfully.

The possibility of mapping the CG directly into an MVD and software classes presents an opportunity for further work to automate the mapping process. Similar work to capture requirements using CG-like representation into UML has been done (Jaramillo, Gelbukh, & Isaza, 2006). While it is not specific to building rules that involve additional complexity of derived entities, properties and functions, it may give an idea of what can be done in the same direction for building rules. The

potential time saved and knowledge retention throughout the whole development process is extremely significant.

References

Beach, T., Kasim, T., Li, H., Nisbet, N., & Rezgui, Y. (2013). Towards Automated Compliance Checking in the Construction Industry. In H. Decker, L. Lhotská, S. Link, J. Basl, & A. Tjoa (Eds.), *Database and Expert Systems Applications* (Vol. 8055, pp. 366-380): Springer Berlin Heidelberg.

Chein, M., & Mugnier, M.-L. (2008). *Graph-based knowledge representation: Computational foundations of conceptual graphs*: Springer.

Eastman, C., Lee, J.-m., Jeong, Y.-s., & Lee, J.-k. (2009). Automatic rule-based checking of building designs. *Automation in Construction, 18*(8), 1011-1033.

Fiatech. (2012). Fiatech Autocode phase 1 report.

Hjelseth, E., & Nisbet, N. (2011). *Capturing normative constraints by use of the semantic mark-up (RASE) methodology*. Paper presented at the CIB W78 2011 28th International Conference-Applications of IT in the AEC Industry.

ICC. (2009). IBC 2009 *http://shop.iccsafe.org/codes/2009-international-codes/2009-international-building-code-tab-combo.html*.

ISO. (2013). ISO 16739:2013 Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51622.

Jaramillo, C. M. Z., Gelbukh, A., & Isaza, F. A. (2006). Pre-conceptual schema: A conceptual-graph-like knowledge representation for requirements elicitation *MICAI 2006: Advances in Artificial Intelligence* (pp. 27-37): Springer.

Malsane, S., Matthews, J., Lockley, S., Love, P. E. D., & Greenwood, D. (2015). Development of an object model for automated compliance checking. *Automation in Construction, 49, Part A*(0), 51-58. doi: http://dx.doi.org/10.1016/j.autcon.2014.10.004

Salama, D. M., & El-Gohary, N. M. (2013). Semantic Text Classification for Supporting Automated Compliance Checking in Construction. *Journal of Computing in Civil Engineering*.

Singapore. Singapore Fire Code *http://www.scdf.gov.sg/content/scdf_internet/en/building-professionals/publications_and_circulars.html*.

Solibri. Solibri Model Checker. *http://www.solibri.com/solibri-model-checker/functionality-highlights.html*. Retrieved 03/01/2014, 2014

Solihin, W., & Eastman, C. (2015). Classification of rules for automated BIM rule checking development. *Automation in Construction, 53*(0), 69-82. doi: http://dx.doi.org/10.1016/j.autcon.2015.03.003

Solihin, W., Shaikh, N., Rong, X., & Poh, L. K. (2004). *Beyond Interoperability of Building Models: A Case for Code Compliance Checking*. Paper presented at the BP-CAD Workshop, Carnegie Melon University. https://www.researchgate.net/publication/280598933_BEYOND_INTEROPERATIBILITY_OF_BUILDING_MODEL_A_CASE_FOR_CODE_COMPLIANCE_CHECKING

Sowa, J. F. (1976). Conceptual graphs for a data base interface. *IBM Journal of Research and Development, 20*(4), 336-357.

Sowa, J. F. (1984). *Conceptual Structures - Information Processing in Mind and Machine*: Addison-Wesley Publishing Company.

Zhang, J., & El-Gohary, N. (2012). Extraction of Construction Regulatory Requirements from Textual Documents Using Natural Language Processing Techniques. *Proc., Comput. Civ. Eng*, 453-460.