BIM-based Automated Planning for Panelized Construction in the Light-Frame Building Industry

by

Hexu Liu

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Construction Engineering and Management

Department of Civil and Environmental Engineering
University of Alberta

ABSTRACT

Building information modelling (BIM) has been recognized as an information technology with the potential to profoundly change the Architecture, Engineering, and Construction (AEC) industry, and has drawn attention from numerous scholars within the construction domain. Despite the reported advancements pertaining to BIM in previous studies, the use of BIM in planning panelized construction (e.g., construction-centric design detailing, construction-oriented quantity take-off, and detailed construction scheduling) has not yet reached its full potential. Discipline-specific BIM design models from architects and structural engineers are insufficient to serve the needs of the construction field. This research thus explores the extended use of BIM to facilitate automated planning for panelized construction.

In terms of construction-centric design detailing, this research exploits a BIM-rule-based automated approach to designing and modelling drywall and sheathing layouts with minimized material waste in order to promote building panel production. In the proposed approach, object-based computer-processable layout design rules are comprehensively formalized based on trade know-how and construction best practice, and integrated with mathematical algorithms in order to generate the optimized boarding layout design with minimized material waste. For construction-oriented quantity take-off, this research proposes an ontology-based semantic approach to extracting construction-oriented quantity take-off information from a BIM design model. This approach allows users to semantically query the BIM design model using domain vocabularies, capitalizing on building product ontology formalized from construction perspectives. As such, quantity take-off information relevant to construction practitioners can be readily extracted and visualized in 3D in order to serve application needs in the construction field. Lastly, this research presents a BIM-based integrated scheduling approach that facilitates the

automatic generation of optimized component-centric activity-level construction schedules for panelized building projects under spatial and resource constraints, by achieving an in-depth integration of BIM product models with work package information, process simulations, and optimization algorithms. This research prototypes an automated planning system for panelized building construction as add-on tools of Autodesk Revit. Three case studies are presented to demonstrate the proposed methodology. Building on the existing body of knowledge in this field, the key contribution of the present research is that it defines three practical problems in a scientific manner and introduces three novel approaches in order to adapt BIM design models for construction practitioners and to advance the current planning practice in panelized construction by integrating construction-oriented intelligence into BIM.

PREFACE

This thesis is the original work by Hexu Liu. Three journal papers and three conference papers related to this thesis have been submitted or published and are listed as below. This thesis is organized in paper format by following the paper-based thesis guideline.

1. **Liu, H.**, Singh, G., Lu, M., Bouferguene, A., Al-Hussein, M. "BIM-based Automated Design and Planning for Boarding of Light-Frame Residential Buildings." *Automation in Construction.*(under review)

2. **Liu, H.**, Lu, M., Al-Hussein, M. (2016). "Ontology-based semantic approach for construction- oriented quantity take-off from BIM models in the light-frame building industry." *Advanced Engineering Informatics,* 30(2), 190-207.

3. **Liu, H.**, Al-Hussein, M., Lu, M. (2015). "BIM-based Integrated Approach for Detailed Construction Scheduling under Resource Constraints." *Automation in Construction*, 53, 29-43.

4. **Liu, H**., Singh, G., Lu, M., and Al-Hussein, M. (2015). "BIM-enabled Boarding Design Optimization for Residential Buildings." *Proceedings, 2015 International Conference on Construction Applications of Virtual Reality (CONVR)*, Banff, AB, Canada, Oct. 5-7.

5. **Liu, H**., Lu, M., and Al-Hussein, M. (2014). "BIM-based integrated framework for detailed cost estimation and schedule planning of construction projects." *Proceedings, 31st International Symposium on Automation and Robotics in Construction and Mining (ISARC)*, Sydney, Australia, Jul. 9-11.

6. **Liu, H.**, Lei, Z., Li, H., and Al-Hussein, M. (2014). "An automatic scheduling approach: building information modeling-based on-site scheduling for panelized construction." *Proceedings, Construction Research Congress*, Atlanta, GA, USA, May 19-21.

ACKNOWLEDGEMENT

This PhD study is a long journey. I could not come to this stage without the help of many individuals. First and foremost, I would like to thank my supervisors, Dr. Mohamed Al-Hussein and Dr. Ming Lu for their immense support, inspiration, visionary guidance, and great patience throughout the course of my study.

I would also like to express my thanks to Jonathan, Gurjeet, Claire, Hamida, Dr. Ahmed Bouferguene, and other faculty members and staff in the Construction Engineering and Management research group. I would also like to thank Ronald, Sang, Sadiq, Lei, Lily, Xinming, Hongru, Rita, Celia, Hamid, and Beda for their help and moral support.

Especially, my deepest gratitude is given to my family, who have been a great support behind me over the years. They have always encouraged me during this journey and provided valuable guidance. My wife, Ningyu, is also very much appreciated for her kind and continuous company through this phase of my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AEC	Architectural, Engineering, and Construction
API	Application Programming Interface
BIM	Building Information Modelling
CAD	Computer-aided Design
DES	Discrete-event Simulation
GUI	Graphic User Interface
IDEF0	Icam Definition for Function Modelling
IFC	Industry Foundation Class
LGS	Light Gauge Steel
LINQ	Language-Integrated Query
LoD	Level of Detail
OSB	Oriented strand board
PSO	Particle Swarm Optimization
QTO	Quantity Take-off
RCPSP	Resource-constrained Project Scheduling Problem
RDF	Resource Description Framework
SPARQL	Simple Protocol and RDF Query Language
SQL	Structured Query Language
SWRL	Semantic Web Rule Language
UML	Unified Modelling Language
WBS	Work Breakdown Structure
XML	Extensible Markup Language

CHAPER 1: INTRODUCTION

1.1 Background and Motivation

Panelized construction provides a cost-effective building solution for light-frame residential buildings, and is taking the place of conventional "stick frame" construction in the residential building industry. Generally, panelized construction moves the framing of building components from the field into the efficient manufacturing environment. Compared with conventional construction systems or methods whereby the majority of construction tasks take place on site, the on-site work involved in panelized construction is limited to the assembly of factory-built building components such as wall and floor panels. Most building components, such as wall panels, are prefabricated in the factory and then delivered to the site for on-site assembly. In the factory, all panelized building components are framed on flat, square framing tables, and other operations, such as sheathing panelized building components, are also performed in the controlled manufacturing environment, resulting in higher quality and greater precision. This sort of construction thus has the potential to enhance construction efficiency and to minimize the waste involved in the construction process (National Association of Home Builders, 2009). However, it has not yet been leveraged to its full capability partially due to the ineffective use of automated and innovative technologies, such as building information modelling (BIM), in the design and planning phases. Current manual building design and planning without the provision of innovative technologies is inefficient and cannot meet the requirements of panelized construction such as a higher level of accuracy. Any errors in building design and planning could increase the cost of panelized construction compared to traditional stick-built construction.

1.1.1 Manufacturing-centric BIM

The effectiveness of discipline-specific BIM models utilized to communicate information among project stakeholders plays an important role in determining the construction efficiency (Alwisy & Al-Hussein, 2010). In practice, architects focus primarily on the architecture-centric design (e.g., architectural BIM model), while construction practitioners rely solely on their experience and tacit knowledge to interpret these discipline-specific models (e.g., architectural and structural BIM models) in order to develop mental pictures of construction/manufacturing-centric building design (i.e., manufacturing-centric BIM model). Nevertheless, the information gap between various discipline-specific models, along with the subjectivity of mental construction-specific information models, contribute largely to misinterpretation of construction-centric building design, thus resulting in construction re-work, material waste, and increase of construction costs.

Meanwhile, increasing industrialization in building construction presents higher requirements along with new challenges for building designers in terms of building information modelling and design documentation (i.e., design drafting) (Alwisy et al., 2012). Building objects in a given BIM model can be developed at different levels of detail (LoDs), ranging from LoD 100 to LoD 500 (ASBO, 2013). With the increase in LoD, building information and design details are increasingly leveraged into BIM models to represent the size, shape, location, quantity, orientation, and non-graphic information of the building (Ramaji & Memari, 2016). Increasing the LoD from one level to another increases by a margin in the range of two to eleven times the modelling time (Leite et al., 2011). Provided that building objects in the given BIM model have been roughly designed by architects and engineers (i.e., to LoD 300 or less), they cannot satisfy the requirements of contractors and fabricators. Manufacturing-centric BIM in this study refers to a BIM model at an LoD of 350 or higher, which can represent detailed subcomponents (e.g.,

blocking, studs, plates, wall bracing, sheathing and drywall sheets, and so on) of building components (Webster, 2014). However, without construction intelligence in existing design and drafting software, designers must devote a significant amount of time to modelling the building design at the appropriate level of detail (e.g., manufacturing-centric BIM) and ensuring the accuracy of the shop drawings to support the manufacturing needs. Improper building design and modelling contributes markedly to primary material waste during construction. Material waste has been identified as a major problem in the construction industry. In particular, the North American construction industry produces up to 24% of all municipal solid waste (Laquatra & Pierce, 2004). According to National Association of Home Builders (1999), the construction of a typical 2,000 ft^2 residential house can even lead to 8,000 lb of solid waste, to which drywall waste alone contributes approximately 2,000 lb (see Figure 1.1). Annually, there are 61,100 tons of gypsum drywall waste directed to landfills in Alberta, Canada (Yu, 2010). Moreover, a home builder must spend over $500 on construction material waste disposal on each new house construction (Home Innovation Research Labs, 2001). Automating manufacturing-centric building design and modelling eliminates the need for designers to spend a significant amount of time to ensure the accuracy of the drawings, and it also provides an effective approach to reduce material waste and to reduce construction costs in the building manufacturing industry.

Figure 1.1 Drywall waste

1.1.2 Quantity take-off

Quantity take-off is "*a detailed measurement of the materials and labour needed to complete a construction project*" (Holm et al., 2005). It serves as the foundation for other downstream tasks in construction, such as cost estimation and schedule planning, and its accuracy can directly affect downstream analyses and decision making. The quantity take-off (QTO) process is an information extraction process during which quantities of building objects or design features are measured based on the 2D design drawings or the 3D model. However, the QTO process, at present, involves substantial manual interventions and remains labour-intensive and error-prone. To generate accurate QTO from 3D product models in an automatic manner, BIM may offer the best approach (Sattineni & Bradford, 2011). In fact, BIM-based QTO is currently the most widely used BIM-based application in the architectural, engineering, and construction (AEC) industry. Nevertheless, a BIM model itself is a purpose-built, product-centric information database, and it lacks domain semantics in connection with specific building trades such that extracting construction-oriented QTO information for the purpose of construction workface planning still remains a challenge. Note that construction-oriented QTO produces quantities in proper units of measure which are taken off for construction activities based on activity definition and detailed specifications of construction methods and materials. Moreover, some information crucial to construction practitioners, such as the topological relationships among building objects, remains implicit in the BIM design model. This restricts QTO information extraction from the BIM model for downstream analyses in construction and building manufacturing. Currently, retrieving QTO information relevant to construction practitioners from a BIM design model is still far from efficient.

1.1.3 Panelized construction scheduling

Regarding construction process planning and productivity improvement, panelized construction, with in-plant fabrication and on-site assembly being the two main processes, presents a distinctive problem: for in-plant fabrication, manufacturing process management is the main focus, while, for on-site assembly, scheduling and management of assembly operations are of particular interest. Nevertheless, prefabricated panels are unique and vary in product design features (e.g., length, having windows or doors, having various connections). Panel fabrication is recognized as a low-volume and high-variety product mix production process. Each pre-fabricated component needs to be installed at its own designed location and be scheduled individually in order to manage and coordinate factory production and on-site construction processes. The success of such projects relies on the reasonable planning for the production, shipping, and installation of the building panels, where improper planning in panelized construction can result in project delays and elevated inventory costs. Consequently, the harmony between on-site construction and in-factory production is significantly important to panelized building construction. In current practice, schedules are planned manually based on practitioners' experience and intuition. These manual methods are not suitable for planning detailed panel schedules (e.g., on-site assembly sequence and schedule) for panelized construction projects due to the large number of elements (panels) involved, and are also prone to errors. In addition, panelized construction poses some challenges to construction practitioners with respect to detailed project planning and management, such as determining the on-site assembly schedule for individual panels under both *technical* and *resource* constraints. To date, the efficiency and effectiveness of detailed project scheduling using BIM in panelized construction is insufficient.

1.2 Research Objectives

This research is built upon the following hypothesis:

"Integrating construction knowledge and trades know-how with the BIM model will improve the efficiency and relevance of construction planning and management in panelized construction."

This research explores the extended use of BIM in panelized construction planning and supplements current BIM models with trades' know-how in order to achieve automated construction planning. Specifically, this study focuses on developing automated design and planning methods and tools with respect to construction-centric design detailing, construction-oriented QTO, and detailed on-site construction scheduling for panelized building projects.

In the process of attaining this goal, the following objectives are pursued:

1) Development of a BIM-rule-based generative approach to manufacturing-centric BIM with a focus on boarding design (i.e., optimizing and modelling sheathing and drywall layout design for light-frame building) in order to adapt BIM design models for construction practitioners and to advance the current boarding practice.

2) Development of an ontology-based semantic approach to extracting construction-oriented quantity take-off (QTO) information from a BIM design model, which allows users to semantically query the BIM model using domain vocabularies, capitalizing on building product ontology formalized from a construction perspective. The proposed ontology addresses the limitation of BIM design models in terms of lacking domain semantics and aligns BIM design models with construction-oriented QTO. As a result, QTO information relevant to construction practitioners can be readily extracted and visualized in 3D in order to serve practical needs in the construction field.

3) Development of a BIM-based integrated planning methodology which achieves an in-depth integration of BIM, evolutionary optimization algorithm, and discrete-event simulation (DES) in order to automate the generation of optimized component-centric activity-level on-site construction plans.

1.3 Thesis Organization

This thesis consists of five chapters. Chapter 1 presents current practices in panelized construction planning and elaborates on the limitations in existing approaches. The goal and objectives of this research are also outlined in this chapter. In Chapter 2, the literature pertaining to BIM-enabled design, level of detail in BIM, parametric modelling, and material waste minimization is critically reviewed. Then, a BIM-rule-based automated approach to designing and modelling drywall and sheathing layouts with minimized material waste is presented. Subsequently, the implementation of the proposed boarding design methodology is illustrated in detail, and a case study of a wood-framed residential building is also presented to demonstrate the effectiveness of the methodology and the prototyped boarding design system. The proposed approach is able to incorporate manufacturing-centric design information into the given BIM model according to trades' know-how such that construction practitioners can make use of such a BIM design model in the construction field. Chapter 3 highlights the limitation of BIM design models in terms of lacking domain semantics in construction-oriented QTO, and it then presents an ontology-based semantic approach to extracting construction-oriented QTO information from a BIM design model. Afterward, the development of a semantic QTO prototype system is presented. A case study is also shown to validate and demonstrate the effectiveness of the prototype QTO system. The proposed approach allows users to semantically query the BIM model using domain vocabularies in order to retrieve construction-oriented QTO information in a

flexible, straightforward manner. Chapter 4 explains the challenges of the existing practice with respect to detailed construction scheduling under spatial and resource constraints. It then presents a BIM-based integrated scheduling approach that automatically generates optimal component-centric activity-level schedules for panelized construction projects by achieving an in-depth integration of BIM, evolutionary optimization algorithm, and discrete-event simulation (DES). Specifically, in the proposed BIM-based scheduling approach, rich product information from BIM models and work package information from a Microsoft (MS) Access Database, are automatically extracted and fed as inputs to the process simulation model that mimics construction logic and performs simulation-based scheduling analysis. This chapter also present a case study of a light-frame residential building and demonstrates the effectiveness of the scheduling methodology and the prototyped scheduling system. Finally, conclusions are summarized and the research contributions are recapitulated in Chapter 5.

CHAPER 2: MANUFACTURING-CENTRIC BIM[1]

2.1 Introduction

In North America, light-frame structures, such as light wood framing and light gauge steel framing systems, are widely used in residential buildings. Wall studs and floor joists in light-frame walls and floors need to be sheathed using sheathing and drywall boarding sheets in order to form the exterior and interior sides. Boarding design herein refers to the layout design of sheathing and drywall sheets on walls and floors according to design principles and construction best practice. In general, sheets of drywall and sheathing are available in rectangular shapes with different dimensions (e.g., 4'×8', 4'×10', and 4'×12') and varying thicknesses (e.g., 1/2″ and 5/8″). Boarding practice requires cutting nominal boarding sheets into designed dimensions followed by fastening (screwing or gluing) them to the wood (or metal) studs and joists. Boarding design is conducted either at the late design stage after all architectural and structural designs are finalized or made on an ad-hoc basis by trades during the construction phase. Design can be improved if construction practitioners are engaged earlier in the design stage to consider boarding layout and design constructability. However, in current practice, boarding design is largely overlooked by designers and construction practitioners due mainly to the fact that it entails construction-centric design knowledge and substantial effort to represent relevant information into BIM design models. For this reason, existing discipline-specific BIM models from architects and structural engineers are insufficient to serve the needs of contractors and sub-contractors during construction. In some cases, construction practitioners base decisions regarding the boarding layout design and the cutting plan of material sheets solely on their experience and rules of thumb. Such a manual process is laborious and often results in

[1] A version of this chapter has been submitted to the journal of Automation in Construction.

considerable material waste. According to the National Association of Home Builders (1999), for instance, the construction of a typical 2,000 ft^2 residential house can lead to as much as 8,000 lb of solid waste, of which approximately 2,000 lb is drywall.

Building information modelling (BIM) is a parametric modelling technology developed based on the object-oriented concept. Domain knowledge and design principles can be interpreted as object behaviours (e.g., geometric rules or constraints) of parametric objects. Such parametric objects are able to retain their design content in response to external and internal stimuli, resulting in intelligent building design (Lee et al., 2006). It offers numerous advantages with respect to alternative design generation, modelling productivity, and elimination of communication errors (Sacks et al., 2004). BIM thus has the potential to provide project designers and construction practitioners with an effective approach to addressing the challenges associated with boarding design for light-frame buildings. Nevertheless, parametric modelling has been limited in its applicability in this area due to ambiguity (i.e., one object's behaviour can be implemented in diverse ways) and complexity (i.e., one building object can be defined by a vast number of parameters and constraints that may crash a BIM model when it is modified improperly) (Lee et al., 2006). In addition, parametric building objects (e.g., a wall and its sub-components) must be designed in a hierarchical manner in order to avoid manual placements of certain parametric objects, and to minimize the amount of effort in design detailing. For example, sheathing and drywall sheets, as sub-components of wall elements, should be defined as constituent objects for wall objects with walls as the main controlling objects. Such hierarchical design requires a well thought-out plan prior to implementation.

This chapter explores a BIM-based generative approach to boarding design (i.e., optimizing and modelling sheathing and drywall layout design) in order to adapt BIM design models for

construction practitioners and to advance the current practice. In the presented approach, enriched building information, including both geometric information and functional information of building components, are extracted in order to facilitate automated generative rule-based boarding design. Comprehensive generative rules are formalized based on industry know-how and construction best practice, and are integrated with mathematical algorithms in order to generate optimized design alternatives with minimized material waste. The generated layout design not only minimizes material waste, but also enhances design constructability.

In this chapter, the literature pertaining to BIM-enabled design, level of detail in BIM, parametric modelling, and material waste minimization is critically reviewed. Subsequently, the BIM-based methodology for optimized boarding design is described. Along with this, boarding design principles are presented. Afterward, extraction of building information relevant to boarding design and optimization of rule-based boarding design is presented. A case study of a wood-framed residential building is also presented to demonstrate the effectiveness of the methodology and the prototype system. Finally, conclusions are summarized and limitations of the present research are discussed.

2.2 Literature Review

2.2.1 BIM-based design and design checking

BIM has been increasingly utilized to facilitate various project activities in the AEC industry during the project life cycle, such as building design (Kaner et al., 2008), design checking or evaluation (Jeong & Ban, 2011), quantity take-off (Liu et al., 2016), cost estimation (Ma et al., 2015; Lee et al., 2014), and project scheduling (Liu et al., 2014; 2015a; 2015b). With respect to building design, BIM is usually regarded as an extension of and enhancement to conventional CAD by building designers, and is expected to improve the productivity of building design

through enhanced functionalities in terms of visualization, navigation, and parametric modelling (Oh et al., 2015). Given this reality, Hu et al. (2010) exploited a 4D construction information model-based safety analysis approach to automatically designing and modelling scaffold systems based upon the structural analysis of temporary building structures. Notably, the scaffold system in their study is used to support temporary building structures during construction. Alternatively, Kim & Terzer (2014) developed a rule-based design and planning system using BIM for temporary scaffolding that is intended to provide construction workers with sufficient work space. Their planning system automated the processes of detecting the need for scaffolding as well as generating scaffolding design by leveraging enriched information in BIM models. Gane & Haymaker (2012) illustrated a novel methodology, namely Design Scenarios, in order to design and manage requirements-driven design spaces within CAD tools, which is intended for use in conceptual design. Alwisy et al. (2012) proposed a BIM approach to automating the design and drafting process for residential building prefabrication, with a focus on wood-framing design. Another important BIM application with respect to building design is design checking. Eastman et al. (2009) surveyed several rule-checking systems in the industry that utilize the IFC-based BIM model as input, and pointed out that "rule-based applications of building model checking for the purpose of architectural design, detailing, and building renovation are just beginning to emerge". Given this trend, Hyunjoo & Francois (2009) utilized BIM along with ontological consistency checking to identify and resolve conflicts and inconsistencies in building design during the design process. Zhang et al. (2013) developed an automated BIM platform for safety checking which assists construction practitioners in preventing fall-related accidents prior to construction.

In addition, BIM provides project stakeholders with an integrated collaboration environment. As a result, building designers in different disciplines can coordinate and communicate with one another through a unified BIM model, aiming to deliver the project within the targeted time, resources, and budget. However, effective collaborative design has not materialized in current practice due to data loss and interoperability issues in BIM-based design applications. Addressing this limitation, Oh et al. (2015) proposed an integrated design system encompassing a BIM modeller, a BIM checker, and a BIM server in an attempt to improve BIM-based collaborative design. Their system provides functions that assist building designers in storing, managing, and sharing the information generated during the collaborative design in an integrated manner.

2.2.2 Level of Detail in Building Information Modelling

Despite the fact that BIM has been increasingly employed in the AEC industry to support building design, BIM currently still lacks construction-oriented sophistication. The reason partially lies in that extensive manual efforts are required for generating construction-centric BIM. In general, building objects can be modeled in a given BIM model at different level of detail (LoDs), such as LoD 100, LoD 200, LoD 300, LoD 400 and LoD 500 (ASBO, 2013). With the increase in LoD, building information and design details are increasingly leveraged into BIM models to represent the size, shape, location, quantity, orientation, and non-graphic information of the building (Ramaji & Memari, 2016). Provided that building objects in the given BIM model have been roughly designed by architects and engineers (i.e., to LoD 300 or less), they cannot satisfy the requirements of contractors and fabricators. Building objects must be developed at LoD 350 or higher in order to represent detailed sub-components (e.g., blocking, studs, plates, wall bracing, and so forth) of building components (Webster, 2014). Construction-

centric BIM in this study refers to a BIM model at a LoD of 350 or higher. However, increasing the LoD from one level to another increases by a significant margin in the range of two to eleven times the modelling time (Leite et al., 2011). Monteiro et al. (2013) reported that the modelling time for building structural elements approximately doubles if their formwork is modelled within BIM models. Similar to formwork, precise sheathing and drywall layout information is not represented explicitly in BIM models developed by architects and structural engineers; significant effort and input are required from construction practitioners in order to enrich BIM models with this information according to practical know-how and design principles. Nevertheless, such detailed BIM models are of vital importance in project coordination and decision making in relation to construction material takeoff and usage during the design and construction stages (Liu et al., 2015).

2.2.3 Parametric modelling technology

Parametric modelling provides an effective means to improve modelling productivity and to generate design alternatives. Attempts pertaining to parametric modelling technology have been carried out in the past few decades. For example, Sacks et al. (2004) examined the requirements, features, and performance of specifications of a new 3D parametric CAD platform with a precast concrete construction example. Subsequently, Sacks et al. (2005) summarized direct and indirect benefits of parametric modelling and provided a benchmark of the impact of parametric modelling in precast construction. Lee et al. (2006) specified parametric building object behaviour (BOB) and its description notation and method for BIM design systems. Sacks et al. (2008) concluded that 3D parametric modelling improves the productivity in drawing production by up to 41%. Cavieres et al. (2011) explored knowledge-based parametric tools for concrete masonry walls. They interpreted construction and structural design knowledge into generative

rules and feedback rule-checking functions within parametric tools in order to improve design efficiency. Manrique et al. (2015) proposed a methodology for automating the generation of shop drawings for wood framing design by using a parametric model within a CAD environment. In spite of the reported advancements in the specific building domain with respect to parametric modelling, however, a fully automated design of board layouts in the light-frame building industry has not yet been achieved. In current practice, practitioners are involved to manually design the location and dimensions of sheathing and drywall sheets based on their tacit knowledge, a process which is labour-intensive, time-consuming, and error-prone. In the present research, trades' know-how and construction best practice with regard to boarding layout design are comprehensively formalized and further integrated with BIM models in order to generate design alternatives in an automated manner.

2.2.4 Material usage minimization

Material waste in the building industry is unavoidable, partially due to the fact that some building elements are generated from raw material of nominal sizes. However, waste can be minimized by implementing effective material management (e.g., mathematical algorithms for material cutting) and information technology (e.g., BIM). Previous research mainly studied material waste minimization from the managerial perspective. For instance, Formoso et al. (2002) investigated primary causes of building material waste, and proposed a number of managerial strategies to reduce it. Li et al. (2003) implemented an incentive reward program to minimize the avoidable material waste during construction. In their study, the bar-coding technique was applied to facilitate the material management program. In addition, material waste minimization was also formulated as the typical cutting-stock optimization problem, and various optimization algorithms, such as linear programming (Gilmore & Gomory, 1961) and combinatorial

algorithms (Manrique et al., 2009), were adopted to solve this problem. Recently, Costa & Sassi (2012) integrated genetic algorithms and ant colony optimization to solve the 2D cutting-stock problem for the glass industry. Aryanezhad et al. (2012) presented heuristic methods for the same problem, and demonstrated that their method is superior to other methods in terms of computational efficiency. More recently, Zheng & Lu (2016) formulated a mixed integer programming (MIP) model for the rebar cutting-stock problem with the objective of minimizing rebar cutting losses and associated total installation cost. With the advance of BIM technology, attempts have also been undertaken to minimize construction material waste with the support of CAD or BIM models. For instance, Manrique et al. (2009) integrated a combinatorial algorithm with a 3D CAD model to optimize the cutting of lumber and sheathing materials for residential buildings. Their work formalized material waste minimization as the cutting-stock optimization problem, rather than the boarding layout design optimization, and utilized only geometric information from the 3D CAD model, rather than enriched building information. In their research, design rules were limited to aligning seams on studs, such that the resulting layout design could be practically infeasible and potentially lead to considerable material waste. Porwal et al. (2012) proposed a BIM-based rebar optimization analysis approach (i.e., one-dimensional cutting waste optimization) to facilitate cost-effective decision making during the design stage. Alternatively, Cheng et al. (2013) developed a BIM-based system for estimation and planning of demolition and renovation waste. Liu et al. (2015) conceived a design decision-making framework for improving construction waste minimization performance based on BIM technology. Won et al. (2016) investigated the amount of design error-induced construction waste that could be prevented by a BIM-based design validation process, and reported that BIM-

based design validation could eliminate 4.3% to 15.2% of design error-induced construction waste.

In short, existing BIM design models are insufficient to serve the needs of specific building trades such as carpenters in the construction field. The literature review reveals that although numerous research efforts with respect to BIM-based design and material usage minimization had been attempted in certain building domains, automated design of board layouts considering comprehensive industry know-how in the light-frame building industry has not yet been realized. In addition, BIM-based parametric modelling is not capable to provide project stakeholders with optimized design solutions with minimized material waste. This study thus exploits a BIM- and rule-based automated design and modelling approach for optimized sheathing and drywall layout design which takes advantage of enriched information available in BIM models in order to automate the boarding design process. Mathematical algorithms are employed in this study in search for the optimized layout design with minimized material waste and enhanced constructability.

2.3 Methodology

Figure 2.1 provides an overview of the methodology. As illustrated, domain knowledge, including construction-centric design principles, and construction best practice, are comprehensively interpreted as object-based computer-implementable generative rules (i.e., machine-readable codes) and applied to expanded building objects within the parametric modelling system. These rules are able to take related rich building information (i.e., geometric and semantic information) from BIM models in order to formulate various boarding design scenarios with minimized joint length under construction constraints. Each design alternative is analyzed in order to generate a thorough cutting list of building elements (i.e., quantities of

sheathing/drywall sheets). Next, the cutting list and the nominal sizes of raw material (i.e., board sheets) available in the market are fed into the cutting-stock optimizer so as to generate the optimized cutting plan with minimal material waste. Finally, the design alternative with minimal material waste among all feasible design alternatives is identified as the optimized layout design. Along with optimized layout design, the material cutting plan and purchase plan are generated by the cutting-stock optimizer in order to assist construction practitioners in planning and managing the field operation. The optimized layout design is also modelled in the given BIM model in order to visualize the construction-oriented design in a straightforward manner. By doing so, construction-centric design information is incorporated into the given BIM model such that construction practitioners can make use of such a BIM model in the construction field.

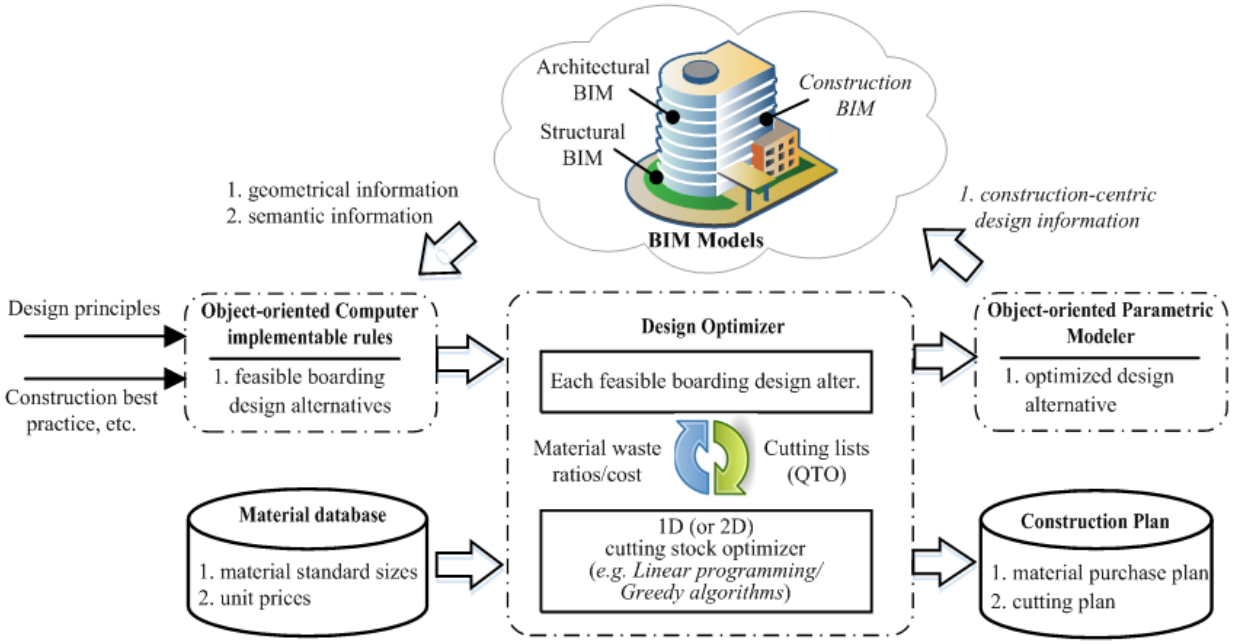
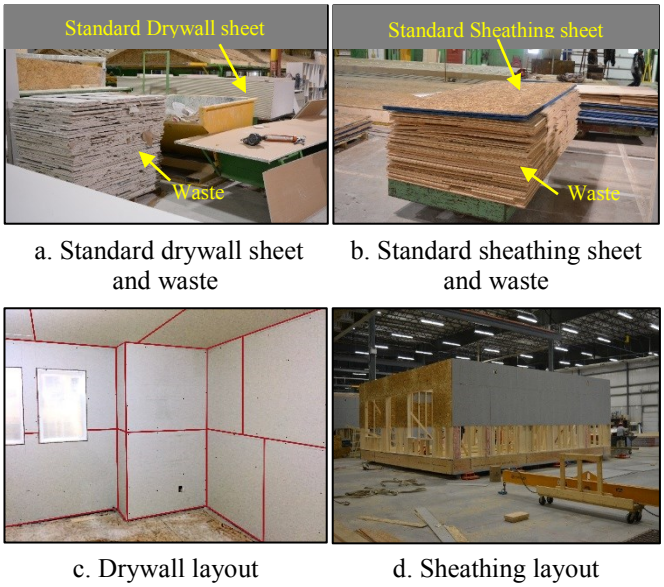


Figure 2.1 Methodology Overview

2.4 Boarding Design Principles

As described above, sheathing and drywall boarding refers to the process of cutting boarding sheets into designed sizes and then fastening (by screwing or gluing) them to the wood (or metal) studs and joists. It is important to follow certain design principles and industry know-how when laying sheathing and drywall sheets on building components in order to improve structural integrity, to reduce material waste, and to boost operational efficiency during construction. Figure 2.2 shows examples of standard boarding sheets and material waste, as well as boarding layout design of drywall and sheathing on walls. This section explains in detail the design principles and industry know-how that pertain to layout design of drywall on walls.



a. Standard drywall sheet and waste

b. Standard sheathing sheet and waste

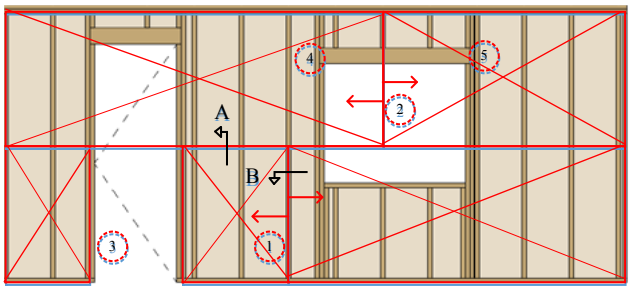
c. Drywall layout

d. Sheathing layout

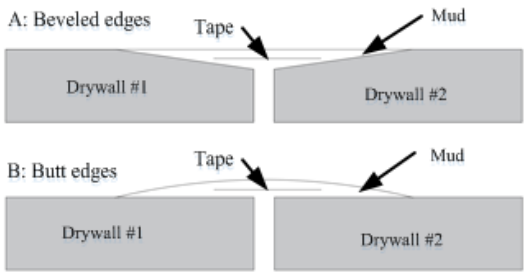
Figure 2.2 Examples of standard sheathing and drywall sheets and material waste

The design principles aim to minimize butt joints (between butt edges) and reduce cracks at seams. These requirements cause the pattern of laying the sheets of drywall and sheathing on the studs to be crucial. Generally, beveled factory edges (see Figure 2.3.b) should adjoin other factory edges, and butt edges of a drywall sheet (also shown in Figure 2.3.b) should adjoin other butt edges. When two factory edges meet, a recess for filling mud into the joints is created,

which makes taping and feathering the seams much easier. On the contrary, a drywall butt edge adjoining a factory edge could create an uneven surface, resulting in difficulty in the taping and finishing of seams. Accordingly, butt joints should be minimized whenever possible, as the area around the joint does not become flush with other areas of the drywall sheet after finishing. For this reason, boarding sheets are usually cut along the short butt edges, and board cutting is formulated as a one-dimensional cutting-stock problem in this study.



a. Staggered drywall sheet layout



b. Beveled edges and butt edges - plan view

Figure 2.3 Staggered drywall sheet layout and drywall edges

Sheets of drywall are always laid out perpendicular to the direction of wood studs where the sheets splice on the stud. When laid perpendicular to the studs, the resulting drywall structure is stronger and has greater resistance to cracking at the seams due to an increase in holding power across the wall as more studs are connected together. Also, in such a layout the seams are in the middle of the wall, making it easier to complete the tasks of taping and finishing. Placing

sheathing and drywall sheets horizontally on walls is thus a common practice in the light-frame building industry. Provided that the wall length is shorter than the height of a standard drywall sheet and the wall height is shorter than the length of a standard drywall sheet, the drywall sheet could be placed vertically parallel to the studs in order to eliminate the drywall seam on the wall. In addition, drywall butt joints should always splice on the stud and be staggered. The staggered joints lead to increased overall strength of the wall, as the staggered layout limits butt joints, which are prone to cracking, to no more than the height of a standard drywall sheet. Drywall can be hung either from left-to-right or from right-to-left along a wall. The sheets on a second wall will overlap the sheets on the first wall, creating a tight corner between the first and second wall. It is noted that the areas around the corners of openings (e.g., doors and windows) are of high stress, such that seams around these locations are prone to cracking. In addition, bulges resulting from the finishing of drywall sheets will interfere with the installation of door or window trim. As a result, in order to avoid cracks and to improve structural integrity, joints should be 10″ away from such locations. Additionally, drywall should be placed at the end of any interior wall that is not merging or connecting to any other wall. Finally, a gap of 1/8″ should always be set as the seam allowance to avoid forcing the drywall into place. Figure 2.3a shows one feasible design of drywall layout. As illustrated in the figure, staggered butt joints (i.e., drywall seams) are located at position 1 and position 2, respectively. The butt joint at position 2 is located in the middle of an opening, instead of at the opening corners such as position 3 and position 4, thus avoiding cracking, while butt joint at position 3 is resting on the edge of the door opening. In order to formulate various design alternatives, butt joints at position 1 and position 2 can be moved left or right under constraints of design rules. Therefore, the layout design can be optimized in order to minimize the material waste.

2.5 Implementation

The automated design and modelling approach is implemented as an add-on of the Autodesk Revit platform using API in C# language. The reasons for selecting Autodesk Revit as the BIM platform in this study are as follows: (1) Revit is a powerful modelling tool, which gives end-users modelling flexibility by means of its built-in functions such as Family Editor; (2) Revit supports API at the programming level; and (3) Revit supports Industry Foundation Classes (IFC), which addresses issues of interoperability. Figure 2.4 presents the architecture of the prototyped Revit-based automated design and modelling system. The inputs for the system include: (1) building design for the project, such as a BIM model containing architectural and structural frame information; (2) material sizes and prices, which indicate the nominal sizes and prices of drywall and sheathing boards on the market; and (3) boarding design patterns (i.e., horizontal stagger and vertical continuous for walls), which allow users to select boarding design patterns for walls in order to cater for the need of a vertical design pattern of boarding sheet layout. The output of this system comprises a construction-centric BIM model allowing construction practitioners to visualize the optimized boarding design in 3D, shop drawings and cutting plan of the resulting boarding design, as well as the boarding sheet purchase plan.

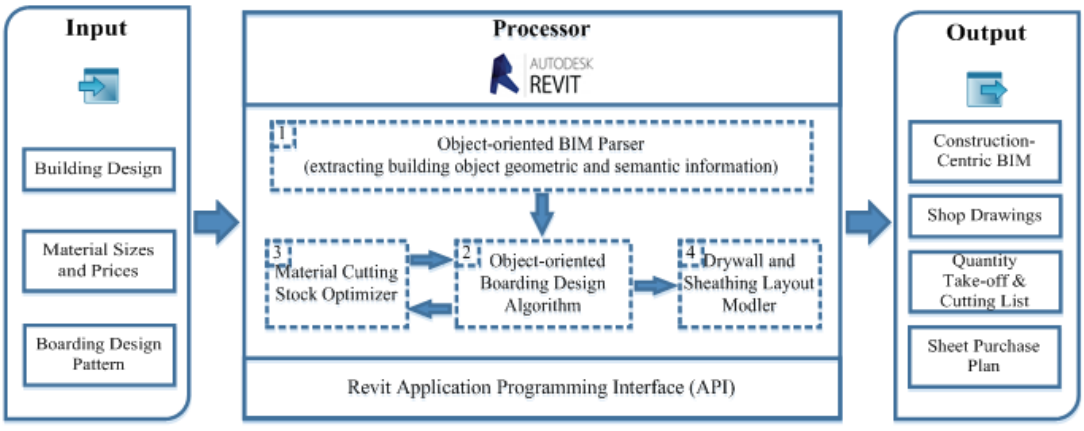


Figure 2.4 System architecture

The core of this prototype system, as shown in the centre of Figure 2.4, consists of four components: (1) object-based BIM model parser, which extracts relevant geometric and semantic information for the downstream design analysis; (2) rule-based boarding design algorithm (i.e., object-based design functions), which is used to design the drywall and sheathing layout in accordance with design principles and industry know-how; (3) cutting-stock optimizer, which is employed to optimize the boarding sheet cutting with the objective of minimizing material waste; and (4) drywall and sheathing layout modeller (i.e., object-based modelling functions), which takes the optimized design parameter as input and models the layout design in the BIM model. The four components are encoded into Autodesk Revit as add-ons through API in C#. Essentially, this study makes use of object-oriented programming principles in order to achieve automated construction design. Objects representing building elements in Revit, as shown in Figure 2.5, are extended to explicitly include properties (i.e., geometric and semantic information) and functions (i.e., object-based design functions and object-based modelling functions in a computer-interpretable form) which work together to generate feasible layout designs. Detailed implementations of the automated approach are discussed below.

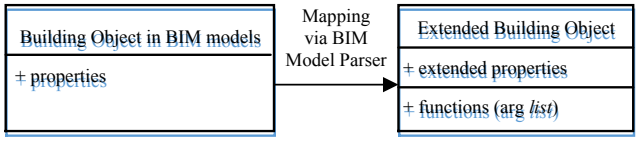


Figure 2.5 Extended parametric objects

2.5.1 Object-based building information extraction

Enriched information in BIM models is utilized to automate the boarding design process. In general, building product information in the BIM models includes geometry, topology, and functional information. Geometric information refers to vertices, edges, and faces of building

components, while topological information elaborates on their location and spatial relationships. Functional information consists of additional properties, such as host information, describing building components. The first two types of information can be held by and derived from traditional 3D CAD models, whereas functional information is only available in the BIM model and it is carried by building objects as properties. Theoretically, sheathing and drywall modelling requires geometric and topological information of relevant building elements. As such, one attempt using 3D CAD models for drywall layout design was successfully made by Manrique et al. (2009). In fact, BIM models are object-based information models in which building objects have types and enriched properties; thus, it is straightforward to recognize building components by their object type and to retrieve the associated properties. BIM technology therefore provides an effective approach to storing and managing enriched building information throughout the project life cycle. The semantic information and its object-oriented representation in BIM models are conducive to boosting the efficiency of information extraction by eliminating certain geometric analyses, thus advancing the traditional 3D CAD-based modelling approach. For example, "Host by" between openings/studs and walls can be readily recognized by the semantic host property of building elements, rather than through complex geometric analysis. Additionally, wall and floor elements usually consist of several layers, depending on the architectural design. Individual boarding sheets are designated within particular sheathing and drywall layers. As a result, semantic material information of wall layers is beneficial to identifying and extracting geometric information for specific wall and floor layers.

Some information, such as wall connections, is implicit in the traditional 3D CAD models. Such information is crucial to the boarding design in that drywall sheets, for example, need be placed at the end of interior walls that are not merging or connecting to any other walls. To do this, the

developed BIM model parser must retrieve such implicit information. In this respect, BIM models conforming to either the open BIM IFC schema or Revit data schema enable an effective means of storing and retrieving this information. For instance, IFC schema defines objectified relationships such as *IfcRelConnectsPathElements* to describe the connectivity between building elements, whereas Revit API provides functions, such as *wall.get_ElementsAtJoin(indexofWallEnd)*, to detect walls adjoined end-to-end. In addition, Revit always forces the elements to automatically adjoin their neighbours where appropriate; therefore, this Revit API function can be used to detect the connections. In short, BIM with enriched building information improves the efficiency of information extraction compared with the traditional 3D CAD model. On the other hand, BIM models are large datasets in which only a portion is needed in automating the boarding design. This study identifies the information model for the boarding design. The excerpt of this information model (i.e., specific model view) is shown in Figure 2.6. Essentially, a small number of classes (e.g., *Geometry*, *LightFrameWall*, *and LightFrameFloor as* shown in Figure 2.6) are defined within *Visual Studio* to enhance the Revit objects by explicitly representing relevant geometric and semantic information. Modelling elements in Autodesk Revit are mapped to those classes, while BIM data, including explicit and implicit data, is extracted by the BIM model parser to instantiate these objects, thereby facilitating boarding design and modelling. As shown in Figure 2.6, “BuildingComponent” is the base class that carries all general information about building components, and “Wall”, “Floor”, “Plate”, and “Stud” are inherited from “BuildingComponent”. Basically, these sub-classes extend “BuildingComponent” with specific properties and functions. For instance, “Plate” and “Stud” have the property of “Host” indicating their hosting element. “WallLayer” and “FloorLayer” are inherited from “Geometry” and are associated with “Wall” and “Floor”,

respectively. The function of "GetDesignLayout()" is attached to "FloorLayer" and "WallLayer", which utilizes the relevant building information to generate feasible layout designs. The design rules executed within this function are explained in the following section.

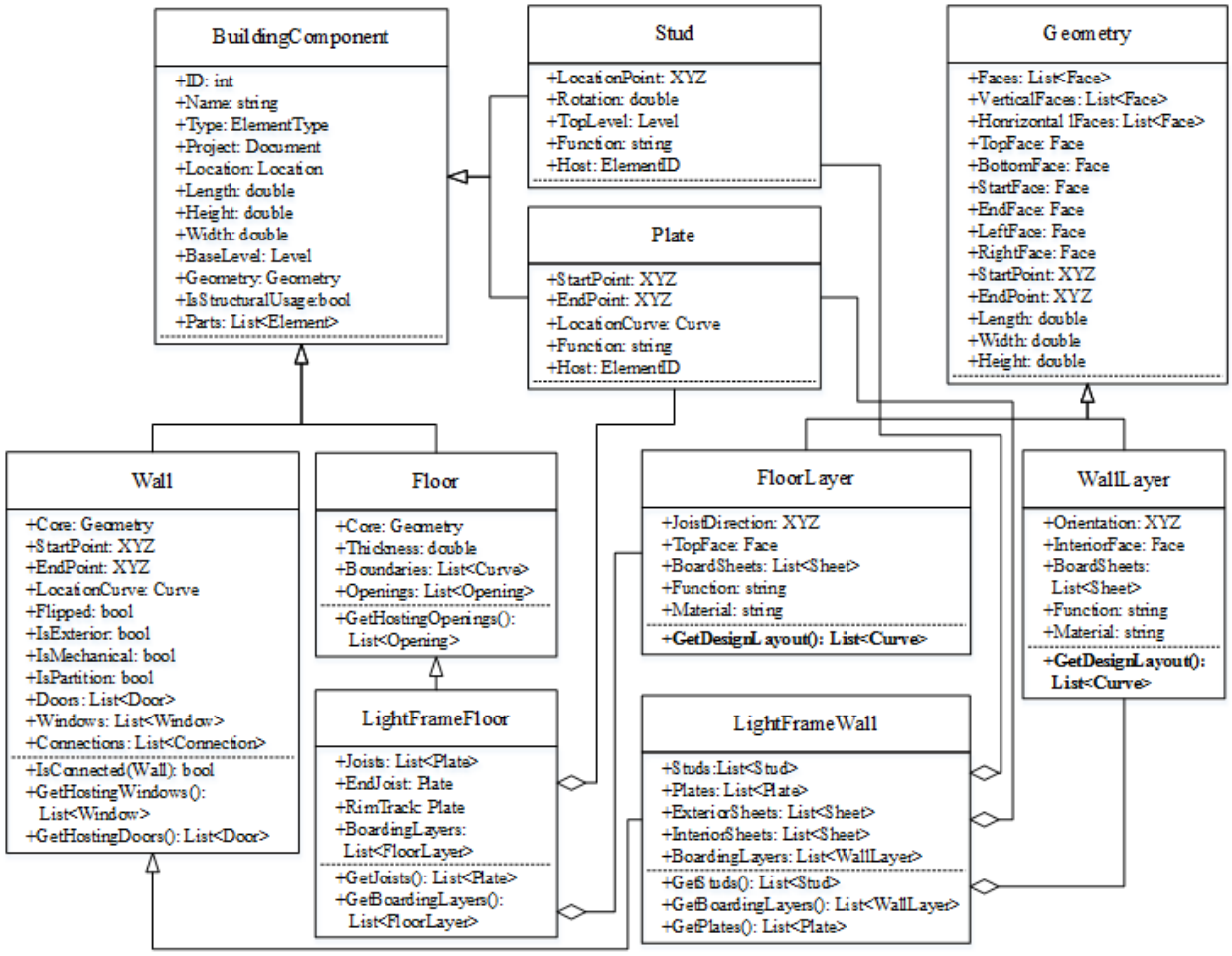
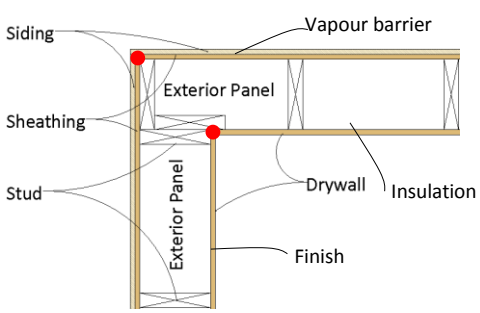
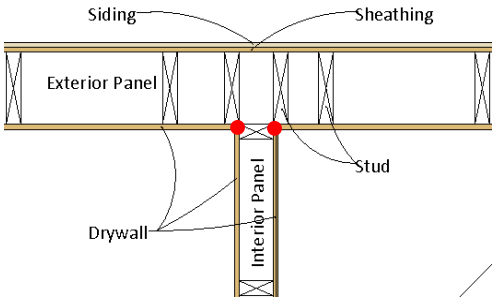


Figure 2.6 Excerpt of information model for boarding layout design using UML

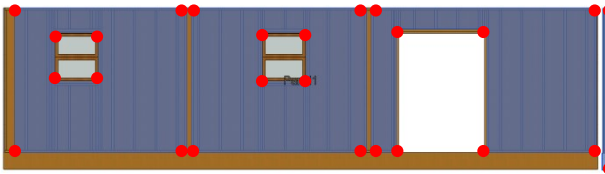
Figure 2.7 presents a sample sketch of a wall panel and its associated sheathing and drywall layers and openings, while a sample sketch of a floor and its associated sheathing and drywall layers is shown in Figure 2.8. Geometric information (e.g., vertices of drywall and sheathing layers) extracted by the BIM model parser is highlighted by the red dots in Figure 2.7 and Figure 2.8.



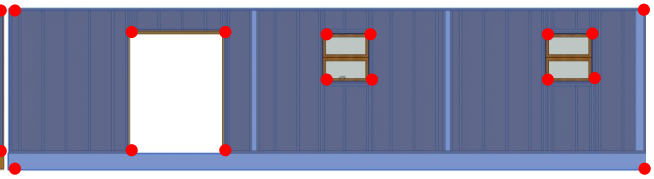
a. Plan view of L-Connection



b. Plan view of T-Connection

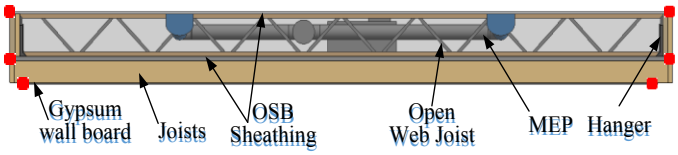


c. Interior elevation of wall

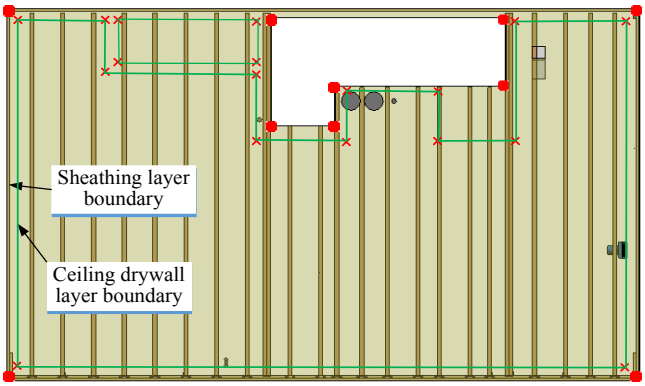


d. Exterior elevation of wall

Figure 2.7 Schematic diagram of wall layers



a. Floor elevation view



b. Floor plan view

Figure 2.8 Schematic diagram of floor layers (drywall ceiling and OSB sheathing)

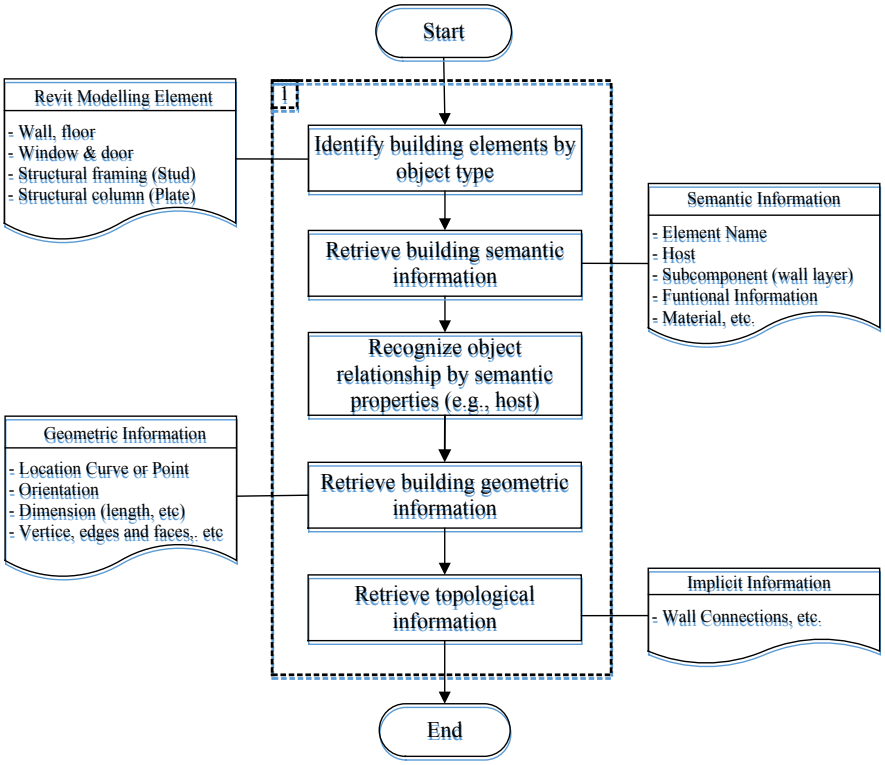
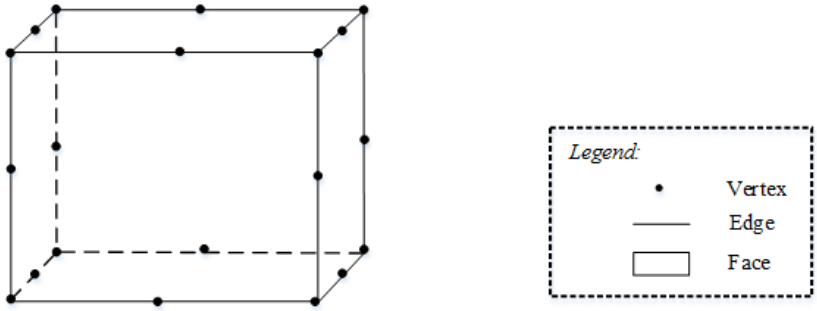


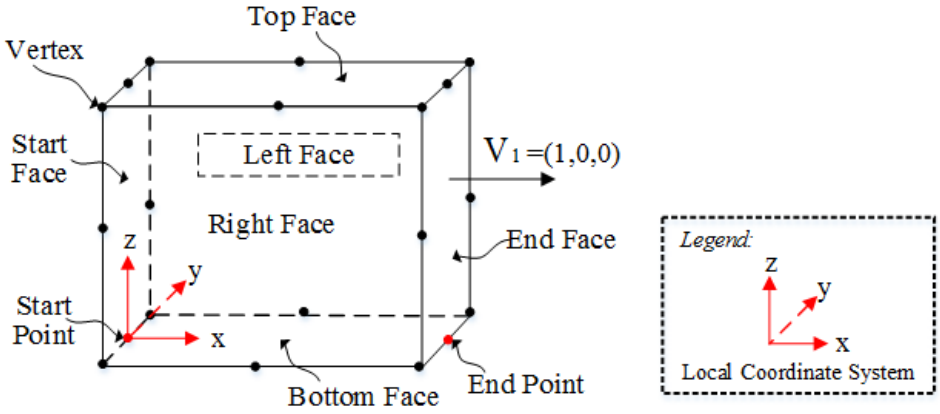
Figure 2.9 Flowchart of building information extraction

The flowchart of building information extraction by the developed BIM model parser is illustrated in Figure 2.9. To begin, the BIM model parser identifies all building elements relevant to sheathing and drywall design and modelling by their types, such as walls, floors, windows, doors, studs, plates, and joists. Subsequently, their enriched functional information, such as host information (e.g., panel name) and wall/floor layer (e.g., material), is retrieved through the Revit API functions of "*element.get_Parameter(paraName)*" and "*elementType.get_Parameter (paraName)*", which is used to detect relationships between walls/floors and their sub-components (e.g., studs and joists). On this basis, geometric information, such as location of studs and windows, is then extracted using the Revit API functions of *element. get_Geometry(), solid.Faces, face.EdgeLoops,* and *curve.GetEndPoint()*. Geometric information for individual sheathing and drywall layers is retrieved using the same geometric functions based upon

semantic material information in the given BIM model. Notably, the boundary representation is used within Autodesk Revit to represent a solid with vertices, edges, and faces as shown in Figure 2.10.a. The developed BIM model parser includes a set of algorithms which interpret the geometrical information (i.e., vertices, edges, and faces) of each solid component. For instance, the normal vector of each face is checked against the vector $(0, 0, 1)$ to determine whether or not they face the same direction (i.e., $V_1 \cdot V_2 = 0 \times 0 + 0 \times 0 + 1 \times 1 = 1$) in order to identify top faces as shown Figure 2.10.b. All this geometrical information is then stored in the class Geometry (see Figure 2.6). This information, along with formalized design principles, is used to design boarding in the following section.



a. Boundary representation of a solid



b. Interpreted geometrical information of a solid

Figure 2.10 Geometrical information of a solid component

2.5.2 Rule-based boarding design

Boarding design principles, including industry know-how and construction best practice, are interpreted as object-based rules. Examples of design rules include *Lay sheet edge on stud, Stop sheet edge at opening, Stagger sheet edge,* and *Avoid edge around opening corner.* These rules are utilized in the design process to generate feasible design alternatives. Basically, once enriched functional and geometric information has been retrieved from the BIM design model, a rule-based boarding design algorithm is launched. The methodological flowchart of the wall boarding design algorithm is presented in Figure 2.11. It begins with identification of the boarding layers of one wall. Then, the sheet orientation is determined based upon the user's configuration of the design pattern for walls, wall dimensions, and board nominal sizes. Following this, board sheet rows are determined by comparing wall height and board height. For each sheet row, the algorithm begins by identifying its start-point; then, one sheet of the board of nominal size is placed accordingly (i.e., vertically or horizontally) at the identified start-point. Subsequently, the end-point of the sheathing/drywall board is calculated. Next, this end-point is checked against the object-based rules in order to ensure that formalized design principles, such as *Lay sheet edge on stud, Stop sheet edge at opening, Stagger sheet edge,* and *Avoid edge around opening corner*, are satisfied. In the case of any non-compliance, the sheathing/drywall board is cut shorter to adjust its end-point, and a new end-point satisfying all design rules is re-

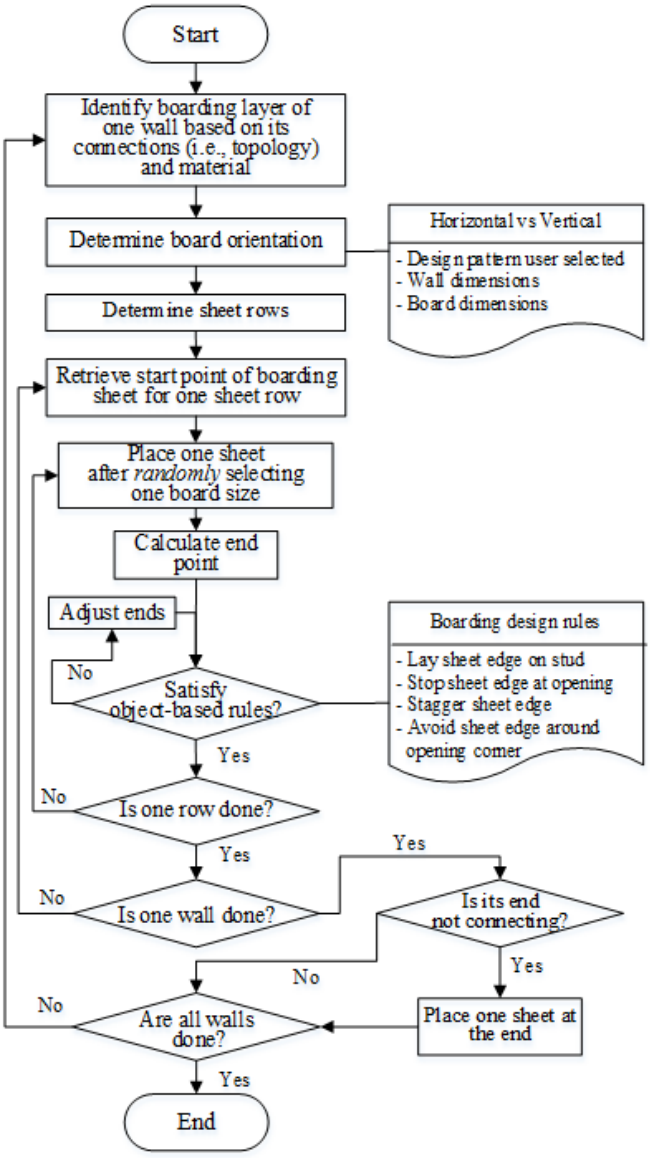


Figure 2.11 Flowchart of wall boarding design algorithm

calculated by the algorithm. This end-point then serves as a new start-point at which to place the next sheathing/drywall board. The processes for one wall do not terminate until all boarding sheet rows have been placed. Finally, connection information of this wall is checked. One boarding sheet will be placed vertically at the end when this end is not connected with other walls. The same process will be applied to all other walls in the BIM model, and the design algorithm does not terminate until boarding sheets have been placed on all walls in the BIM

model. The partial detailed implementation of the wall boarding design algorithm in C# is provided in Appendix B.

A similar procedure is followed for floor boarding layout (see Figure 2.12). It also begins by identifying boarding layers. Then, the joist direction is identified in order to determine the boarding sheet orientation, because the boarding sheet orientation is always perpendicular to the joist direction. Once the boarding sheet orientation is determined, rows of boarding sheets on the floor are calculated. Subsequently, the start-point of one row of sheets is retrieved, and one sheet of the board of nominal size is placed at the identified start-point; then, the end-point of the board is calculated. The board is then checked to confirm whether it covers one opening as shown in Case 1. If so, the board edge will be adjusted to the nearest opening edge. Similar to the wall design algorithm, this end-point is also checked against the object-based rules in order to ensure that formalized design principles are satisfied. In case of any non-compliance, the sheathing/drywall board is cut shorter to adjust its end-point, and a new end-point satisfying all design rules is re-calculated by the algorithm. This end-point then serves as a new start-point at which to place the next sheathing/drywall board. The processes for one row do not terminate until all boarding sheet layers have been placed, and, in turn, the design algorithm does not terminate until boarding sheets have been placed on all floors.

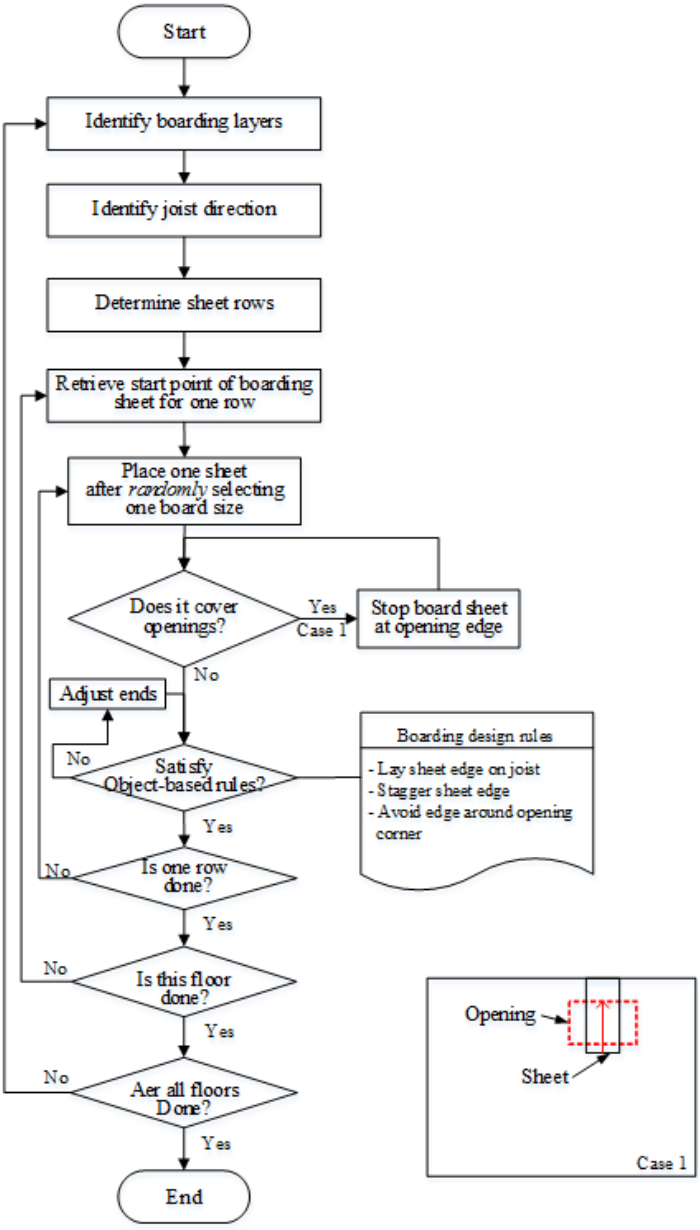


Figure 2.12 Flowchart of floor boarding design algorithm

2.5.3 Design optimization

This research studies the material usage optimization from the design point of view. The objective of this research is to generate the near-optimal boarding layout design with minimized material waste for light-frame buildings under construction constraints. The objective function is expressed in Equation (1):

$$O.F. = min\{\mathrm{W}_1, \mathrm{W}_2, \cdots, \mathrm{W}_d\}, \quad \mathrm{d} = 1,2, \ldots, \mathrm{N} \tag{1}$$

$$\mathrm{W_d} = min\left(\sum_{i=1}^{x} L_{i,d} - \sum_{j=1}^{t} y_{j,d}\right) \tag{2}$$

$$q_d = \{y_{1,d}, y_{2,d}, \cdots, y_{t,d}\} = f(num.of\ seams,\ loc.of\ seams) \tag{3}$$

$$\text{St.} \quad \frac{Length\ of\ boarding\ row}{Max.(Nominal\ board\ sizes)} \leq num.of\ seams \leq \frac{Length\ of\ boarding\ row}{Min.(Nominal\ board\ sizes)} \tag{4}$$

Where, $O.F.$ represents the objective function; W_d denotes the minimized material waste associated with the design alternative d; d is the index of one design alternative in a list of N design alternatives; $L_{i,d}$ denotes the length of board stock $\boldsymbol{i}$; x is the number of stocks; $y_{j,d}$ is the length of jth boarding sheet; and t is the number of boarding sheets generated according to design rules; q_d is the quantity take-off (i.e., cutting list) for the design alternative d, and it is determined based on boarding design (i.e., number of seams and location of seams), as expressed in Equation (3) and Figure 2.3.a; $num.of\ seams$ and $loc.of\ seams$ are the decision variables of the mathematical model; "loc. of seams" should always be subject to all the boarding design rules, and "num. of seams" in each boarding sheet row should be subject to Equation (4). Notably, altering the design in terms of number of seams and location of seams will lead to the generation of different q_d value, resulting in different material usage and material waste. The minimized material waste of one design alternative, $\mathrm{W_d}$, is expressed as in Equation (2) and is calculated by the cutting-stock solver in this study.

In general, the sheathing and drywall design optimization is an iterative process, as shown in Figure 2.13. For each iteration, one design alternative is formulated by using boarding design algorithms described in the previous section. The optimization algorithm randomly selects one nominal boarding size (e.g., $4' \times 8'$, $4' \times 10'$, or $4' \times 12'$) when placing individual boarding sheets (see Figure 2.11 and Figure 2.12). Upon completion of the boarding design configuration for all

panels (i.e., walls or floors), the design alternative is analyzed to obtain a thorough cutting list of sheathing/drywall sheets (i.e., quantities of cutting items). The cutting-stock solver is then triggered and takes the combination of available material stock sizes and the generated thorough cutting list as inputs to formulate an optimized cutting plan, which minimizes material cutting waste for this design scenario. Three algorithms, Greedy First Fit, Greedy Best Fit, and Greedy Next Fit, are executed sequentially in the cutting-stock solver (Montibelli, 2014). Greedy algorithms are selected in this study due to the fact that they can provide optimized solutions in a reasonable timeframe (Esparza, 2003). For a given design scenario, the optimized cutting waste and material cost can be obtained from the cutting-stock solver. After saving this design scenario, the next iteration is then triggered and another combination of nominal boarding stock sizes (i.e., number of seams and location of seams) is used by the algorithm to formulate a new design. The algorithm does not terminate until it reaches certain termination criteria, such as completing the specified number of iterations (e.g., 100). Finally, the design with the minimized amount of material waste is identified by the prototype system, and the successful design is used to formulate the Microsoft Excel-based boarding sheet purchase plan and cutting plan, which are automatically generated by the prototype system.

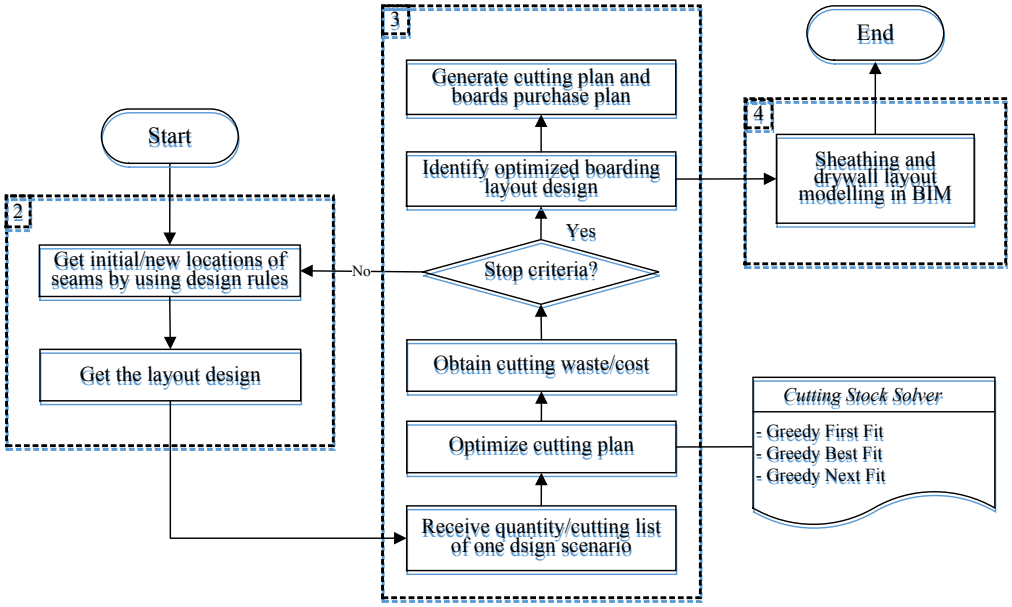


Figure 2.13 Flowchart of optimizing boarding design

2.5.4 Boarding layout design modelling

Ultimately, the optimized layout design is modelled automatically in the BIM model by the sheathing and drywall layout modeller, as shown in Figure 2.4 and Figure 2.13, in order to obtain a construction-centric BIM model and to generate shop drawings. Since this study is implemented as an add-on for Autodesk Revit, the boarding layout design is modelled by using the construction modelling functions of Autodesk Revit such as "Divide Parts". It should be noted that "Part" is a modelling element allowing construction modellers to plan the installation of pieces of building components and their sub-components. Part elements can be generated from layer-structured modelling elements such as Wall and Floor in order to represent their layers (Autodesk Revit, 2015). Furthermore, by using the Revit API function, "*PartUtils.DivideParts (doc, partsToBeCutted, id, cutting Curves, sketchPlane)*", these parts can be divided into smaller discrete parts, which can be independently scheduled for the purpose of construction planning. As a result, part elements are used to model individual boarding sheets. The sheathing and

drywall layout modeller transforms the optimized layout design into an array of curves that cut the layers of sheathing and drywall within Revit API. These curves represent the boarding seams and are the inputs of the above mentioned Revit API function. By doing this, the design modelling is materialized in an automated manner.

2.6 Case Study

The developed prototype system is tested in a wood-framed single-family house. The building shown in Figure 2.14 consists of three storeys and 75 wall panels, including 68 light-framed walls and seven precast basement walls. Oriented strand board (OSB) sheathing boards are placed on exterior sides of exterior light-framed wall panels and top sides of floor panels, while gypsum drywall boards are used for interior sides of exterior light-framed wall panels and bottom sides (i.e., ceiling) of floor panels, as well as for both sides of interior light-framed wall panels. The building model is first built in Autodesk Revit 2015; then, a suite of commercial Revit add-ons, Metal Wood Framer (MWF) (StrucSoft Solutions, 2015), is employed to frame building components such as wall panels and floor panels. Following this, the developed prototype system is launched in Autodesk Revit to design the boarding layout. Construction practitioners must provide available nominal board size and corresponding unit cost information through the graphic user interface (GUI), as shown in Figure 2.14, to this prototype system, thereby enabling design optimization.

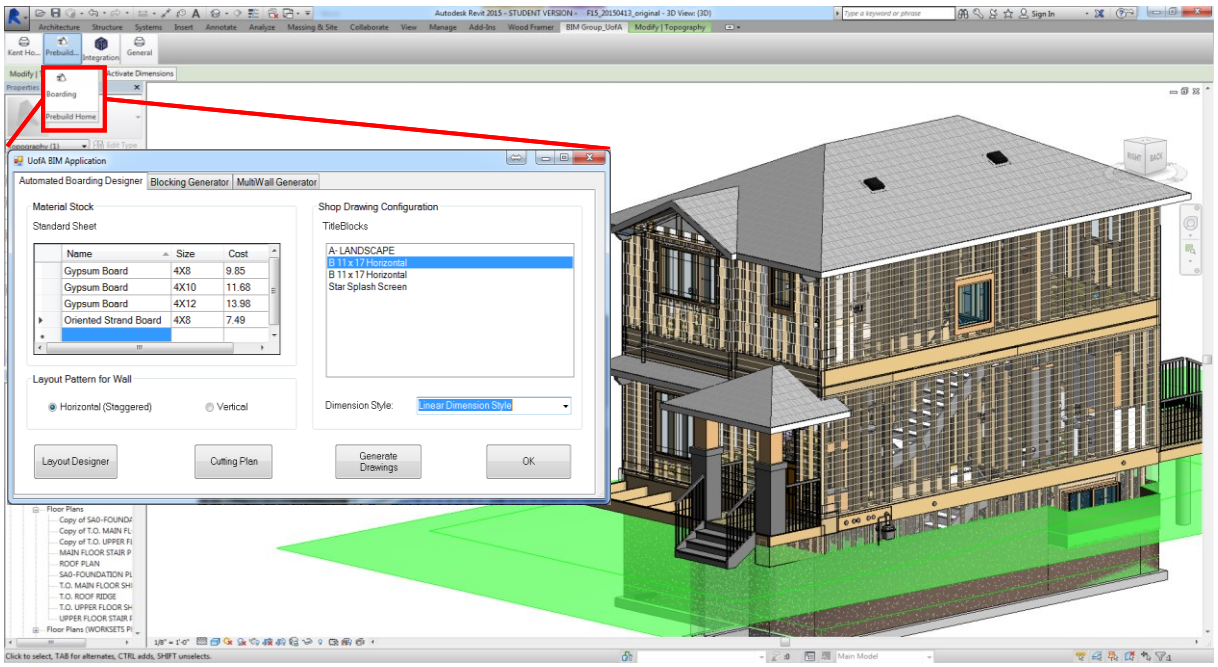


Figure 2.14 Interface of automated boarding designer

Table 2.1 Cost (in CAD) and material waste information of optimized boarding layout design

	Material	Size (ft)	Unit Price	Number of Sheets	Material Cost	Material Waste Ratio
Wall	Gypsum Board	4 × 8	$9.85	86	*$1,535.14*	*6.85%*
		4 × 10	$11.68	23		
		4 × 12	$13.98	30		
	Oriented Strand Board	4 × 8	$7.49	68	*$509.32*	*7.3%*
Floor	Gypsum Board	4 × 8	$9.85	31	*$552.39*	*6.53%*
		4 × 10	$11.68	2		
		4 × 12	$13.98	8		
	Oriented Strand Board	4 × 8	$7.49	60	*$449.4*	*5.84%*

Table 2.1 tabulates the nominal board information as inputs for the case study. Total material cost of optimized boarding layout designs and their material waste information are also summarized in this table. As shown in Table 2.1, the material cost of gypsum drywall for walls is CAD 1,535.14 with a material waste of 6.85%, while its sheathing material cost is CAD 509.32 with a material waste of 7.3%. As for floors, the material cost of gypsum boards is CAD 552.39 with a material waste of 6.53%, while its sheathing material cost is CAD 449.4 with a material waste of 5.84%. Additionally, the GUI allows for users to set up shop drawing configurations in

order to generate shop drawings for drywall and sheathing layout on walls. The system outputs, including construction-centric BIM (box 1 in Figure 2.15), shop drawings with quantity take-off (box 2 in Figure 2.15), as well as the Excel-based cutting list and boarding sheet purchase plan (shown in Table 2.2), are generated automatically by clicking corresponding buttons on the GUI. Part of the generated Excel-based cutting plan (see Table 2.2) shows how many sheets are cut from standard material boards and where each sheet is installed. In addition, all standard material boards are listed in "Size". Based on this information, construction practitioners can plan and manage the prefabrication. It is worth noting that the material waste of drywall sheets averages 12% according to the California Integrated Waste Management Board (2007). In comparison with the generated results from the prototyped system, material waste for drywall sheets is reduced below the industry benchmark. In the future, other optimization technologies will be investigated to further optimize the boarding design.

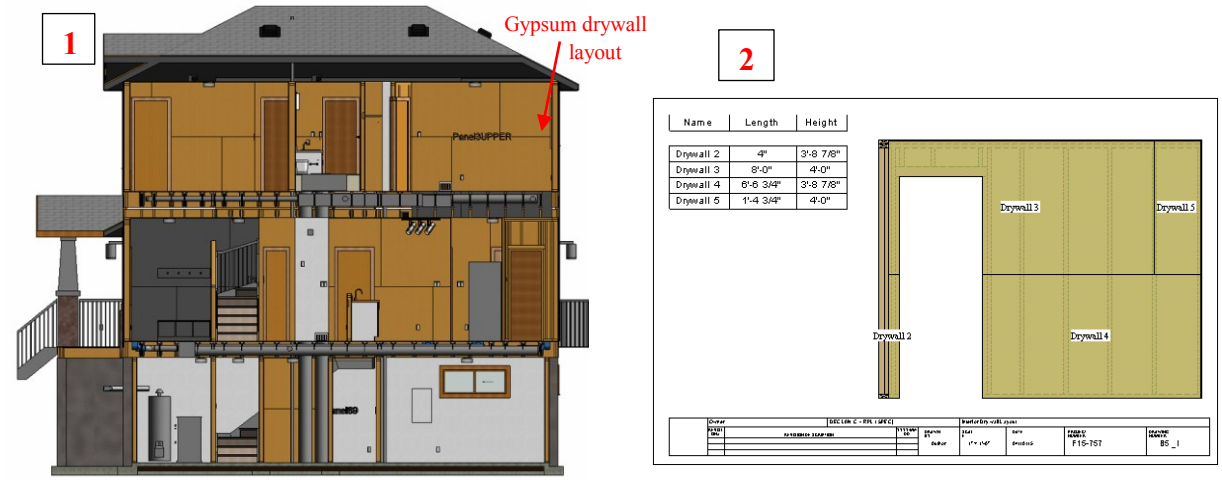


Figure 2.15 Outputs of automated boarding designer

Table 2.2 OSB board sheet cutting plan for floors in Excel

Count	Size	Used (SF)	Waste	Cutting Plan	
				Cutting List	Location
1	4′ 0″ × 8′ 0″	18.33	43%	4′ 0″ × 8′ 0″	Floor Panel 2
9	4′ 0″ × 8′ 0″	32	0%	4′ 0″ × 8′ 0″	Floor Panel 2
9	4′ 0″ × 8′ 0″	32	0%	4′ 0″ × 8′ 0″	Floor Panel 1
9	4′ 0″ × 8′ 0″	32	0%	4′ 0″ × 8′ 0″	Floor Panel 3
1	4′ 0″ × 8′ 0″	31.26	2 %	4′ 0″ × 8′ 0″	Floor Panel 3
1	4′ 0″ × 8′ 0″	31.2	2 %	4′ 0″ × 8′ 0″	Floor Panel 1
1	4′ 0″ × 8′ 0″	31.9	0 %	4′ 0″ × 8′ 0″	Floor Panel 1
1	4′ 0″ × 8′ 0″	30.88	4 %	4′ 0″ × 8′ 0″	Floor Panel 3
1	4′ 0″ × 8′ 0″	28.49	11 %	4′ 0″ × 8′ 0″	Floor Panel 2
1	4′ 0″ × 8′ 0″	31.64	1 %	4′ 0″ × 8′ 0″	Floor Panel 2
1	4′ 0″ × 8′ 0″	25.19	21 %	3′ 11 1/4″ × 8′ 0″	Floor Panel 2
1	4′ 0″ × 8′ 0″	30.42	5 %	3′ 9 5/8″ × 8′ 0″	Floor Panel 1
1	4′ 0″ × 8′ 0″	24.30	24 %	3′ 9 5/8″ × 8′ 0″	Floor Panel 3
1	4′ 0″ × 8′ 0″	23.17	28 %	3′ 11 1/4″ × 8′ 0″	Floor Panel 2
1	4′ 0″ × 8′ 0″	25.19	21 %	3′ 11 1/4″ × 8′ 0″	Floor Panel 3
1	4′ 0″ × 8′ 0″	30.80	4 %	0′ 2 1/2″ × 4′ 9 67/256″	Floor Panel 2
				0′ 5 1/2″ × 8′ 0″	Floor Panel 3
				1′ 0″ × 0′ 8 1/4″	Floor Panel 3
				0′ 5 1/2″ × 8′ 0″	Floor Panel 2
1	4′ 0″ × 8′ 0″	29.94	6 %	3′ 11 1/4″ × 7′ 7 1/4″	Floor Panel 1
1	4′ 0″ × 8′ 0″	29.94	6 %	3′ 11 1/4″ × 7′ 7 1/4″	Floor Panel 3
1	4′ 0″ × 8′ 0″	29.94	6 %	3′ 11 1/4″ × 7′ 7 1/4″	Floor Panel 2
1	4′ 0″ × 8′ 0″	31.92	0 %	4′ 0″ × 6′ 8″	Floor Panel 3
				3′ 11 1/4″ × 1′ 4″	Floor Panel 3
1	4′ 0″ × 8′ 0″	31.92	0 %	4′ 0″ × 6′ 8″	Floor Panel 1
				3′ 11 1/4″ × 1′ 4″	Floor Panel 2
1	4′ 0″ × 8′ 0″	31.92	0 %	4′ 0″ × 6′ 8″	Floor Panel 2
				3′ 11 1/4″ × 1′ 4″	Floor Panel 1
1	4′ 0″ × 8′ 0″	25.08	22 %	4′ 0″ × 6′ 3 1/4″	Floor Panel 1
1	4′ 0″ × 8′ 0″	25.08	22 %	4′ 0″ × 6′ 3 1/4″	Floor Panel 3
1	4′ 0″ × 8′ 0″	25.08	22 %	4′ 0″ × 6′ 3 1/4″	Floor Panel 2
1	4′ 0″ × 8′ 0″	24.24	24 %	3′ 11 1/4″ × 6′ 1 111/128″	Floor Panel 1
1	4′ 0″ × 8′ 0″	31.28	2 %	3′ 11 1/4″ × 5′ 7 83/128″	Floor Panel 1
				4′ 0″ × 2′ 3 1/4″	Floor Panel 1
1	4′ 0″ × 8′ 0″	32	0%	4′ 0″ × 5′ 4″	Floor Panel 2
				4′ 0″ × 2′ 8″	Floor Panel 2
1	4′ 0″ × 8′ 0″	32	0%	4′ 0″ × 5′ 4″	Floor Panel 3
				4′ 0″ × 2′ 8″	Floor Panel 3
1	4′ 0″ × 8′ 0″	32	0%	4′ 0″ × 5′ 4″	Floor Panel 1
				4′ 0″ × 2′ 8″	Floor Panel 1
1	4′ 0″ × 8′ 0″	28.83	10 %	4′ 0″ × 4′ 11 1/4″	Floor Panel 1
				4′ 0″ × 2′ 3 1/4″	Floor Panel 3
1	4′ 0″ × 8′ 0″	28.83	10 %	4′ 0″ × 4′ 11 1/4″	Floor Panel 3
				4′ 0″ × 2′ 3 1/4″	Floor Panel 2
1	4′ 0″ × 8′ 0″	19.75	38 %	4′ 0″ × 4′ 11 1/4″	Floor Panel 2
1	4′ 0″ × 8′ 0″	32	0%	4′ 0″ × 4′ 0″	Floor Panel 2
				4′ 0″ × 4′ 0″	Floor Panel 3
1	4′ 0″ × 8′ 0″	30.42	5 %	4′ 0″ × 4′ 0″	Floor Panel 1
				4′ 0″ × 3′ 7 1/4″	Floor Panel 1
1	4′ 0″ × 8′ 0″	28.83	10 %	4′ 0″ × 3′ 7 1/4″	Floor Panel 3
				4′ 0″ × 3′ 7 1/4″	Floor Panel 2

2.7 Discussion

Construction-oriented design detailing such as boarding design is a labour-intensive, time-consuming, and error-prone task in the AEC industry, which partially presents one major barrier that impedes the adoption of BIM in the construction field. Furthermore, trades' know-how in the construction industry remains mainly in the minds of those experienced trades people and is generally missing from existing design and drafting software. As such, existing design and drafting software cannot provide the functionalities to facilitate the automated construction and manufacturing-centric design. Moreover, the construction and manufacturing-centric designs generated from existing design and drafting software usually are not subject to practical constructability analysis, resulting in massive material waste and re-work in construction. This study thus successfully integrates trades' know-how, which is formalized based on the boarding practice of light-frame buildings and mathematical algorithms with BIM models. The generated layout design not only minimizes material waste, but also enhances design constructability and cost-efficiency in construction. Notably, the generated boarding layout design and board cutting plan are optimized by the prototype system, but may not represent the global optimal solution due to their non-deterministic polynomial-time hard characteristics and the heuristic feature of the proposed algorithm. In fact, the layout design optimization is an optimization problem where the board cutting optimization is nested. In this study, the optimized design is determined by selecting the best scenario among all the generated feasible design alternatives. For each design alternative, the board cutting plan optimization is solved by formulating a one-dimensional cutting-stock problem. Greedy algorithms are used to solve this cutting-stock problem due to short computing time to converge on solutions, while it is noted the generated cutting plan is practically feasible but may not be the global optimal solution. Within a limited time frame, the

prototyped system is able to generate an optimized layout design and detailed cutting plan taking into consideration of constructability. In the future, the following extensions can be pursued in order to improve the performance of the prototyped system: (1) cutting-stock optimization can be extended from one dimension into two dimensions in order to further reduce material waste; and (2) other combinatorial algorithms can be incorporated into this prototype in order to optimize boarding design more efficiently. In addition, the layout design optimization is conducted for floors and walls separately. It is anticipated that optimizing boarding layout for walls and floors simultaneously could further reduce boarding material waste.

2.8 Conclusion

This study seeks an automated approach to boarding layout design for light-frame residential buildings by utilizing BIM. An Autodesk Revit-based automated design and modelling prototype system is developed through Revit API. This prototype system incorporates common boarding design principles and cutting-stock optimization in order to provide optimized boarding design to construction practitioners. Furthermore, a BIM model parser is developed as the basis of this study to retrieve enriched building information in BIM models, thereby enabling BIM-based automated design. The prototype system is tested using a wood-framed residential building project. The generated boarding design from the prototyped system is found to minimize seam length and material waste while ensuring design constructability. It demonstrates that the prototyped system is capable of assisting stakeholders in planning and managing the prefabrication process.

The main contribution of the present research is the problem definition and automation of the solution in designing board layouts and planning board cutting, by taking advantage of enriched building information in BIM models, and comprehensively formalizing industry know-how in

terms of current boarding practice. In comparison with traditional 3D CAD technology, BIM contains enriched product information of building components in the form of object-oriented data schema, thus enabling more effective information extraction and facilitating automated rule-based design. Comprehensive industry know-how is successfully represented in a computer-interpretable form as object-oriented rules in order to formulate boarding layout design with enhanced constructability at the workface level. BIM models are integrated with formalized object-oriented design rules and mathematical algorithms in order to optimize design solutions and to reduce material waste. Additionally, this study supplements current BIM models with construction-oriented intelligence (i.e., trades know-how) and generates construction-centric BIM that is instrumental to carpenter trades in performing their work at the workface.

CHAPER 3: SEMANTIC QUANTITY TAKE-OFF[2]

3.1 Introduction

A building information model (BIM) is a digital representation of physical and functional characteristics of a facility (National BIM Standard, 2013). It is a product-centric and object-oriented information model whereby enriched building information is hosted by parametric building objects (e.g., walls and floors) as properties. This information can be retrieved from a BIM design model for building design analyses, such as energy analysis and structural analysis. Hence, the BIM model has the ability to support decision making in various aspects of the AEC industry, and boosts work efficiency by minimizing the rework of modelling or collecting building information for different purposes. As such, a large body of research has been focused on leveraging BIM models with discipline-specific information and information exchange between a BIM authoring program and discipline design tools. Nevertheless, it remains a challenge to tailor BIM to suit construction management tasks such as quantity take-off (QTO) in connection with workface planning, *which is "the process of organizing and delivering all the elements necessary, before work is started, to enable craft persons to perform quality work in a safe, effective and efficient manner"* (COAA 2014). This is due to the fact that the BIM product model and the construction process model rely on different schemas to organize product and process data. A BIM model, including the Industry Foundation Classes (IFC) based open BIM model, is product-centric and represents an assembly of parametric building objects with properties, whereas a process model is a collection of processes usually organized by a material and method classification system (e.g., the MasterFormat developed by Construction Specifications Institute and Construction Specifications Canada) on the basis of material

[2] A version of this chapter has been published in the journal of Advanced Engineering Informatics, 30, pp. 190-207.

information, construction method, product design feature, and so forth. For this reason, one activity with a particular construction method (unique production rate and unit cost) in the process model might be only applicable to a specific group of building elements or for a portion of one building element or for a group of non-explicitly modelled building design features in a BIM product model. It is challenging for construction practitioners to obtain quantities in connection with construction activities from a BIM design model. Considerable human intervention must thus be involved to interpret the process model and to manually quantify the BIM product model in accordance with the process description.

Indeed, BIM-based QTO is currently the most widely used BIM-based application in the AEC industry. However, the quantities extracted from a BIM design model usually consist of tabular data of explicitly modelled building element dimensions and are product-oriented. This quantity information needs to be further manipulated by means of formulas or filter/aggregation functions in order to obtain construction-oriented QTO information for use by construction planners and trades personnel. Such a cumbersome manual process poses a challenge from the perspective of construction practitioners who take off work packages for detailed construction planning. Furthermore, some information that is relevant to construction practitioners is only implicitly represented in the BIM model, such as the topological relationships and various intersections among the building elements. It is challenging for construction practitioners to extract such specific building information from a BIM design model when it has not been modelled explicitly. For instance, although the open BIM IFC schema defines objectified relationships such as "IfcRelConnectsPathElements" in order to describe the connectivity between building elements, various connections (e.g., L-connection or T-connection), as well as their detailed properties (e.g., connection angle), are not explicitly defined in either IFC or the Autodesk Revit schema. Hence,

instances of “T-connection” or “L-connection” representing the connection of walls are not explicitly present in the BIM design model. For this reason, information pertaining to connections (e.g., L-connection) cannot be readily extracted. Such implicitly modelled information restricts space-related information extraction (e.g., quantities of specific types of intersections); hence, the BIM design models are insufficient to account for the details necessary to serve the intended purpose. Additionally, existing BIM design models lack standardized industrial BIM object definitions in specific building domains. For example, studs and plates in light-frame walls are usually represented as “Structural Column/Framing” in the Autodesk Revit BIM design model and as “IFCMember” in the IFC-based BIM design model. These representations are not sufficient for construction practitioners (e.g., trades personnel) in taking off their work packages. As such, BIM design models lack domain semantics in terms of specific building trades. Construction practitioners need to understand the various complex BIM schemas or BIM object definitions in terms of their specific decomposition structure in order to obtain the desired QTO. This would considerably increase the workload and difficulty in their daily planning work. Given this reality, the varying object definitions at present make the BIM models less useful to construction practitioners in performing their specific tasks, while retrieving QTO information relevant to construction practitioners from a BIM design model without domain semantics is still far from efficient.

This chapter presents an ontology-based semantic approach to extracting construction-oriented QTO information from a BIM design model. It allows users to semantically query the BIM model using domain vocabularies, capitalizing on building product ontology formalized from a construction perspective. The proposed ontology addresses the limitation of BIM design models in terms of lacking domain semantics and aligns BIM design models with construction-oriented

QTO. As such, quantity take-off information relevant to construction practitioners can be easily extracted and visualized in 3D in order to serve practical needs in the construction field. A prototype application is implemented in Autodesk Revit to demonstrate the effectiveness of the proposed approach in the domain of light-frame building construction.

3.2 Literature Review

To date, various approaches have been explored by which to retrieve QTO information from 2D design drawings or 3D CAD/BIM models in an automated manner, such as generating quantities using Open BIM-based QTO systems (Choi et al., 2015). Among these, BIM has emerged as the best automated approach to generating accurate QTO from 3D product models (Sattineni & Bradford, 2011). Most BIM applications are able to provide the QTO feature and allow the nearly seamless quantity information exchange for downstream analyses such as cost estimation. Nevertheless, BIM-based QTO may not provide all the necessary quantity data about the product in the event that the BIM model is not designed with sufficient construction detail. To realize automatic QTO at a sufficient level of construction detail, the BIM model has to be "redesigned", which demands even more effort than performing manual QTO process. As such, some studies have sought to explore an automatic approach to designing the BIM model in performing a QTO. Kim et al. (2009) explored an automated modelling method by which to model a building's interior. Liu et al. (2015c) studied an automatic approach to construction-centric BIM with the main focus on the sheathing and drywall modelling for a residential house. Noting that once the detailed information is represented in the BIM model, the thorough QTO in the form of tabular data could be generated by use of the routines in BIM tools. All these efforts pertaining to automatic modelling can improve the efficiency of QTO. Nevertheless, leveraging the BIM model may also result in a redundant information database and further pose challenges to

retrieving specific quantity take-off information. In this context, Monteiro & Poças Martins (2013) reported that modelling guidelines enable users to extract a thorough QTO in accordance with existing specifications. Those modelling guidelines could filter the relevant information at the modelling phase, rather than at the quantity extraction phase, thus boosting the QTO efficiency.

One important factor impeding BIM-based QTO applications in the construction field is that some information, such as the spatial or topological relationships among building objects, is implicit in the BIM model. To tackle this problem, Borrmann et al. (2009b) developed a spatial query language for BIM models which enabled the spatial analysis of building and partial building information extraction. The newly developed query language covered spatial operators such as *mindist, maxdist, isCloser* and *isFarther*, which was proven to be a promising approach for partial model extraction that satisfies certain spatial constraints. Subsequently, this spatial query language was extended by adding other topological operators, including *within, contain, touch, overlap, disjoint* and *equal* in the 3D space using the 9-intersection model (Borrmann & Rank, 2009a; Daum & Borrmann, 2014). Nepal et al. (2012) described a methodology for querying the BIM model for construction-specific spatial information. Custom spatial XQuery predicates such as *Overlaps, Touches, Disjoint, Intersect, Proximate,* and *On-grid* were created to support spatial queries over the BIM model. Similarly, Kim & Cho (2015) proposed a geometric reasoning system, namely, Construction Spatial Information Reasoner (CSIR) that derives construction-specific spatial information of a BIM model in order to support automated construction planning. Indeed, the use of ontology technology can help reduce manual involvement in recognizing design conditions that considerably affects construction costs. Staub-French et al. (2002; 2003) formalized a feature ontology to represent the cost-driving features of

building product models such that practitioners can generate cost estimates more expeditiously. Nepal et al. (2013) described a new approach using ontology-based feature modelling for construction information extraction from a given BIM model. In their approach, a feature ontology including feature type and feature property is formalized and a feature-based model is generated by the developed feature extractor in order to facilitate construction-specific information extraction. The information extraction is realized through formalized form-based query specification templates. Semantic query is not supported, and queried results cannot be visualized in the BIM model. Additionally, detailed information about component intersections cannot be identified in the proposed method. Lee et al. (2014) illustrated an ontology-BIM-based approach for building cost estimation with the limitation of only focusing on tiling work. In their study, ontological inference was utilized to search for work items that are pertinent to particular building elements and materials on the basis of BIM data.

Another main challenge associated with QTO is the classification system used to organize the quantity measurements (Monteiro & Poças Martins, 2013). Today, there are a few classification systems, such as MasterFormat, UniFormat, and internal formats in companies that are commonly adopted by construction practitioners and scholars. For example, Zhao et al. (2015) explored an automatic approach of QTO for modular construction which pre-loaded the industry company's classification system—called "part number"—into the BIM model during the modelling phase. Thus, quantities can be automatically extracted from the BIM model into the unit price database in Excel according to the pre-loaded classification system via the Autodesk Revit application programming interface (API). Similarly, Choi et al. (2015) leveraged a BIM model with a 10-digit construction classification code in order to facilitate the QTO process in their prototype system. The lack of a standard classification system challenges construction

practitioners in regard to compatibility among various documents and quantity information exchanges during a project life cycle (e.g., from design to construction). Additionally, quantities extracted from a BIM design model are usually in the form of tabular data of explicitly modelled building elements with product-oriented dimension values. Human intervention is still required to manipulate (e.g., filter and group) this tabular data in order to obtain the quantity compliant with the work package description and the work breakdown structure (WBS). In contrast to rule-based QTO, Lawrence et al. (2014) introduced a flexible mapping strategy which augments a BIM-based design model with cost information in order to create and maintain the cost estimation. The developed flexible mapping approach described relationships between explicit BIM objects and cost items through queries (in the XQuery language or structured query language) on the building design; it was conducive to estimating in terms of updating the cost estimation. The proposed approach was intended for the early design stage of projects even when the design is still incomplete and evolving. Substantial effort and XQuery knowledge were required to formulate complex queries, presenting a hurdle which impeded its adoption in the AEC industry.

On the other hand, BIM-based QTO is an information extraction process during which quantities of building elements or design features are determined based on the 3D product model. A large number of studies emphasized on extracting specific information from the BIM model. In general, building information was extracted either from a BIM model in commercial software (e.g., Autodesk Revit and Tekla) or from an IFC-based open BIM model as inputs for downstream analyses such as construction scheduling and cost estimation. For instance, Liu et al. (2014; 2015a; 2015b) investigated a BIM-based automatic scheduling approach whereby enriched building information, including QTO, was extracted from a Revit BIM model via

Autodesk Revit API. Kim et al. (2013) established a prototype for automating the generation of construction schedules by automating quantity data extraction from an IFC-based BIM model, and parsing building information as the inputs for scheduling. Zhang & Issa (2013) reported ontology-based partial building information extraction from an IFC-compliant BIM model by means of semantic search, instead of pure syntactic analysis. However, their research encompassed only the geometry portion of IFC specifications. Ma et al. (2011) identified an information requirement model in accordance with construction estimating practices for tendering in China, and extended existing IFC schemas to account for specific information requirements respectively. Subsequently, Ma et al. (2013) introduced a semi-automatic method to conduct cost estimation for tendering building projects based on the use of a design model through the open IFC standard. Further attempts to enhance information exchanges among BIM applications have been carried out in recent years. For instance, Yang & Zhang (2006) presented a new approach to the development of building design objects with the objective of enabling semantic interoperability in building designs. Venugopal et al. (2012) proposed an object-oriented and modular mechanism for embedding semantic meaning in model views in order to improve information exchanges among BIM applications.

In short, an ontology-based semantic QTO approach, which enables construction practitioners to semantically query BIM design models using domain vocabularies in order to retrieve building quantity information from a construction perspective, has yet to be formalized.

3.3 Background and Scope

3.3.1 Ontology and semantic query

In the context of computer science, ontology is defined as "*explicit formal specifications of the terms in the domain and relations among them*" (Gruber, 1993). In other words, an ontology is a

formal definition of types, properties, and relationships of domain entities, which provides the vocabularies to describe the domain knowledge. Ontology is thus a promising solution to share common understanding of domain knowledge (e.g., the structure of information) (Noy & McGuinness, 2001). Within the ontology, classes (types) with properties describing themselves represent the terms or concepts in the domain, whereas relations describe interrelationships among classes (terms). Although ontological modelling is similar to object-oriented modelling in the view of syntax (e.g., class and property), ontology allows for *explicitly* representing domain terms and their relations in the form of *class, property*, and *relationship* in an intuitive and structured manner. Specifically, classes, properties and relationships are stand-alone entities in ontology such that properties and relationships can exist without classes. Ontology allows for multiple inheritances among classes, properties, and relationships, respectively (e.g., sub-class, sub-property, and sub-relationship can be explicitly defined). Ontology allows for arbitrary user-defined relationships among classes, whereas the class relationship in object-oriented modelling is limited to the subclass-superclass hierarchical relationship (Siricharoen, 2007). It is noteworthy that objects in object-oriented modelling are related through attributes (i.e., properties) and objectified relations (i.e., classes), rather than through explicit relationship entities as in ontological modelling. Ontology allows users to explicitly specify characteristics/properties (e.g., symmetry, transitivity, and inversion) to relationships and the nature of the relationships (e.g., Equivalent To) between classes, properties, and relationships. More importantly, ontology, founded on logic, represents domain knowledge in an intuitive and structured manner such that ontology allows for automated reasoning that enables the user to check for conflicts of ontologies and infer new facts. For this reason, information extraction from a given BIM repository can be minimized, and the remainder can be inferred on the basis of

extracted information within ontology. In addition, ontology is easier to be extended and merged due to the fact that the substantial work needed for mapping and converting data in different applications can be reduced (Ma et al., 2015). Ontology allows for semantic queries capitalizing on formalized classes as well as properties and relationships. An ontology-based query can "understand" the semantic definition of these ontology entities; hence, it can retrieve the defining triples from the schema resources (Beetz et al., 2009). Due to the fact that ontology mixes the schema specification with individual specific data, ontology-enabled semantic query using domain vocabularies can be executed not only on specific ontology data, but also on the ontology terms (Martinez-Cruz et al., 2011). In comparison, Language-Integrated Query (LINQ) enabled by object-oriented modelling can only be executed on specific data sets, rather than data set schema. All these features, which are generally not provided by object-oriented modelling, make ontology superior in the representation of domain knowledge. Ontology has thus been successfully applied in various industries to facilitate domain knowledge management. In the construction industry, some ontology applications have also been successfully carried out in order to assist project stakeholders in cost estimation (Lee et al., 2014), code compliance checking (Zhong et al., 2012; 2015), construction planning (Benevolenskiy et al., 2012), and so forth. With BIM being an important focus in both industry and academia, research has been undertaken to maximize the benefit from integrating BIM and ontology for the AEC industry. For instance, Beetz et al. (2009) lifted the IFC specification onto a logically rigid and semantically enhanced ontological level by strictly transforming EXPRESS schemas into ontologies, and developed IfcOWL. Zhang & Issa (2013) explored an ontology-based algorithm in the extraction of a partial BIM model from the original model in order to reduce the difficulty of manipulating the complete model, which was typically large and complex. More recently,

Venugopal et al. (2015) proposed IFC reforms using ontology in order to address the limitation of IFC in terms of lacking semantic clarity and ambiguous nature and to make the data exchanges more semantically robust. Ontology is utilized in this study to facilitate construction-oriented QTO.

There are many representation languages used to construct ontologies, such as Knowledge Interchange Format (KIF), Resource Description Framework (RDF), and Web Ontology Language (OWL). The semantics of the information in an ontology depend on the representation languages, as each language raises its own semantic restrictions (Martinez-Cruz et al., 2011). Semantics herein refers to the meaning or context of the information. In the early 2000s, ontology was implemented by object-oriented property-value representation that could declare classes and their properties (Ma et al., 2015). In this sense, ontology with object-oriented property-value representation is similar to object-oriented modelling. In contrast, RDF/OWL-based ontology, as described above, provides stronger expressive power. It allows users to explicitly specify far more information about classes and properties and to define a set of constraints and axioms held among concepts, relationships and individuals. These constraints and axioms facilitate machine understanding of the information (Venugopal et al., 2012). The RDF model represents a metadata data model with its schema, namely: "RDFS", which is defined by a set of terms with specifiable meanings. Unlike markup languages such as Extensible Markup Language (XML), where semantics are implicitly expressed, RDF makes the semantics of information explicit (Zhong et al., 2015). As such, RDF-based ontology with its explicit formality increases semantic awareness of computer applications. Semantic awareness herein is defined as the ability of computer applications to interpret and represent the meaning of the information. Additionally, the RDF model is a graph-based data model that represents

information in a directed and labelled graph data format (Powers, 2003); it allows structured and semi-structured data to be merged and exposed, as well as for data to be shared across different applications (W3C Semantic Web, 2015). Compared with object-oriented property-value representations, RDF-based ontology is more flexible and more easily extended, and thus is more suitable for representation of domain knowledge. In this research, RDF/OWL is employed to implement ontology and provide domain semantics in reference to domain terms, including their properties and interrelationships, as well as ontology reasoning. Simple Protocol and RDF Query Language (SPARQL) is a semantic graph query language designed to query RDF (DuCharme, 2011). Due to that semantic query on large ontologies by means of graph query languages (e.g., SPARQL) is less complex than the use of complete reasoning rules and ontology reasoning engines (Beetz et al., 2009), it is more straightforward for construction practitioners to formulate SPARQL queries on BIM models in comparison with ontology reasoning rule formulations. In this sense, an RDF-based ontology in conjunction with SPARQL would boost the information extraction efficiency.

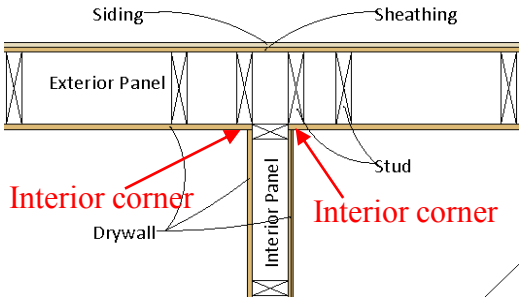
3.3.2 Construction-oriented QTO

Construction-oriented QTO produces quantities in proper units of measure which are taken off for construction activities based on activity definition and detailed specifications of construction methods and materials. As described previously, extracting building information implicitly represented in a BIM model is difficult, while retrieving building information without semantic domain terms and their interrelationships is inefficient. Owing to this reality, a challenge confronting construction practitioners is how to retrieve construction-oriented QTO information from a BIM design model. In this respect, an ontology can be utilized to enhance the BIM model by defining distinguishable domain terms or classes to represent features of this type. On the

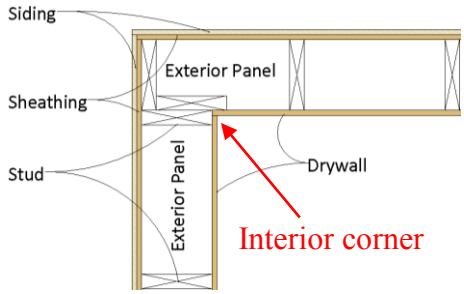
premise of some analysis such as topological analysis and ontology reasoning on a given BIM model, those implicit design features can be detected and then explicitly stored into an ontology-enhanced BIM model. As the building product ontology is formalized from the specific building domain and the construction practice, it enables construction practitioners to semantically query the ontology-enhanced BIM model using their domain vocabularies in order to retrieve the construction-oriented QTO information efficiently without the need to understand the technical structure of the underlying complex BIM schema. This section uses specific examples to illustrate these problems associated with construction-oriented QTO in the domain of light-frame building construction and how to address those problems using the proposed ontology-based approach.

Construction projects are completed by various builders/sub-contractors/trades-people. Builders and sub-contractors are coordinated to work together on different work packages based on their expertise. Builders and sub-contractors thus only need to retrieve quantity information concerning their specific work packages, rather than retrieving all the building information embedded in the BIM design model. For instance, framing subcontractors or carpenters are to frame walls and install blocking in wall frames; finishing carpenters are to install the interior trim, interior doors, windows/door casing, railing, and other interior elements; stucco subcontractors are to paint the building exterior; and drywallers are to install and finish drywall sheeting for residential building projects. In order to take off the quantity of the drywallers' work, for example, determining the surface area of interior walls is important. This information can be easily extracted from the BIM model; however, the drywallers also need to tape and finish drywall sheets and wall corners, which takes time and incurs cost in construction. Consequently, drywallers need to have a clear understanding of how the walls intersect.

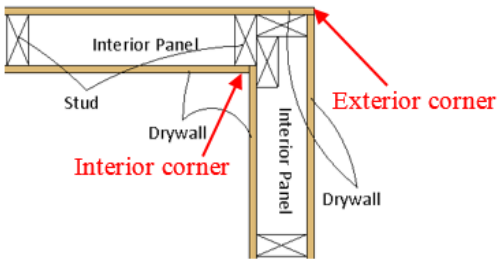
Moreover, interior corners are usually taped using paper corner bead, whereas metal corner bead is used for exterior corners (see Figure 3.1). Drywallers thus must know not only the number of general wall connections but also detailed wall connection information (e.g., L-connection, T-connection, Ext/Int-connection and Ext/Ext-connection) in order to obtain the correct quantity of metal corner bead. Similarly, estimators require this information in order to take off work packages based on detailed construction specifications during the detailed estimating process. For example, as shown in Table 1, cost item "092915100411" described as "*Accessories, gypsum board, corner bead, galvanized steel, 32 mm × 23 mm × 2,450 mm*" is used for pricing metal corner bead. Given these complexities and due to the fact that detailed building component intersections are not represented explicitly, it is not possible to retrieve intersection quantities by merely applying existing BIM-based quantity routines and methods as proposed in previous research such as Nepal et al. (2012; 2013).



a. T-connection (Exterior Wall & Interior Wall)



b. L-connection (Exterior Wall & Exterior Wall)



c. L-connection (Interior Wall & Interior Wall)

Figure 3.1 Corner bead for T-connection and L-connection

With regard to wall framing, as summarized in Table 1, work packages for bearing walls are taken off by length of studs for line number “061110405167”, length of plates for line number “061110405106”, and number of window/door bucks for line number “061110400340” or “061110400170”. In order to obtain quantities for each aforementioned item, each type of wall frame subcomponent (see Figure 3.2) needs to be explicitly modelled with distinguishable entities. For example, common studs associated with structural walls need to be distinguished

from opening studs such as window buck, king studs, jack studs, rough sill, cripples, and header (see Figure 3.2) so as to derive the total length of studs for item “061110405167” and number of window/door bucks for item “061110400340” and item “061110400170”. However, wall frame subcomponents in a BIM design model are generally represented by the same kind of model entity. As described earlier, all studs are usually modelled with the modelling element, called “Structural Column”, and all plates are represented as “Structural Framing” in Autodesk Revit. Within the open BIM schema, IFC, all studs and plates are represented as “IFCMember”. Without domain semantic awareness, all the model elements for wall frame subcomponents are identified as the identical entity (e.g., IFCMember); hence, without understanding of the complex BIM schema and human intervention from BIM experts, it is not possible to filter the studs in non-structural walls and the opening studs including cripple studs, king studs and jack studs in structural walls so as to obtain the required QTO for common studs in structural walls (line number “061110405167”) from a BIM design model.

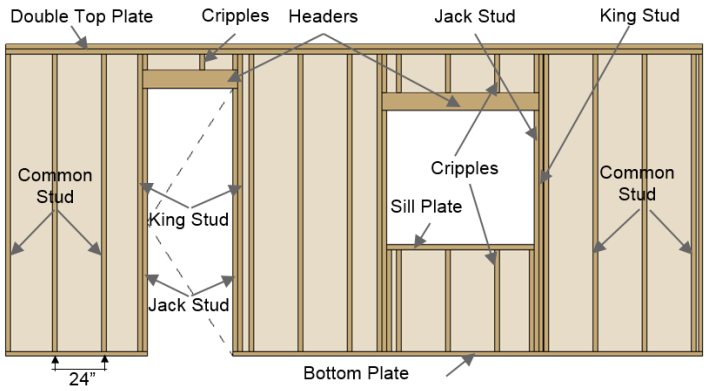


Figure 3.2 Stud-framed wall panel

Table 3.1 Unit price and production rate items from RSMeans Online (Gordian Group, 2015)

Line Number	Description	Crew	Daily Output	Unit
	Wall Connection			
092915100411	Accessories, **gypsum board**, **corner bead**, **galvanized steel**, 32 mm × 32 mm × 2,450 mm	1 Carp.	35	Ea.
	Bearing Wall Framing			
061110405167	Wall framing, ***studs***, ***50 mm × 152 mm, 2,450 mm high wall, pneumatic nailed***	2 Carp.	365.76	m
061110405106	Wall framing, ***plates***, ***treated, 50 mm × 152 mm, pneumatic nailed***	2 Carp.	274.32	m
061110405162	Wall framing, walls, for ***second story and above***, add extra labour	-	-	m
061110400340	Wall framing, ***window buck***, king studs, jack studs, rough sill, cripples, header and accessories, ***50 mm × 152 mm wall, 914 mm wide, 2,450 mm high***	1 Carp.	24	Ea.
061110400170	Wall framing, ***door buck***, king studs, jack studs, header and accessories, ***50 mm × 152 mm wall, 1,828 mm wide, 2,450 mm high***	1 Carp.	32	Ea.
	Non-bearing Wall Framing			
061110260180	Wood framing, ***partitions***, standard & better lumber, ***50 mm × 102 mm studs, 305 mm O.C., 2450 mm high***, ***includes single bottom plate and double top plate***, excludes waste	2 Carp.	24.38	m
061110261500	Wood framing, ***partitions***, ***for horizontal blocking***, 50 mm × 152 mm, add	2 Carp.	182.88	m
061110261702	Wood framing, ***partitions***, ***for headers for openings***, material only, add	-	-	m^3
061110261600	Wood framing, ***partitions, for openings***, add extra labour	2 Carp.	76.20	m

Non-bearing wall framing is taken off by length of wall as for line number "061110260180" characterized by detailed framing features such as "*50 mm × 102 mm studs, 305 mm On Centre, 2,450 mm high, includes single bottom plate and double top plate*". For this item, framing features, such as wall structural usage, wall height, its stud size, stud spacing, and having double top plate or single top plate, need to be detected by the take-off system in order to obtain the non-bearing wall length having this kind of framing feature. By explicitly modelling stud, plate and their interrelationship with walls as required by exterior wall framing, stud size can be identified. Stud spacing and having double top plate or single top plate are other features which need to be modelled explicitly as wall properties in order to take off the work package for line number "061110260180". Hence, the domain terms in light-frame building industry, such as king stud,

jack stud, and cripple stud, need to be formalized into the proposed ontology in order to address the lack of domain semantics in wall frame modelling and to align BIM design models with construction-oriented QTO.

3.4 Ontology-based semantic QTO approach

This study proposes an ontology-based semantic approach to construction-oriented QTO from a BIM design model. Ontology is employed in the study in order to enhance BIM models in terms of domain semantics, including: (1) domain terms, (2) properties, and (3) interrelationships. Domain terms such as Stud and Plate in the light-frame building industry and various wall connections are generalized into the product ontology. Their interrelationships and properties are defined explicitly in the ontology, providing the semantic foundation to the building quantity information retrieval application, as well as rich domain vocabularies, with which construction practitioners are familiar. This allows for construction practitioners to semantically query a BIM design model for explicit and implicit BIM data using their domain vocabularies without the need of understanding the technical structure of the underlying complex BIM schema. It addresses the challenges described previously with respect to construction-oriented QTO, and enables estimators or field contractor/sub-contractors to obtain QTO for construction work packages in a more efficient manner.

Figure 3.3 shows as overview of the proposed methodology. As depicted in the figure, a construction-oriented product ontology is developed by formalizing domain terms, their interrelationships, and properties in the light-frame building industry. With the exception that building terms in the existing BIM model, such as IfcWall, and IfcSlab, some other terms including stud, plate, L-connection, T-connection and the forth are added into the product ontology. It is noteworthy that construction-oriented product ontology not only contains

formalized terms from domain knowledge, but also includes specific BIM data. In order to populate this product ontology (i.e., ontology terms) with specific building information (i.e., ontology individuals), the terms in the formalized product ontology are first mapped with the BIM modelling elements within a building modelling tool (e.g., Revit, Tekla) or vendor-independent platform (e.g., IFC). Then, the BIM design model is analyzed against ontology terms using "BIM data parser" in order to extract specific BIM data, whereas "ontology individual generator" transforms extracted BIM data into the product ontology. Ontology reasoning enabled by "ontology reasoner" can be further applied in order to infer new information or facts on the basis of explicit BIM data. Finally, an ontology-enhanced BIM model is generated for applications in construction planning. Semantic query can be formulated against "semantic query processor" in order to semantically query the ontology-augmented BIM model, thereby obtaining the required construction-oriented QTO information.

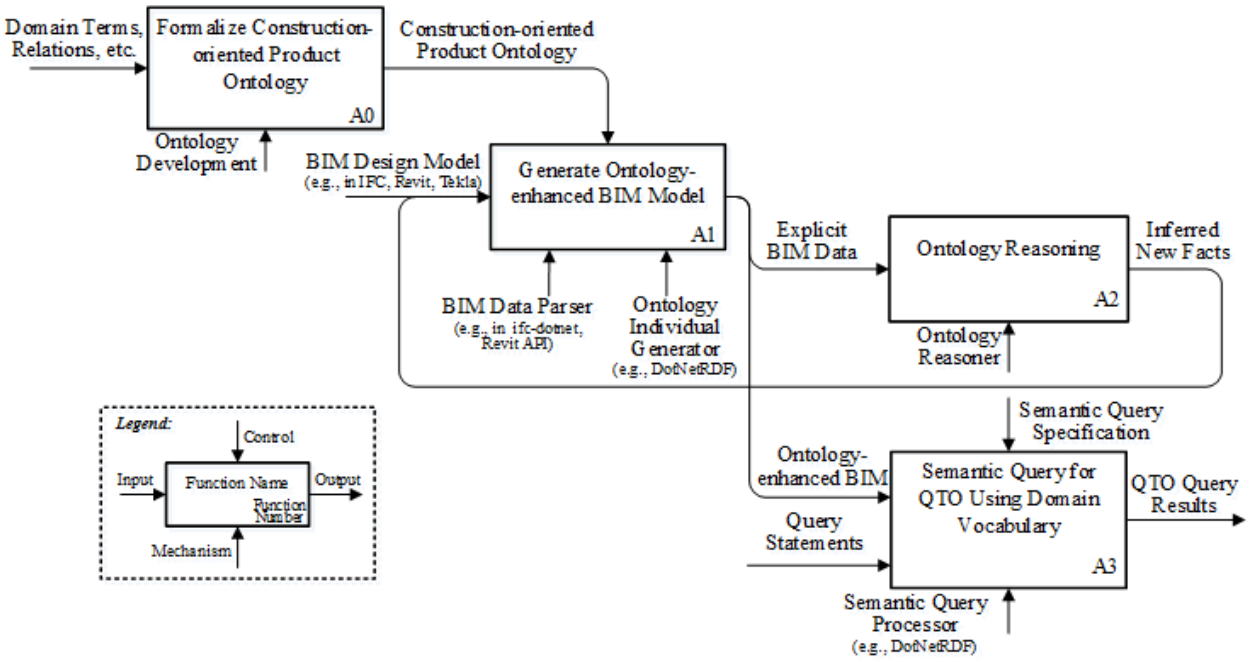


Figure 3.3 Overview of proposed methodology

3.5 Prototype application

3.5.1 Overview

The system architecture for implementing the proposed semantic QTO approach is presented in Figure 3.4. Generally, it includes a BIM design model, a BIM data parsing tool/library, an ontology editor, an ontology reasoner, and an RDF tool. The BIM design model is developed using the BIM authoring tool, Autodesk Revit, which gives end-users modelling flexibility (e.g., its built-in functions such as Family Editor) and supports API to enable third-party add-on programming. The ontology in this research is established using Protégé 4.3, a free, open-source ontology editor (Protégé, 2014). BIM data is parsed from an Autodesk Revit BIM model using Revit API, and dotNetRDF, an open source .Net Library for RDF (Vesse et al., 2014), is employed to build the ontology-augmented BIM model by populating the formalized ontology with extracted BIM information. A default ontology reasoner in Protégé 4.3 is employed to infer new facts (i.e., implicit design features) in the ontology-augmented BIM model based upon explicit BIM data. SPARQL (which is supported by dotNetRDF) is used to query the ontology-augmented BIM model in order to obtain construction-oriented QTO information. All the system components are integrated through Autodesk Revit API in C# language, and the prototyped system is programmed as an add-on for Autodesk Revit. A detailed explanation of the methodology is presented in the following sections.

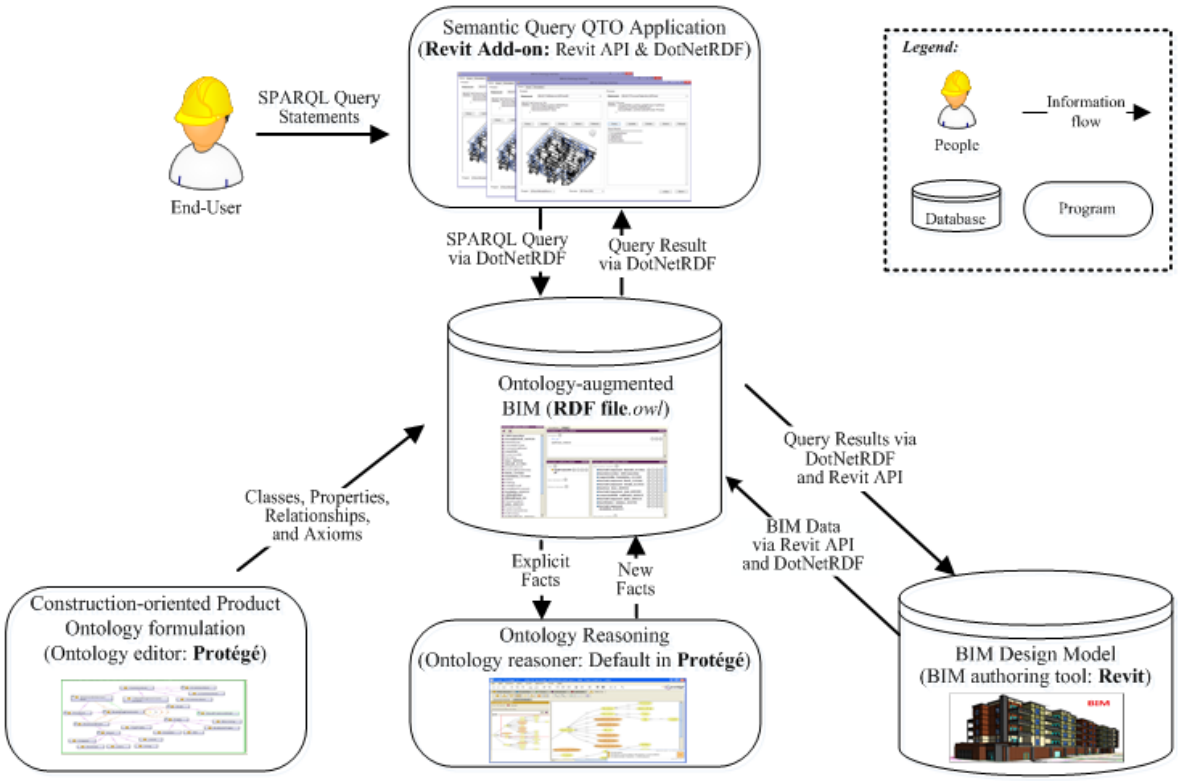


Figure 3.4 System architecture

3.5.2 Construction-oriented product ontology

Construction-oriented product ontology in this research is intended to allow construction practitioners (particularly, trades personnel) and QTO professionals to take off quantities for construction work packages and to enable effective workface planning. This ontology is established in order to align a BIM design model with construction process oriented QTO and to enable semantic querying in the domain of light-frame building industry. As described above, this ontology augments the BIM model by adding light-frame building terms, including their properties and interrelationships, and implicit design feature terms such as "L-connection", which are relevant to construction practitioners. Figure 3.5 presents part of the construction-oriented product ontology formalized within Protégé 4.3. It is worth mentioning that term, property, and interrelationship are represented by Class, Data Property, and Object Property,

respectively, in Protégé 4.3. As shown in Figure 3.5, “Product” is the root term in the ontology, and “BuildingElement”, “ElementPart”, and “ElementIntersection” are inherited from “Product”. Basically, inheriting indicates an *Is-a* relationship which means that each term (“BuildingElement”, “ElementPart”, and “ElementIntersection”) is a “Product”. Furthermore, some building terms such as “Wall”, “Door”, and “Window”, which are contained in the existing BIM schema, are defined under “BuildingElement”. “Plate”, “Stud”, “Drywall”, etc. are defined under “ElementPart”. An intersection among building elements is described by “ElementIntersection”. A few object properties (interrelationships) are defined in order to describe relationships among those concepts. For example, “hasSubComponent” is an object property to describe the relationship between “BuildingElement” and “ElementPart”, whereas “hasOpening” has to do with the interrelationships among “Wall”, “Window” and “Door”. Besides this, “hasIntersection” describes the relationship between “BuildingElement” and “ElementIntersection”. Various connection types are further detailed by using “LConnection”, and “TConnection”, and these terms inherit “ElementIntersection”. It is notable that sub-terms in the ontology inherit both properties and interrelationships of their base terms. This entails, for example, that “hasIntersection” also describes the relationship between “TConnection” and “Wall” due to the fact that “TConnection” and “Wall” are inherited from “ElementIntersection” and “BuildingElement”, respectively.

As shown in Table 1, construction-oriented QTO is taken off on the basis of product design features. All features need to be distinguishable in the QTO system. In addition to terms and their interrelationships, some term properties that characterize the building terms are defined in the proposed product ontology as depicted in Figure 3.5. For instance, *Stud* has a type property describing its stud size (i.e., “50×102” or “50×152”), while stud spacing and wall function (e.g.,

bearing/non-bearing, IsAcoustic, IsExterior, IsFireRated, IsPartition, and so forth) are also described explicitly as Wall properties in order to quantify the work packages such as framing partition wall. "Level" is another property defined for "BuildingElement", since building elements, such as walls on the second level of a building and above, demand extra labour time and cost in comparison with first-level walls, as illustrated in Table 1, and QTO for wall framing must be taken off according to the floor level. It is noteworthy that "WorkZone" is defined as a property of "BuildingElement" so as to provide construction practitioners with the flexibility of taking off quantities of work packages based on the horizontal construction zone when performing location-based QTO on large construction projects.

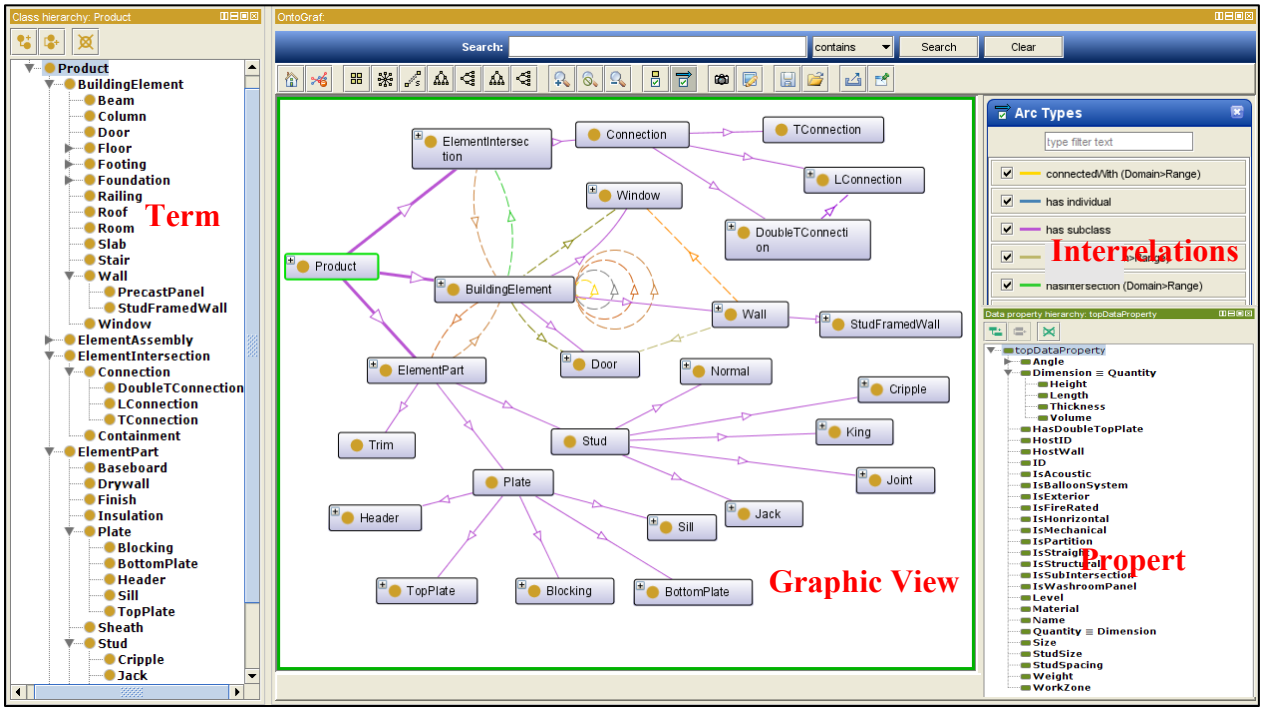


Figure 3.5 Construction-oriented product ontology

3.5.3 Ontology-enhanced BIM model generation

To enable semantic QTO search, a BIM design model is augmented by the proposed construction-oriented ontology. Building information needs to be extracted from the BIM design

model and inputted to the ontology in order to obtain the ontology-augmented BIM model. The ontology in this research is established using Protégé 4.3 (a free, open-source ontology editor supporting RDF/XML files) and saved into an RDF file. BIM data in this study is extracted from a BIM design model using Revit API and inputted into the RDF file using DotNetRDF, resulting in the ontology-augmented BIM model.

3.5.3.1 Parsing Revit BIM data

Due to the fact that ontology augments the BIM model with domain terms and their properties and interrelationships, which are not defined explicitly in a given BIM model, there are two kinds of ontology terms: (1) basic building terms already defined in the BIM design model, and (2) extended domain terms which are missing in the BIM design model. Modelling elements in Autodesk Revit are mapped with those terms in the construction-oriented product ontology, while BIM data, including explicit and implicit data, is extracted from existing design-oriented BIM models to populate the construction-oriented ontology. It is noteworthy that the majority of implicit construction-oriented BIM data (e.g., topological information) are derived on the basis of the explicit design-oriented BIM data by using algorithms, whereas the other complementary portion of implicit BIM data is inferred from ontology reasoning. Explicit BIM data is directly extracted from existing design-oriented BIM models. In the following sub-section, two types of BIM data extraction, explicit and implicit, are described in detail.

3.5.3.1.1 Extracting Explicit BIM data

Generally, building product information in the BIM model can be categorized into three groups: geometric, spatial/topological, and enriched functional. Geometric information refers to vertices, edges, and faces of building components, while spatial/topological information elaborates on their location and spatial relationships. Enriched functional information refers to additional

attributes or properties describing building components such as host information. Spatial information in Autodesk Revit is described by "Location" as shown in Figure 3.6. The exact location information of building elements can be extracted according to the class diagram in Figure 3.6. Its geometric information such as vertices, edges, and faces is described by "GeometryElement" and can be retrieved using the "Element.get_Geometry()" function. However, topological information is not represented explicitly in the Revit BIM model, and therefore must be derived by conducting topological analysis based on related spatial and geometric information. Enriched functional information is embedded into the BIM model as properties of parametric building objects. Since Autodesk Revit is a family-based BIM solution, where all building elements are grouped into "families" (Autodesk Revit, 2015), properties are categorized into two groups: type property and instance property. *Type property* is defined at the family type level and shared by a group of building elements with the same type. In contrast, *instance property* pertains to individual building elements. It is worth noting that Autodesk Revit has two kinds of family: system family and loadable family. Basic building elements such as Wall (see Figure 3.6) are *system families* that are predefined in Revit, whereas other building elements such as doors and windows are *loadable families* represented as "FamilyInstance" (see Figure 3.6). As shown in Figure 3.6, type property is defined as FamilySymbol and WallType classes for FamilyInstance (e.g., windows and doors) and Wall respectively, and all type properties are then attached to individual elements as one common property. In contrast, each individual instance property is attached as one property to individual elements. To retrieve enriched functional information, two functions, "element.get_Parameter(string)" and "elementType.get_Parameter (string)", can be utilized with the property name as input parameters, respectively. Material information is described by "Materials" and can be retrieved

using “element.GetMaterialIds()”, “element.GetMaterialArea()”, and “element.GetMaterialVolumn()”.

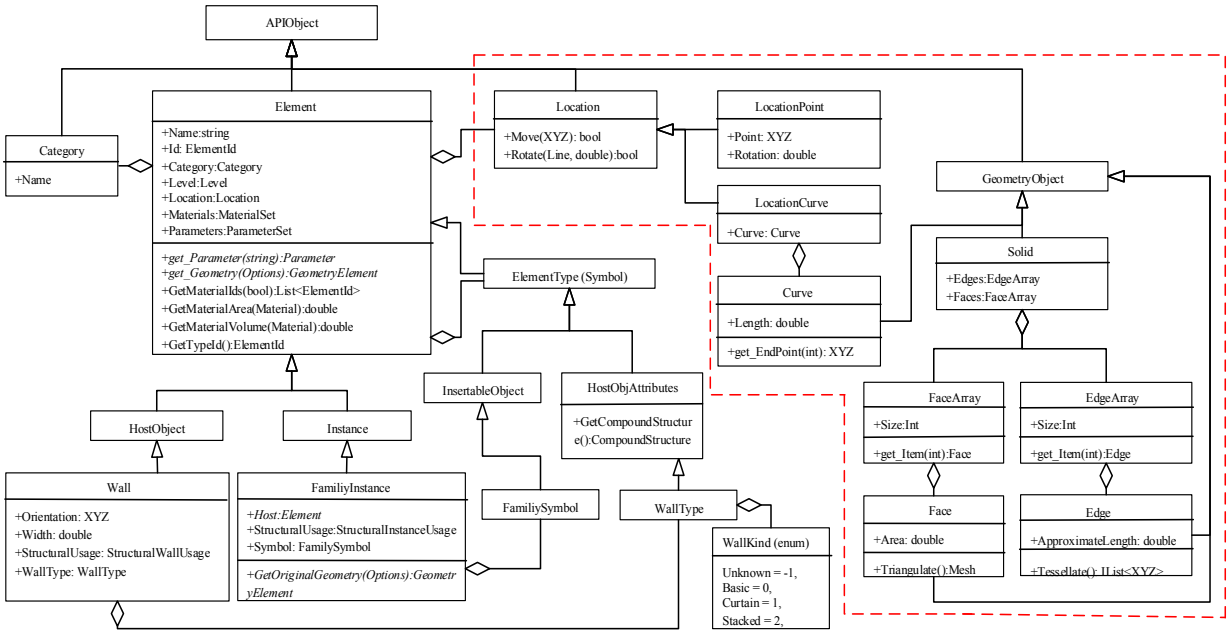


Figure 3.6 Autodesk Revit building elements in UML (Autodesk Ltd., 2014)

Taking a wood-framed wall (see Figure 3.7) as an example, wall, stud, plate, and opening information is vital for construction-oriented QTO. In Revit, walls are instances of “Wall” class, whereas windows, studs, and plates are instances of “FamilyInstance” class. It is “Category” that further identifies a building object’s type (e.g., Window, Structural Column, and Structural Framing). As denoted in Figure 3.6, Wall is defined explicitly as a subclass of “Element” in the Revit class diagram. Some information such as “Id”, “Name”, “Level”, “Material”, and geometric information is defined as instance properties, while wall layer information is given as its WallType property. Nevertheless, there is no modelling element named “Stud” or “Plate” in Revit. Structural Column and Structural Framing, each of which is a type of “FamilyInstance”, are alternative elements that can be used to model “Stud” and “Plate”. Similarly, its geometric and spatial information is described using “Location” and “GeometryObject”. Its type property is

defined in “FamilySymbol”. All this information can be extracted by referring to the class diagram shown in Figure 3.6. Herein, it needs to be noted that the host information for doors and windows is retrieved in a different manner from studs. The host relationship between walls and windows/doors is stored in the “Host” property of FamilyInstance due to the fact that walls are the valid host elements for windows/doors, and Revit saves this intelligent relationship to the BIM model during the modelling process. On the contrary, structural columns/frames used to model studs and plates do not have host information in the Revit internal data schema because they are stand-alone building elements that may not be hosted by other building elements. In this case, a property should be defined by the user in order to save this hosting relationship between walls and studs. Note that a suite of commercial Revit add-ons, Metal Wood Framer (MWF) (StrucSoft Solutions, 2015), is employed to frame wall panels in this research. Hosting information of studs is saved into its “BIMSF_Container” property by this commercial program. In addition, this commercial program defines a property “BIMSF_Description” for Structural Column and Structural Framing to store its function information such as King Stud and Jack Stud. Stud hosting information and function information can be retrieved using “Element.get_Parameter (BIMSF_Container/BIMSF_Description)” (see Figure 3.7). It should be noted that while the hosting information can be retrieved from “Host” and “BIMSF_Container” properties, the implicit inverse relationship (e.g., Wall.HasDoor and Wall.HasSubComponent as shown in Figure 3.7) does not exist in Autodesk Revit BIM. In this regard, the proposed ontology enhances the interrelationships among terms by explicitly defining them and specifying the nature of the relationships (e.g., Inverse Of, Equivalent To, and Sub Property Of) among the domain term interrelationships. Ontology in turn can create new information by reasoning/inferring about the explicit information. More specifically, ontology reasoning can not

only confirm and check “known knowns”, but also shed light on some “known unknowns”. For example, “hasDoor” as depicted in Figure 3.8 is declared as a sub-property of “hasOpening”, whereas “hasOpening” is the inverse of “hostedBy” in the proposed ontology in Protégé 4.3. When the explicit hosting information of Door A (e.g., hostedBy) is extracted from the BIM design model and added to the ontology, ontology reasoning infers its implicit inverse relationship and deduces the fact that Wall A is hosting Door A as shown in Figure 3.7. The inferred fact is then saved explicitly in the ontology, which boosts the efficiency of information extraction.

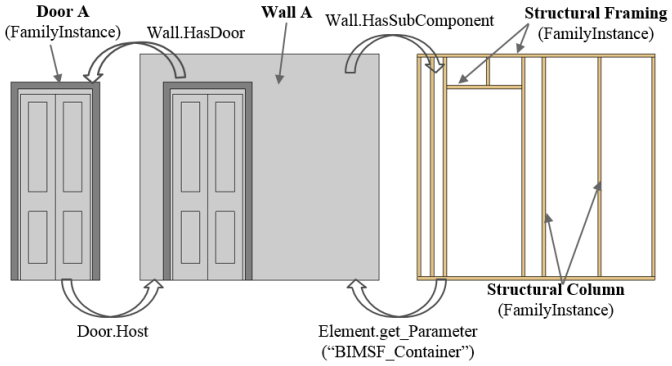


Figure 3.7 Host relationship and its inverse relationship

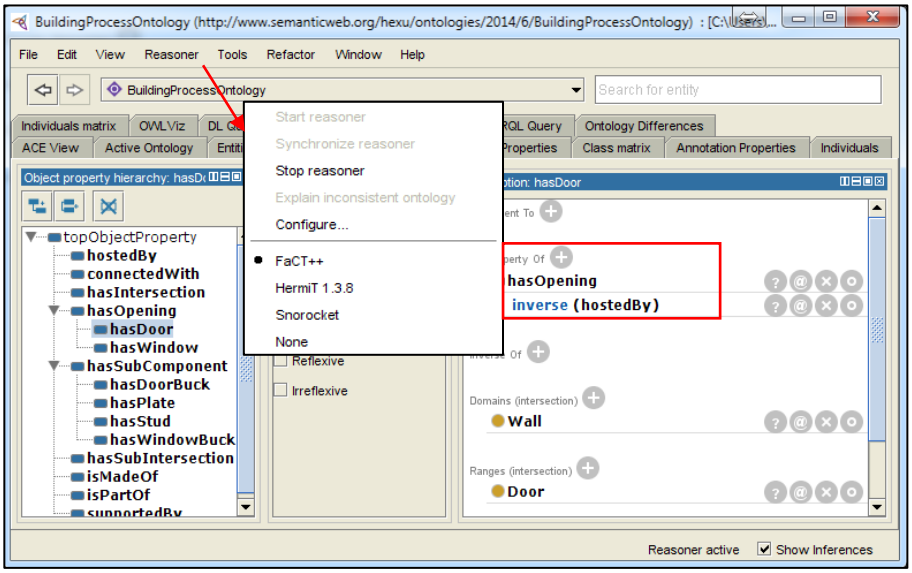


Figure 3.8 Defining term interrelationship and ontology reasoning within Protégé 4.3

3.5.3.1.2 Extracting derived BIM data

A BIM model is an assembly of building objects, but some building element intersections are not explicitly modelled in the BIM design model. Construction practitioners, however, specifically need to obtain detailed intersection information. In terms of domain terms such as element intersections defined implicitly in the existing BIM schema, further analysis is required in order to derive this information, after which the analysis results can be stored explicitly in an ontology-augmented BIM model in order to facilitate the building information extraction. This section takes wall connection and stud spacing as examples to illustrate in detail the extraction of implicit BIM data.

Wall connection, one type of intersection, is crucial to construction practitioners in determining the quantity of intersection corner bead, as described above. Commonly seen wall connections in building projects such as *T-connection, L-connection and Double-T-connection*, as shown in Figure 3.10, are identified in this study. An algorithm is developed by which to detect those wall-to-wall connections based on geometric information of faces, edges, and points (as depicted in Figure 3.9). Basically, the algorithm first takes every combination of two walls in order to check whether they are connected by sharing one face. More specifically, a connection relationship between two walls is derived by checking whether two faces (each from one building component) overlap on one contact area denoted by Eq. (1) and whether their normal vectors defined as pointing outward from the solid object are in opposite direction as Eq. (2). The location and geometric information are extracted by referring to “Location” and “GeometryElement” in the class diagram shown in Figure 3.6. The algorithm then takes one vertical face (e.g., start face, end face, left face and right face in Figure 3.9) from each component; given that F_1 is from component 1 and F_2 is from component 2, normal vectors of F_1 and F_2 are checked to determine

whether or not they are in opposite directions using Eq. (2). This is done in order to exclude the containment relationship. If the vectors are opposite ($e.g. V_1 \cdot V_2 = 1 \times -1 + 0 \times 0 + 0 \times 0 = -1$), then all points (including vertices of building components and middle points of edges) of F_1 are checked to determine whether or not they lie in F_2. If there are more than 3 points that do not lie in a straight line (see Figure 3.10), the two elements are considered to be connected. All points of F_2 are also checked against F_1 in order to consider the case in which F_2 is inside of F_1. Subsequently, the connection type (e.g., T-connection, L-connection) of the detected connection is identified by checking whether the end faces (e.g., start face and end face) of these two walls overlap by sharing one contact area or one edge as denoted by Eq. (3). Also, more detailed connection information such as connection angle and wall joining end, as shown in Figure 3.10 and Figure 3.11, is analyzed. Connection angle is crucial information in determining quantity of corner beads for drywallers. As shown in Figure 3.10a, when two walls adjoin at 180°, an L-connection is made with a primary angle of 180°; however, no corner bead needs to be placed at this connection. The angles in Figure 3.10 are thus derived from the direction vector of walls and are stored explicitly in the ontology-enhanced BIM model. Afterward, all the detected connections between each two walls are checked to determine whether or not they share the same joining wall and its joining end in order to derive wall connections with multiple walls joining together as shown in Figure 3.10c and Figure 3.10d. The T-connection Figure 3.10c consists of two L-connections that share the same joining wall, whereas the T-connection in Figure 3.10d is made of three L-connections. Once all wall-to-wall connections are detected, new connection entities are then created in the ontology-augmented BIM model, and detailed properties, such as connection angle and wall joining end, are populated based on the analysis results. Also, “hasIntersection” relationships are established between wall entities and those

connection entities in the ontology-augmented BIM model. It is worth mentioning that as described above, wall connections with multiple walls consist of several wall connections, each between a pair of two walls, as shown in Figure 3.10c and Figure 3.10d. This containment relationship between connections is detected and stored in the "hasSubIntersection" object property as shown in Figure 3.11. In summary, the governing equations for deriving wall connections are given as Eq. (1) to Eq. (3).

$$F_1^\circ \cap F_2^\circ \neq \emptyset \quad \text{Eq. (1)}$$

$$\mathrm{V}_1 \cdot \mathrm{V}_2 = -1 \quad \text{Eq. (2)}$$

$$F_{end,1} \cap F_{end,2} \neq \emptyset \quad \text{Eq. (3)}$$

where F_1 and F_2 represent faces from two solid geometries, V_1 and V_2 denote their respective normal vectors, and $F_{end,1}$ and $F_{end,2}$ represent the start face or end face from two solid geometries.

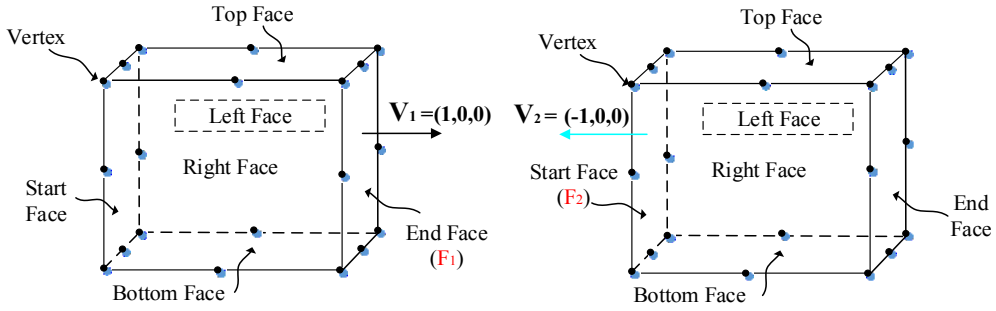
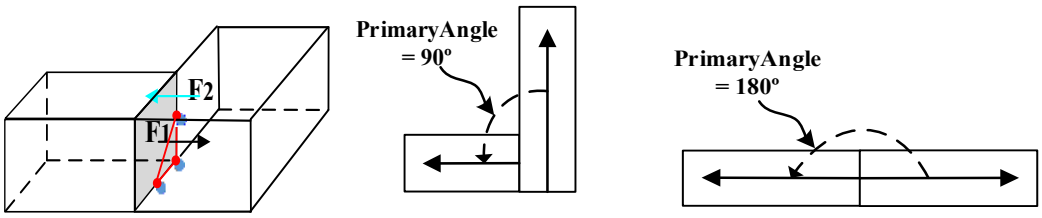
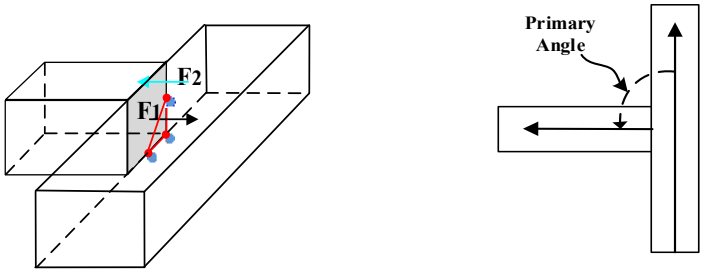


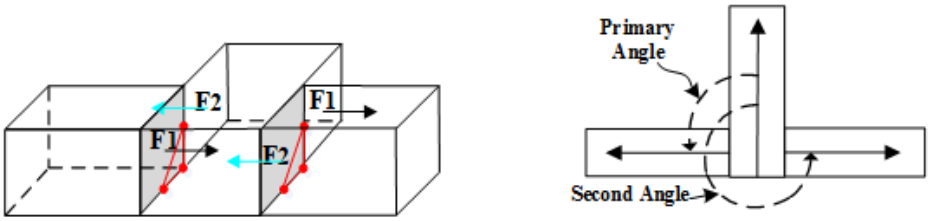
Figure 3.9 Geometric information of building component



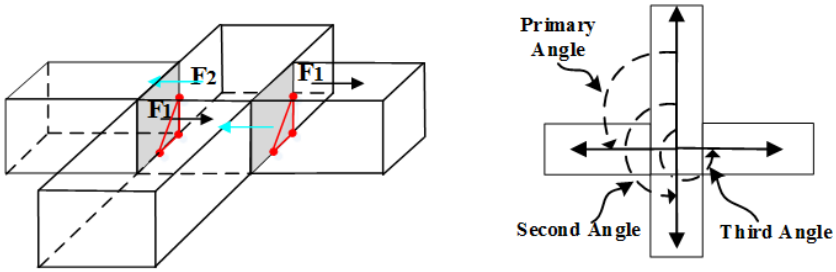
a. L-connection between two walls



b. T-connection between two walls



c. T-connection among three walls (two L-connection)



d. Double-T-connection among four walls (three L-connection)

Figure 3.10 Various wall connections

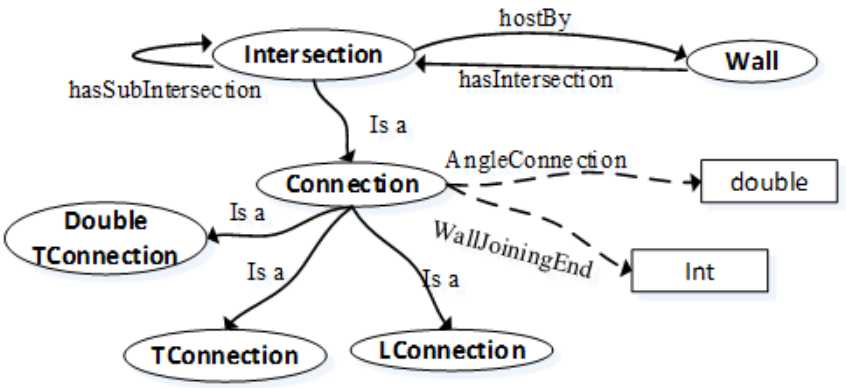


Figure 3.11 Properties and interrelationships of connection

The proposed algorithm runs through every possible combination of two walls in order to check whether they are connected or not. Each wall is checked against other walls, and the algorithm is implemented through one iteration loop with one nested iteration loop. As a result, its time complexity is O (N^2), where N represents the number of walls. Figure 3.12 presents the measured performance result of the proposed geometric algorithm, with the horizontal axis representing the number of walls and the vertical axis showing the elapsed milliseconds of executing the proposed algorithm in the prototype system, respectively. As shown in Figure 3.12, the elapsed milliseconds increase in a square power relation to the number of walls, which in turn demonstrates that the algorithm is coded in a reliable manner in the prototyped system.

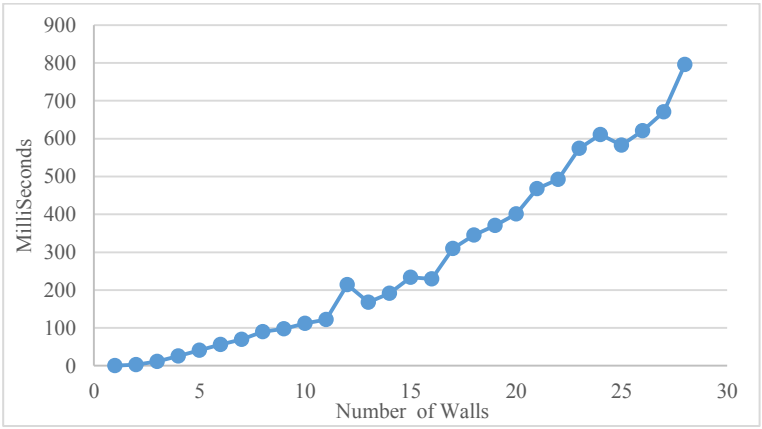


Figure 3.12 Measured performance result of “Connection” algorithm

Stud spacing denotes the maximum centre-to-centre distance between adjacent studs in a light-frame wall. To obtain stud spacing information, opening studs, such as king studs and jack studs, need to be filtered out so that only location information pertaining to common and cripple studs is retrieved. Then, all distances between adjacent studs are calculated in Revit API. The maximum of all the calculated Euclidean distances defines the stud spacing denoted as Eq. (4). Generally, three steps are executed in the proposed algorithm. To begin, the common and cripple studs are sorted in the ascending order of their distances to the wall origin; then, distances between adjacent studs are calculated through an iteration loop; finally, the stud spacing is determined as the maximum value of all calculated distances. The time complexities in these steps can be expressed as O (N * log (N)), O (N), and O (N), respectively. Here, N denotes the number of studs. Accordingly, the worst-case time complexity for calculating stud spacing is O (N * log (N)). Figure 3.13 presents the measured performance result of the proposed geometric algorithm. The horizontal axis represents the number of studs, whereas the vertical axis shows the elapsed milliseconds of executing the proposed algorithm in the prototype system. As shown in Figure 3.13, the elapsed milliseconds increase in relation to the number of studs.

$$\text{Stud Spacing} = \max_{1 \le j \le m-1} \left(\sqrt{\sum_{i=1}^{2} \left(p_{i,j} - p_{i,j+1} \right)^2} \right) \quad \text{Eq. (4)}$$

where $\{p_{i,j}\}_{i=1,2}$ are the coordinates of location point of the j^{th} stud, and m is the number of studs, including common and cripple studs, in the wall.

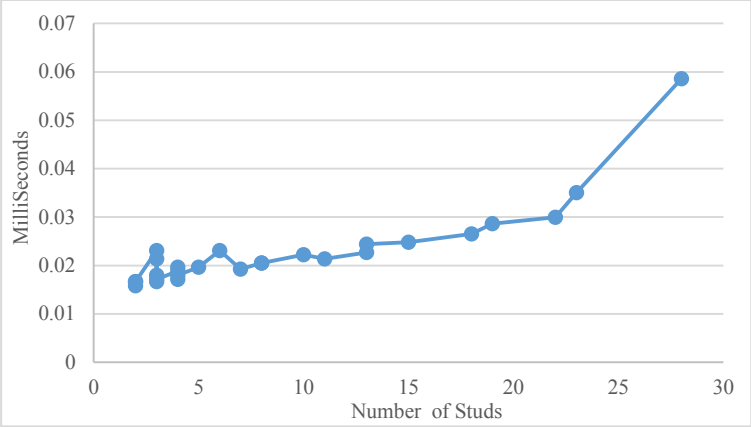


Figure 3.13 Measured performance result of "GetStudSpacing" algorithm

3.5.3.2 Populating established ontology with Revit BIM data

Once BIM data (including explicit data and derived data) have been analyzed and extracted using Revit API, dotNetRDF is integrated with the Revit API in order to populate the established ontology with the extracted BIM data and generate an ontology-augmented BIM model in RDF. Basically, the construction-oriented product ontology (referring to ontology terms) is established in Protégé 4.3 and saved into a RDF file with an extension of owl. DotNetRDF provides functions such as *RDFGraph.Assert(new Triple (subject, predicate, object))* to write extracted BIM data into this file in order to generate ontology individuals. The partial detailed implementation for generating ontology individuals in C# is provided in Appendix C. It should be noted that the ontology-augmented BIM model keeps references of building objects in the Revit BIM model by recording identification numbers of corresponding building elements into ontology entities. In this way, the query results can be visualized in 3D in the original Revit BIM model, which enables construction practitioners to more clearly envisage the search results. With respect to implicit design features such as intersections, their host building elements in turn can be highlighted to enable feature visualization.

3.5.4 Semantic query

The purpose of construction-oriented QTO is to take off work packages on the basis of product design features and construction methods, construction-oriented QTO is thus product design feature-based QTO. In other words, construction practitioners obtain the QTO by (1) filtering the product design features based on descriptions of work packages and (2) performing take-off on applicable building elements or product design features. For instance, the work package "061110405167" for wall framing, as shown in Table 3.1 and Figure 3.14, is to take off the total length of studs hosted by walls that are resting on the 1st level of the building, made of 50 mm × 152 mm studs, with a height of 2,540 mm, and functions as structural walls. This work package only applies to studs with these particular design features, which are associated with unique unit price and production rate; various work packages can be defined in order to factor in the effect of design feature details on production rate and unit price in detailed construction planning. Nevertheless, it is a challenge for construction practitioners to manually look for applicable building elements and product design features, thereby obtaining the required QTO. A query-based approach allows construction practitioners to search through BIM design models for the desired information in a flexible, straightforward manner. This approach reduces the laborious manual work and human errors associated with looking for and performing take-off on relevant building elements and product design features. It thus provides a promising solution to retrieve construction-oriented QTO information. SPARQL, released by the W3C RDF Data Access Working Group (DuCharme, 2011), enables a semantic query using formalized domain vocabularies. In this study, it is employed to query the ontology-augmented BIM model in RDF in order to materialize construction-oriented QTO. Figure 3.14 presents one example of a SPARQL query for taking off the aforementioned wall framing work package. As shown in the

figure, a query using SPARQL is expressed as a collection of conditions in a "subject-predicate-object" triple structure, allowing construction practitioners to obtain its quantity information. A triple structure, as illustrated in Figure 3.14, includes three parts: the first is always the subject, while the predicate and the object are the second and third, respectively. Variables are indicated by a "?" prefix, and the prefix, "proOnto", herein refers to the established construction-oriented product ontology. It should be noted that users can formulate query statements by using richer vocabularies generalized in the construction-oriented ontology. Richer vocabularies are used in "predicate" and "object" in order to filter the design feature. For example, "proOnto:isPartOf" is a vocabulary to describe the relationship between "BuildingElement" and "BuildingPart", while "proOnto:Height" is a property of Wall in the ontology. Each triple represents one filtering condition for each design feature. For instance, "stud proOnto:StudSize "L2X6"^^xsd:string" defines that framing studs are "50 mm × 152 mm" (2″ × 6″) type stud, whereas "hostWall proOnto:IsStructural true." requires that the walls hosting studs should be structural walls. As a result, construction practitioners can obtain the construction-oriented QTO information by formulating the SPARQL query in a flexible, straightforward manner.

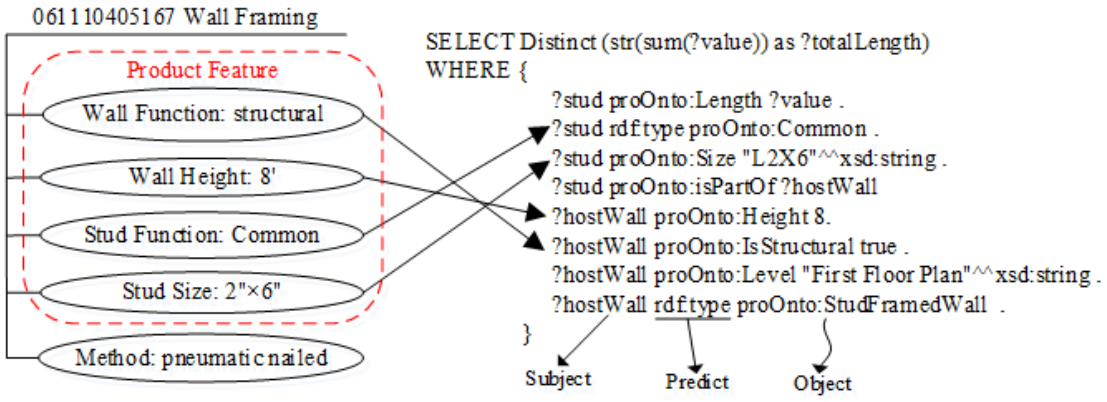


Figure 3.14 SPARQL query

This research implements the proposed approach as a Revit add-on. The dotNetRDF is adopted and integrated in the Revit API in order to allow construction practitioners to formulate queries on the QTO system user interface. Figure 3.15 presents the prototyped Graphic User Interface (GUI) where the SPARQL query can be inputted. The SPARQL query is executed through the dotNetRDF.Net library, from which query results can be retrieved. More importantly, query results can not only be shown in the developed GUI literally, but can also be transferred to Revit API in order to visualize query results within Autodesk Revit by highlighting corresponding building elements (see Figure 3.15). Each query statement, along with its item name that has been inputted within the GUI, can be saved and updated into a query database via corresponding buttons (e.g., Update and Delete) on the GUI. These query statements can be reused across different projects in order to expedite query formulations. In the case of new projects, construction practitioners are able to browse through the query database via a drop-down list and reuse the existing query statements. In future work, query statements in the database can be integrated with unit price and production rate databases in order to effectively support construction planners in cost estimation and scheduling.

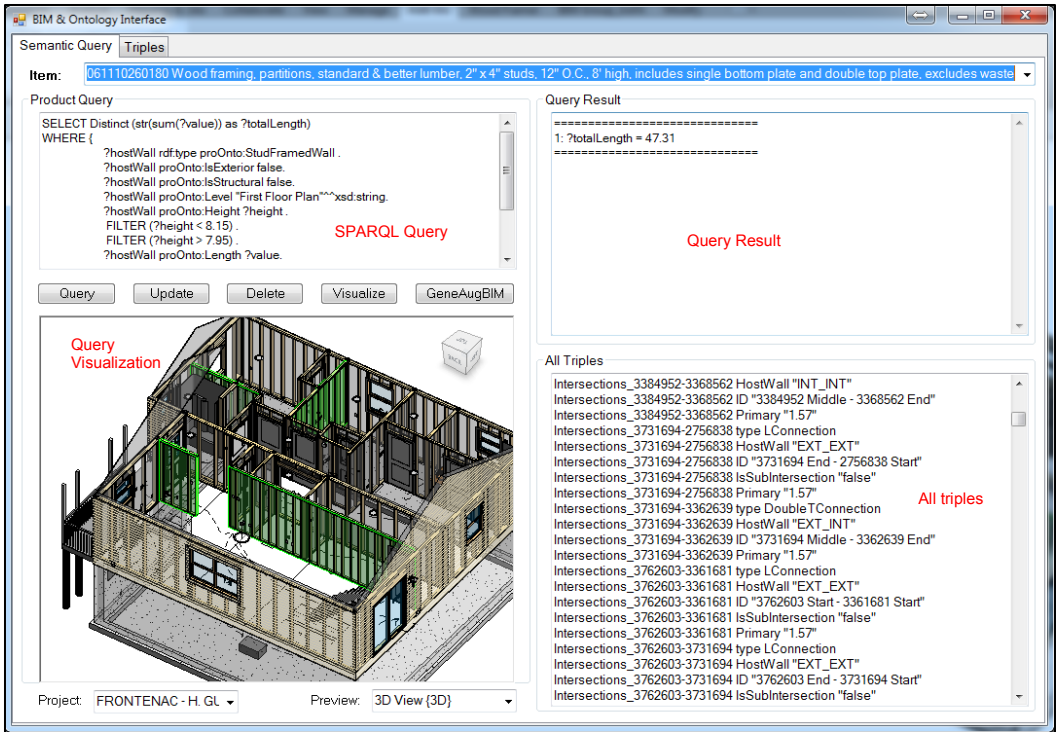
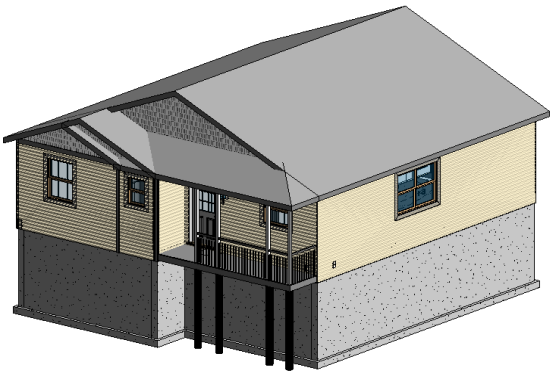


Figure 3.15 Semantic query through prototyped GUI of Revit add-on

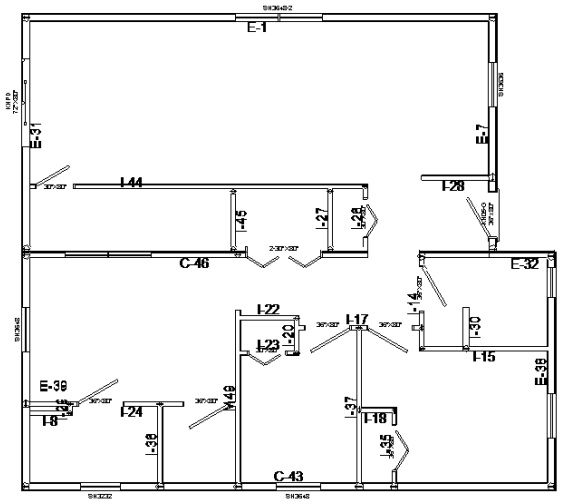
3.6 Validation

To validate the proposed methodology and the prototyped system, a wood-framed residential building, as shown in Figure 3.16, was chosen for case study. The building consists of two storeys and 34 wall panels, including 12 exterior walls and 22 interior walls. The building model was built in Autodesk Revit 2015. Due to the fact that Autodesk Revit, as a general BIM modelling tool, does not provide efficient functionalities for detailed construction framing, a suite of commercial Revit add-ons, Metal Wood Framer (MWF) (StrucSoft Solutions, 2015), was employed to frame the building components such as wall panels. The developed prototype system was then launched in Autodesk Revit to conduct construction-oriented QTO. It is noteworthy that the ontology-enhanced BIM model was first generated or updated by "GeneAugBIM" on the GUI as shown in Figure 3.15. It took only a few seconds for the prototype system to populate ontology individuals in the case project, since the DotNetRDF

library, an open source .Net Library for RDF, was integrated with Autodesk Revit API through programming. The substantial manual work involved in the generation of ontology individuals was thus eliminated. An ontology reasoner in Protégé 4.3 called "FaCT++" was then invoked to infer implicit design features (e.g., inverse relationships between ontology entities), as shown in Figure 3.8. Afterward, SPARQL queries were formulated in the GUI to take off work packages and visualize query results based on the ontology-enhanced BIM model.



a. 3D view



b. Plan view

Figure 3.16 The two-storey residential building of the case study

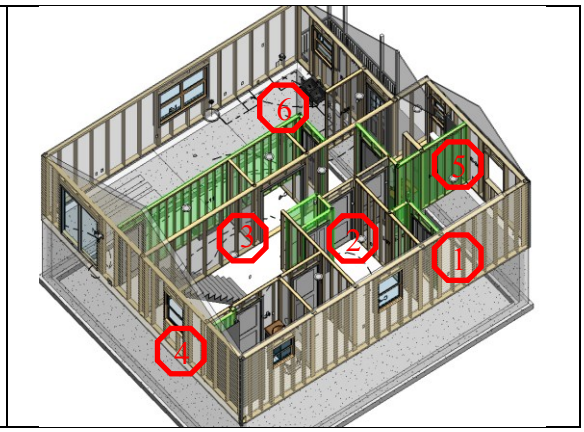
Four examples were utilized to test and validate the prototyped system with respect to new domain terms, new term interrelationships and new term properties. SPARQL queries for these

four examples are shown in Figure 3.17. New domain terms were tested by taking off exterior corners of L-connections for work package “092915100411”, listed in Table 1. Since wall connections were modelled explicitly in the ontology-enhanced BIM model, semantic query could be formulated to query this feature (see Figure 3.17b). Note that only exterior corners of L-connections with two interior walls are taped with metal bead. Hence, a triple “*?hostWall pronto:IsExteior false*” as shown in Figure 3.17a was added into the query statement in order to filter L-connections hosted by exterior walls. New term interrelationships were tested by taking off studs for structural wall framing “061110405167”, since taking off stud framing of structural wall requires explicit modelling of the hosting relationship between walls and studs. Wall framing “061110405167” is defined only for “50 mm × 152 mm” studs hosted by bearing walls, and this item is taken off by summarizing stud lengths. The SPARQL query language provides aggregate functions, such as SUM, MAX, ORDER BY, and GROUP BY, and these aggregate functions (e.g., *sum* as shown in Figure 3.17c) can be used in the query statement in order to take off applicable design features in the BIM model. Non-bearing wall framing for “061110260180” was taken off in order to test the new term property “StudSpacing” and “HasDoubleTopPlate”. As shown in Figure 3.17e, “*?hostWall proOnto:StudSpacing "12"^^xsd:string*” defines that the stud spacing of the walls is 305 mm (12″), whereas “*?hostWall proOnto:HasDoubleTopPlate true*” requires that the walls are framed with double top plates. The query results are tabulated in Table 3.2. A manual QTO was conducted to verify and validate the semantic QTO. The prototyped system provided the same QTO results as manual QTO, while significantly improving QTO efficiency in comparison with manual QTO. In addition, as demonstrated in Figure 3.15, the prototyped system is able to visualize the query results highlighted on the GUI, which can enhance communication among construction practitioners. Moreover, this

visualization feature allows the user to check quickly whether the required QTO has been done for all relevant building design features. Figure 3.17 shows the resulting design features in green for the four corresponding SPARQL queries. For instance, Figure 3.17b highlights all the walls hosting the required L-connections. As depicted in Figure 3.17d, studs for structural walls with 2,743 mm of height are not highlighted in green due to the fact that this height does not fall into the range specified in the corresponding query statement. Studs around the opening, such as king studs and jack studs, are also not coloured in the figure since they are filtered out by the query statement as expected. In terms of partition framing for “061110260180”, only walls with the exact design feature of "*50 mm × 102 mm studs, 305 mm O.C., 2,450 mm high, single bottom-plate and double top-plate*” are highlighted in Figure 3.17f, whereas other partition walls that may only embrace one of the specified features are not highlighted after executing the corresponding query statement. As another example, door buck for structural wall framing “061110400170” is taken off in order to evaluate the interrelationship between door bucks and walls, “hasSubComponent”, that is inferred through ontology reasoning. As “hasSubComponent” is the inverse of “isPartOf” in ontology modelling, and only “isPartOf” is extracted from Revit-based BIM design models, “hasSubComponent” is inferred through executing ontology reasoning and can be utilized in the SPARQL query, as demonstrated in Figure 3.17g. All the queries are found to generate the expected results, a finding which demonstrates the reliability of the prototype system.

```
SELECT Distinct (str(?lConnection) as ?intersection)
 WHERE {
                ?lConnection rdf:type proOnto:LConnection .
                ?lConnection proOnto:Primary ?angle .
                ?hostWall proOnto:hasIntersection ?lConnection .
                ?hostWall proOnto:Height ?height .
                ?hostWall proOnto:IsExterior false .
                ?hostWall proOnto:Level "First Floor Plan"^^xsd:string .
                ?hostWall rdf:type proOnto:StudFramedWall  .
                 FILTER (?angle != 3.14) .
                 FILTER (?height < 8.15) .
                 FILTER (?height > 7.85)
              }
```

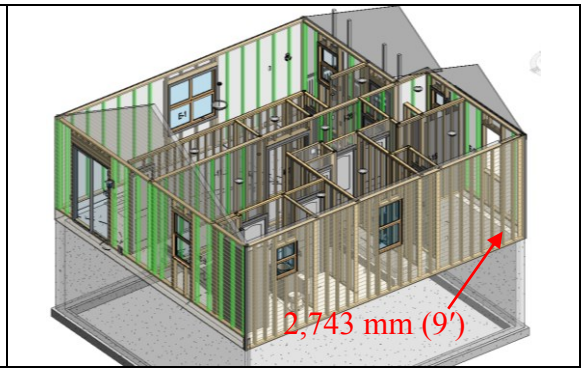
a. Query statement for metal corner bead



b. Visualization for metal corner bead

```
SELECT Distinct (str(sum(?value)) as ?totalLength)
WHERE {
                ?stud proOnto:Length ?value .
                ?stud rdf:type proOnto:Common .
                ?stud proOnto:Size "L2X6"^^xsd:string .
                ?stud proOnto:isPartOf ?hostWall
                ?hostWall proOnto:Height ?height.
                ?hostWall proOnto:IsStructural true .
                ?hostWall proOnto:Level "First Floor Plan"^^xsd:string .
                ?hostWall rdf:type proOnto:StudFramedWall  .
                FILTER (?height < 8.15) .
                FILTER (?height > 7.95)
             }
```

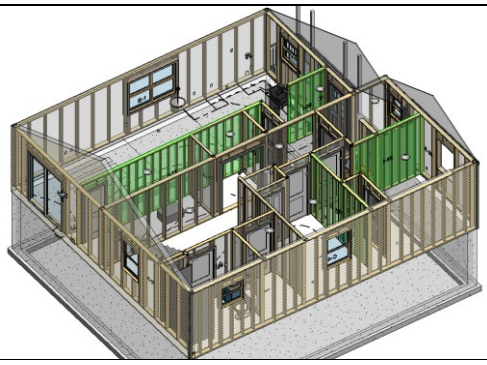
c. Query statement for structural wall stud framing



d. Visualization for structural wall stud framing

```
SELECT Distinct (str(sum(?value)) as ?totalLength)
WHERE {
                ?hostWall rdf:type proOnto:StudFramedWall .
                ?hostWall proOnto:IsExterior false.
                ?hostWall proOnto:IsStructural false.
                ?hostWall proOnto:Level "First Floor Plan"^^xsd:string.
                ?hostWall proOnto:Height ?height .
                 FILTER (?height < 8.15) .
                 FILTER (?height > 7.95) .
                ?hostWall proOnto:Length ?value.
                ?hostWall proOnto:StudSize "L2X4"^^xsd:string .
                ?hostWall proOnto:StudSpacing "12"^^xsd:string .
                ?hostWall proOnto:HasDoubleTopPlate true.
             }
```

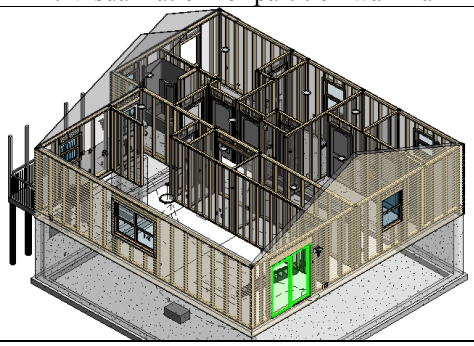
e. Query statement for partition wall framing



f. Visualization for partition wall framing

```
SELECT (str(?doorDuck) as ?doorBuck)
WHERE {
                ?hostWall rdf:type proOnto:StudFramedWall .
                ?hostWall proOnto:StudSize "L2X6"^^xsd:string .
                ?hostWall proOnto:hasSubComponent ?doorDuck.
                ?doorDuck rdf:type proOnto:DoorDuck .
                ?doorDuck proOnto:Length ?length.
                FILTER (?length < 6.1) .
                FILTER (?length > 5.9) .
              }
```

g. Query statement for door buck



h. Visualization for door buck

Figure 3.17 Examples of semantic query

Table 3.2 SPARQL query results

Line Number	Description	Quantity	Unit
061110405167	Wall framing, studs, 50 mm × 152 mm, 2,540 mm high wall, pneumatic nailed	120.47	*m*
092915100411	Accessories, gypsum board, corner bead, galvanized steel, 32 mm × 32 mm × 2,450 mm	6	*Each*
061110260180	Wood framing, partitions, standard & better lumber, 50 mm × 102 mm studs, 305 mm O.C., 2,540 mm high, includes single bottom plate and double top plate, excludes waste	14.42	*m*
061110400170	Wall framing, door buck, king studs, jack studs, header and accessories, 50 mm × 152 mm wall, 1.828 mm wide, 2.540 mm high	1	*Each*

3.7 Discussion

Ontology, as a formal approach to naming domain entities and describing their interrelationships and properties, provides a promising solution to organizing information for construction management applications. Ontology technologies such as RDF are the basis for the semantic web as it allows structured and semi-structured data to be shared and integrated across different applications. Moreover, a data model using ontology technology such as RDF is saved in a graph structure which represents and stores data with nodes, edges, and properties. This graph data model enables semantic queries and provides the semantic foundation for knowledge-based computer applications. As such, the developed RDF/OWL-based ontology in this study is utilized to enhance current BIM design models by adding domain terms and their interrelationships and properties. Existing BIM data in a given BIM model is extracted and further analyzed to derive the building information (e.g., wall connections) that is not modelled explicitly in the BIM. By deriving the implicit BIM data, instead of explicitly modelling it, a large amount of human efforts during the modelling phase can be saved; meanwhile, no additional efforts are required to obtain construction-oriented QTO information during the quantity take-off process. On the other hand, deriving the implicit BIM data enables QTO professionals to take off the implicit BIM features on the basis of existing design-oriented BIM

models, which improves the QTO efficiency. All explicit and derived BIM data are used to populate the construction-oriented product ontology, and ontology reasoning is further applied to infer new information on the basis of extracted BIM data in order to generate the ontology-enhanced BIM model in an RDF/OWL file. The resulting RDF model conceptually functions as a domain-specific "model view" of the given BIM model while enabling semantic queries to facilitate construction-oriented QTO. As a result, construction practitioners can semantically query a BIM design model in order to generate QTO for construction activities by using their domain vocabularies, without the need to understand the technical structure of the underlying complex BIM schema.

Due to the fact that BIM design models lack domain semantics and standardized BIM object definitions in specific building domains, construction practitioners may take off their work packages based on BIM models designed using various BIM authoring tools. In this case, they need to understand various BIM object representations for the same building objects or design features (e.g., Structural Column and IFCMember for studs in light-frame walls), which increases their workload and complicates the daily planning work. Formalizing the domain terms into ontology allows users to apply a unique domain vocabulary they are conversant with in order to semantically query the BIM design model, thus addressing this problem to a certain degree and improving QTO efficiency. Domain terms and their various relationships are explicitly represented within RDF-based ontology in a "subject-predicate-object" triple structure, while the SPARQL query statement is formulated in the same triple structure using domain vocabularies. Moreover, each triple in a SPARQL query statement defines one filtering condition in connection with each design feature for the desired QTO. SPARQL queries are well aligned with users' mental models of the targeted domain. Hence, it is straightforward to formulate

SPARQL queries on BIM models in order to retrieve construction-oriented QTO information. SPARQL query statements in the present research are formulated by the authors based on the description of work packages. Once users are acquainted with SPARQL, it would generally take less than one minute to formulate one query statement. Of course, this means that some learning effort would be necessary in order for construction practitioners or QTO professionals to master SPARQL. Moreover, construction practitioners may find it difficult to design SPARQL queries using rich vocabularies. Therefore, in future work, GUI utility functions will be investigated to empower construction practitioners to take full advantage of rich vocabularies in formulating their own SPARQL query statements in a user-friendly fashion. Such user-friendly utility functions will maximize the benefit of the proposed semantic QTO approach.

3.8 Conclusion

This research proposes an ontology-based semantic approach for construction-oriented quantity take-off (QTO) and develops a prototype QTO system as an Autodesk Revit add-on in the particular context of light-framing building construction. The established construction-oriented product ontology enhances the current BIM design models by extending domain terms and their properties and interrelationships and aligns the BIM models with construction-oriented QTO, such that construction practitioners can take off the quantities for work packages under certain design features that may not be modelled explicitly in BIM. This ontology is established in Protégé 4.3 in the format of RDF/XML; dotNetRDF (an open source .Net Library for RDF) is integrated into the prototyped system. Hence, Simple Protocol and RDF Query Language (SPARQL) can be utilized to query the ontology-augmented BIM model, which allows construction practitioners to formulate semantic queries using richer vocabularies they are conversant with. The main contributions of this research are summarized as follows:

1) Establishing a construction-oriented product ontology, which extends current design BIM models by adding domain terms and their properties and interrelationships without changing the original BIM schema. Detailed wall connection information is derived based on existing BIM spatial and geometrical data and modelled explicitly in the ontology-enhanced BIM model, all intended to support construction process oriented QTO.
2) Prototyping a semantic QTO system as an Autodesk Revit add-on. This system includes: (i) a Revit BIM model parser that converts the Revit BIM model into an ontology-augmented BIM model in an RDF file; and (ii) a semantic search graphic user interface (GUI) that enables semantic queries on the ontology-augmented BIM model, capitalizing on the semantic awareness provided by RDF-based ontology and SPARQL query. In addition, the prototyped system is capable of visualizing the query results in order to facilitate communication among project stakeholders.

However, this prototyped system currently has some limitations. For instance, although ontology reasoning by using the default reasoners in Protégé 4.3 can infer some implicit information such as relationships between ontological entities in the prototyped system, the rule-based ontology reasoning by Semantic Web Rule Language (SWRL) is not yet supported due to the fact that the present research does not encompass SWRL. It is anticipated that SWRL-based ontology reasoning will further provide semantics to the QTO system and will be addressed in the future. Additionally, the construction-oriented product ontology is formalized in the particular context of light-frame building construction. Algorithms for detecting implicit design features such as StudSpacing and HasDoubleTopPlate are specific to light-frame building, whereas the geometrical algorithms designated to detect various implicit connections are generic for all kinds

of building projects. For other types of building projects, construction practitioners can still rely on the geometrical algorithms to detect such intersection information in planning day-to-day work, but other algorithms and specific domain ontologies need to be further developed or customized so as to adapt the proposed semantic QTO approach to applications on other types of building projects. Additionally, zone-based wall intersection detection, instead of checking detections on every combination of two walls, will be instrumental in simplifying the geometrical algorithms and improving computing efficiency. Ontology development is a continuous process and the established ontology should be continuously updated in order to satisfy new requirements of construction-oriented QTO. Another potential extension in future research is to shift the prototype system from a vendor related Revit-based application to a fully standardized IFC-based BIM application.

CHAPER 4: PANELIZED CONSTRUCTION SCHEDULING[3]

4.1 Introduction

Building information modelling (BIM), described as "*digital representation of physical and functional characteristics of a facility*" (National BIM Standard, 2013), has been regarded as a potential solution to challenges within the Architecture, Engineering, and Construction (AEC) industry due to the following capabilities (1) BIM is able to store all the information pertaining to a facility, which lays the foundation by which the BIM tools perform a variety of analyses, such as structural analysis and schedule planning analysis (Weygant, 2011); and (2) BIM can facilitate information exchanges and interoperability between software applications during the project life cycle (Howard & Björk, 2008), which boosts work efficiency and enhances communication and collaboration among project participants. Applications of BIM have thus garnered much attention within the construction industry in recent years. In particular, researchers and construction practitioners have explored different ways to perform schedule planning with the support of BIM. However, BIM in most cases functions as a database of 3D building components and provides only limited information of each component (e.g., quantity take-offs) for the downstream scheduling analysis. Rich building information embedded in BIM is not being fully utilized in order to facilitate the automatic generation of project schedules, entailing substantial manual work, especially in construction sequencing and information exchanges between BIM modelling tools and scheduling tools. In this case, BIM in current practice offers only limited advantages over traditional 3D-CAD models.

Construction schedules and plans should be formulated at the appropriate level of abstraction and detail (Fischer & Aalami, 1995), and construction activities need to be manageable from the

[3] A version of this chapter has been published in the journal of Automation in Construction, 53, pp. 29-43.

construction perspective. As such, construction activities can be formulated by three rules: 1) type of work (distinct activities requesting different resources); 2) operationally significant function (distinct activities carried out on components with different functions); 3) operationally significant location (distinct activities carried out in different zones) (Gray, 1986). Nonetheless, these three general rules are not sufficient to cater for the needs of some projects such as panelized building projects. Defining activities/processes in panelized construction should distinguish each individual building component, instead of distinguishing each construction zone, in that each pre-fabricated component is unique and needs to be installed at its own designed location and be scheduled individually in order to manage and coordinate factory production and on-site construction processes. In the current practice most construction schedules generated from BIM are formulated at the project level where constructing one building component is usually assumed to be one construction activity, or at the construction zone level where activities are defined for particular construction zones. These activity-level schedules do not delve into different construction operations which request different resources in accordance with specific construction methods in order to build individual building components. Moreover, when construction scheduling involves details at the activity level, both technology precedence constraints and resource constraints must be taken into account in order to create a meaningful detailed schedule. In current practice, resource constraints are overlooked in BIM-based scheduling; optimization technology is not yet integrated with BIM and process simulation model to address resource-constrained scheduling problems.

The research presented in this chapter explores a BIM-based integrated scheduling approach that automatically generates optimal component-centric activity-level schedules for construction projects by performing simulation-based scheduling from the BIM model. More specifically, in

the proposed BIM-based scheduling approach, rich product information from BIM models and work package information from a Microsoft (MS) Access Database, are automatically extracted and fed as inputs to the process simulation model that mimics the construction logic and performs simulation-based scheduling analysis. The in-depth integration of 3D BIM product model and process simulation model yields an activity-level construction schedule. An evolutionary optimization algorithm is also incorporated into the proposed approach to evaluate various construction sequences under technical and resource constraints and ultimately obtain the optimized activity-level schedule.

4.2 Literature Review

Traditionally, construction scheduling is formulated manually in the form of 2D bar charts by means of critical path method (CPM). This is a laborious and highly error-prone process that challenges construction practitioners. Recently, with the advances in 3D computer aided design (CAD) and information technology, researchers and construction practitioners have been seeking to develop computer-assisted scheduling tools in order to boost scheduling efficiency and relevance.

4.2.1 Construction planning using 4D CAD

Among these efforts, one well-known concept in the construction domain is 4D CAD, also known as 4D visualization. 4D CAD models, the BIM prototypes that leverage 3D models for schedule information, are able to assist project participants in visualizing the construction plan (CPM plan) in 3D and identifying conflicts prior to construction commencing. 4D technology has proven to be more effective than traditional CPM or Gantt chart for construction planning (Staub-French et al., 1999). Based on the concept of 4D CAD, Chau et al. (2005) has further developed an information system, called 4D graphics for construction planning and site

utilization, that extends 4D technology into the field of resource management and site space utilization. Subsequently, Lu et al. (2009) proposed a methodology for integrating 4D CAD with 3D animation of operation simulation in order to visualize construction operations involving dynamic interaction of various construction resources. Incorporating a scheduling feature can further enhance 4D visualization; the current technology requires additional effort to link the external schedule with the 3D objects for the purpose of visualization (Tulke & Hanff, 2007). It should be noted that "scheduling feature" here refers to the direct generation of schedules, including the schedule logic and activity times from BIM or 3D models.

4.2.2 BIM-based scheduling

Since BIM hosts enriched project information, which is required for schedule analysis, it is capable of supporting the generation of construction schedules. In this respect, attempts to automate the process of project scheduling based on BIM or 3D-CAD models have been carried out in recent years. For instance, De Vries & Harink (2007) developed an algorithm that generates component-level construction schedules from a 3D-CAD model. This algorithm determines construction orders of building components based on their spatial relationship (i.e., which component is adjacent to or on top of another component). Similarly, Kataoka (2008) introduced an automated scheduling approach that formulates construction schedules based on pre-defined construction method templates and 3D building geometries. The developed system is intended for use prior to the building structural system being specified. Tauscher et al. (2009) proposed a novel IFC-based method to semi-automatically generate construction schedules. In their study, construction schedules are generated by means of case-based reasoning based on data extracted from an IFC-based BIM model. More recently, Moon et al. (2013) developed a BIM-based construction scheduling approach which employs BIM (to visualize construction activities)

as well as optimization theory (to reduce activity overlaps). Their method did not encompass the concept of BIM-based scheduling (i.e., direct schedule generation) defined in this chapter. Meanwhile, Kim et al. (2013) automated the generation of construction schedules by extracting building information from IFC-based BIM models as the inputs for scheduling analysis. The precedence relationships among construction-zone level activities are determined by using formalized sequencing rules in their prototyped system. Another recent effort has been a BIM-based framework proposed by Chen et al. (2013) to yield the near-optimum schedule. Their framework involved a manual process for explicitly establishing a complete activity network and assigning quantity take-offs from 3D CAD to activities. In their study, 3D CAD only provided quantity take-offs for the process simulation model. Due to the fact that the framework was not integrated with an optimization algorithm for the purpose of efficiently exploring the search space for optimum solutions, the near-optimum schedule was obtained by simply picking the best solution in multiple runs.

4.2.3 Simulation-based scheduling

Simulation-based scheduling approach has been suggested by previous scholars for detailed scheduling at the construction operation level, capitalizing on the capability of discrete-event simulation (DES) to mimic the construction operation logic and investigate the resource allocation among activities. A number of simulation-based scheduling models/tools have been developed to date, including an activity-based simulation model for project scheduling (Zhang et al., 2002) and the simplified discrete-event simulation approach (SDESA) (Lu, 2003). Lu et al. (2008) further developed a Simplified Simulation-based Scheduling system (S3) to perform resource-constrained critical path analysis by integrating SDESA with particle swarm optimization (PSO). Taghaddos et al. (2009 & 2012) developed a simulation-based scheduling

system for module assembly in industrial projects. Hu & Mohamed (2010) introduced a state-based simulation mechanism for facilitating project schedule updating, and Hong et al. (2011) proposed an estimation model for core wall construction of high-rise buildings. The aforementioned efforts in simulation-based scheduling sought to address the schedule problem by leveraging the advantages of DES, but have not yet explored the seamless integration of process simulation with BIM to facilitate the automated generation of construction schedules.

Integrating BIM with process simulation can assist practitioners in scheduling construction project due to the fact that BIM can provide quantity take-off information for the process simulation model, as validated in a study by Wang et al. (2014). In their study, a stand-alone module (Visual Basic application) was developed to read quantity take-off information in MS Access and feed it as inputs to the predefined simulation model in order to generate the construction schedule. The quantity take-off in the study was generated and saved into MS Access database manually through the "Schedule" function in Revit. The generated schedule served as a project-level schedule for each type of building element, since BIM only provided quantity take-offs, instead of rich product information to the roughly pre-defined process simulation model. To address the need for detailed activity-level scheduling, an in-depth integration (rich building information exchanges) between BIM and process simulation model must be achieved.

Konig et al. (2012) have conceived of an intelligent concept by which to store interdependencies among activities to be reused in future project scheduling based on BIM and DES. Two kinds of templates were defined in their research: (1) a process pattern for individual building components, and (2) complex interdependencies among building elements. These templates were

developed based on construction knowledge and experience, and were assigned to building elements manually, entailing a large amount of manual effort to generate the schedule.

The literature review reveals that BIM-based scheduling has been addressed in the previously related research, either by applying construction sequencing rules to BIM or by performing simulation-based scheduling with BIM feeding quantity take-offs to the process simulation model. The former approach usually summarizes spatial relationships (e.g., supported by, embedded in) as construction sequencing rules in order to infer the precedence relationship between construction activities. Then, the derived precedence constraints are taken into account in construction scheduling, whereas resource constraints are generally ignored or implicitly dealt with. On the other hand, the simulation-based approach at present has yet to take full advantage of the rich product information in BIM models so as to automate schedule generation. This necessitates a large amount of human judgment and intervention involved in construction sequencing and simulation model development (building simulation network), especially when construction projects are scheduled at the component-centric activity level. In fact, establishing a complete activity network manually is a challenge at times due to the practical need to adjust construction technology under physical and spatial constraints in the field and the dynamic precedence constraints between activities caused by resource allocation strategies, which is further elaborated in the "Background of Light Gauge Steel Construction" section of this chapter, and the simulation model cannot even mimic construction processes without the provision of enriched information from the 3D BIM model. For instance, wall panels for structural usage and non-structural usage will go through different processes and capture different resources in the simulation model, respectively. The research presented in this chapter thus seeks a new BIM-based integrated scheduling approach which can address the difficulty of manually building a

complete activity network and overcomes the limitation of defining a fixed activity network in project planning through exchanging enriched information between BIM and DES model with an incomplete simulation network. Notably, the DES model at best represents an essential but incomplete simulation network of the complicated building processes being modeled.

4.3 Integrated Methodology

Unlike previous research, in which BIM only provides quantity take-offs for project-level or construction zone level scheduling, this approach achieves in-depth integration among BIM product models, process simulation, and optimization models, thereby facilitating automatic generation of optimized component-centric activity-level schedules for construction projects. Within the integrated system, a BIM product model is supplemented with work breakdown structure (WBS) information, while the process simulation model can gain rich product information (including quantity take-offs) from BIM and work package information (e.g., operation productivity) from WBS in order to generate component-centric activity-level construction schedules. Moreover, this research requires the planner to build part of the activity network manually as per constraints that need to be observed and remain constant during construction, instead of a complete activity network as in previous research, in order to address the difficulty of manually building a complete activity network and overcomes the limitation of defining a fixed activity network in project planning. The dynamic precedence constraints on activities will be derived at run time of DES through BIM-simulation integration, whereas resource-induced precedence constraints are addressed by using priority dispatching rules to allocate limited resources in the simulation model. In addition, an evolutionary algorithm, namely, particle swarm optimization, is selected for computational efficiency and effectiveness in arriving at optimum solutions for large, complex systems, and hence is incorporated into the

methodology in order to optimize the construction sequences with the objective of minimizing project duration under resource constraints. The integration is realized through the enriched information entity, as shown in Figure 4.1 and Figure 4.2. Figure 4.2 also shows all types of information extracted from the BIM product model and from the MS Access database as attributes of enriched information entities. The generated schedule of each activity is recorded by simulation entities following execution of the process simulation model. Afterward, all the information carried by entities is exported into an XML file, and the schedule in the XML file can be presented in the form of a bar chart or a network diagram in order to facilitate communication among construction participants. Autodesk Revit, MS Access, Simphony.NET 4.0 simulation engine (Mohamed & AbouRizk, 2000), and MS Project are all employed in this research in order to achieve the objective. The methodology is shown in Figure 4.1, and a detailed explanation of the methodology and interactions among different components are presented in the following sections.

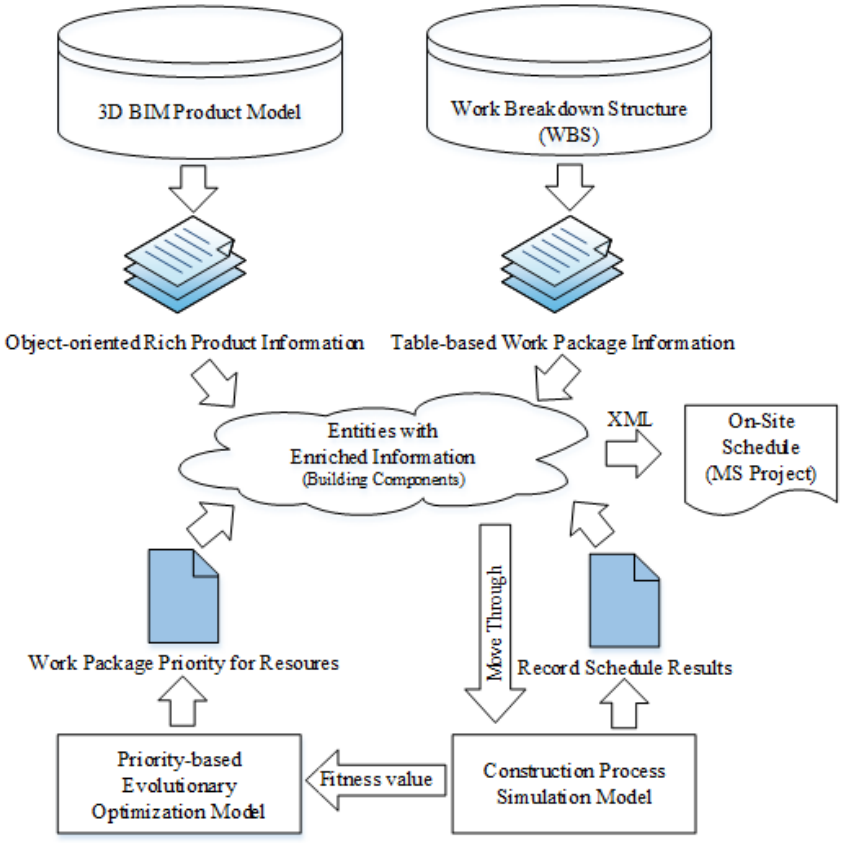


Figure 4.1 Overview of integrated methodology for detailed schedule planning

4.3.1 Building Information Model

Due to the fact that BIM embraces all the product information about the building product, such as material information and functional information, in this research BIM serves as the central database that supplies the process simulation model with the enriched product information. In other words, BIM not only provides quantity take-offs, but more importantly includes other rich product information (see Figure 4.2) such as topology/connections, supports, and functionalities (e.g., structural usage) to support the process simulation with respect to technical construction logic. For instance, the process simulation model can control the construction sequence of structural floors and their supporting elements (e.g., structural walls) based on the functional and supporting information of floors that is automatically extracted from the BIM model, and should

always proceed with the construction of structural floors after all their supporting elements are completed. In this respect, rich building product information from BIM provides the necessary input for automating the process of detailed construction scheduling.

Notably, the information with regard to "Connections" and "Supports" of building components is not presented in the BIM model explicitly. Further analysis, such as topological analysis and structural supporting relationship analysis based on geometrical and functional information, is thus required. A detailed explanation of this analysis is given in the "Implementation" section of this chapter.

4.3.2 Work breakdown structure (WBS)

Currently, BIM is largely associated with product design model, while work breakdown structure (WBS), or work package information, is not represented in the BIM system. This potential deficiency poses a challenge to BIM-based activity-level construction scheduling. In order to address this issue, this chapter proposes an approach to supplement BIM with work package information by storing WBS in a MS Access database. Additionally, work package items are organized into "Tables" in MS Access in accordance with the "Type" of building component being constructed, thereby facilitating the integration of BIM with WBS information. Subsequently, the process simulation model, discussed in the following section, will receive enriched inputs from both the BIM model and MS Access database by relating each Access table with a corresponding building element in Revit. In addition, information about all available resources for the project is stored in another MS Access table, and this information is extracted as inputs for the process simulation model in order to identify the effect of resource availability on project duration and to assist construction practitioners in resource management at the project level.

4.3.3 Construction process simulation

Although a BIM model in Revit can be integrated with WBS information to obtain information on the productivity and resource requirements of each work package in MS Access through the aforementioned approach, the construction schedule still cannot be generated from the BIM model due to the fact that BIM in general does not contain any knowledge with regards to the construction logic. To address this issue, construction process simulation is incorporated into this research. The process simulation model must be developed to mimic the construction processes in detail, and to process construction tasks in consideration of the specified construction method, based on rich product information from BIM.

4.3.3.1 Integration of BIM product model and process simulation model

A BIM model usually is an assembly of pre-defined 3D building objects, also known as parametric objects, where rich building information is embedded into building components as their attributes or parameters. Some simulation modelling systems, such as *Simphony*, a simulation environment developed by researchers at the University of Alberta (Mohamed & AbouRizk, 2000), provides a similar concept called "entities", which makes the simulation capable of handling complex and interactive problems. Entities are allowed to carry attributes with them as they move through the process simulation model. Consequently, parametric objects in a BIM model can be fully or partially represented by entities in a process simulation model, given that simulation entities are designed to represent building components. Hence, the integration of a BIM product model and a process simulation model can be realized by enriched information entities, which extract rich building product information of building components from BIM and WBS information from MS Access. These entities move through the process simulation model as guided by this enriched information.

Figure 4.2 shows the entity relation diagram of the enriched information entity representing building components. As depicted in the figure, the attribute of "Priority for Resource" is generated from the optimization model, while "Schedule Information", such as durations and predecessors, is provided by the process simulation model. All the information denoted by solid circles in Figure 4.2, with the exception of the work package IDs, is extracted from the BIM product model. The attributes in the dashed-circles are sourced from WBS stored in MS Access. (It should be noted that ID of work packages is a composite attribute which combines "Project", "Level", "Unit", "Element Type" and "Element ID" of building elements and "Name" of work packages in order to identify each work package under each building element.)

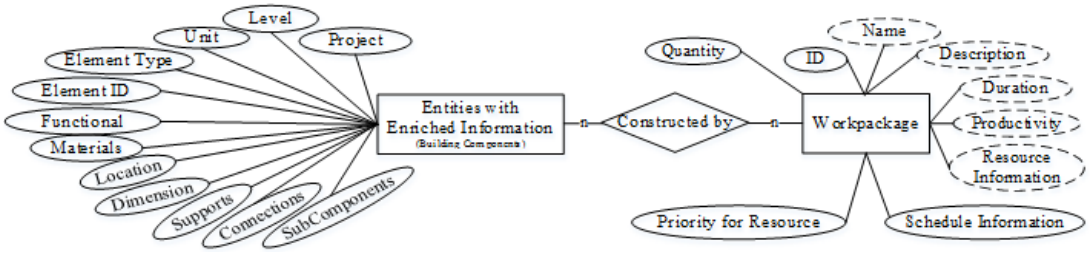


Figure 4.2 E-R diagram of entity with enriched information

4.3.4 Optimization of construction schedule

Resource limitations lead to a resource-constrained project scheduling problem (RCPSP) whereby limited resources are allocated among tasks with pre-defined resource requirements. The objective of this research is to produce the optimal activity-level schedule with the minimum project duration for a competent-centric construction project under resource constraints. Note only construction sequences that are practical to adjust on site are optimized, instead of adjusting quantities of required resources for each activity. This is consistent with a typical RCPSP problem as defined in the literature. Optimization of construction schedules by adjusting resource requirements is out of scope of this research and can be studied in future. The

targeted scheduling problem can be solved by using evolutionary algorithms, such as genetic algorithm (GA) and the particle swarm optimization (PSO) algorithm introduced by Eberhart et al. (1995). Zhang et al. (2006) successfully applied PSO in the construction domain, and they demonstrated two PSO-based solutions—priority-based particle representation and permutation-based representation—in solving the RCPSP problem. Lu et al. (2006) further demonstrated that PSO is superior to GA in converging.

In addition, DES systems such as Simphony in general allow the specification of priority dispatching rules, which can also be utilized to schedule activities under resource constraints. In this approach, resource conflicts among activities can be addressed due to the fact that construction activities are served by limited resources in descending order of their priority number in simulation models. Meanwhile, this approach has the potential of being integrated with evolutionary algorithms to optimize construction schedule. This research thus integrates PSO with the process simulation model in order to solve the RCPSP problem. The integration between process simulation and evolutionary optimization based on PSO algorithms is demonstrated in the later section. It should be noted that other evolutionary algorithms can also be incorporated into the methodology in the similar manner.

4.3.4.1 Integration of process simulation and optimization

The priority-based particle representation of PSO is employed in the present research to optimize construction sequences, since it can be seamlessly integrated with the priority dispatching rule in DES. In general, the PSO algorithm by means of priority-based particle representation searches for the optimum solution by identifying a combination of priority key values assigned to each activity (work package). As depicted in Figure 4.1, the optimization model feeds the process simulation model with work package priority information. The process simulation model,

serving as the "objective function calculator", in turn calculates the fitness value (project duration) for the optimization model. More detailed interaction is shown in Figure 4.3. To begin with, the PSO initializes the particles' positions (priorities of all work packages) through random sampling; for a given particle, the priority information of construction work packages is published to the process simulation model by attaching it with simulation entities as attributes. Simulation entities assigned with priorities are then served by the required limited resources in descending order of priority number in the process simulation model. Following execution of the simulation model, the fitness value (project duration) of each particle is obtained from the process simulation model, and then sent back to the PSO. The PSO further identifies the global best position of all particles and the local best position for each particle in the current iteration. Afterward, each particle in the PSO updates its current state, including velocity and position, based on the global and local best positions of particles. The next iteration is then started and new positions of particles are evaluated in the process simulation model. The iteration processes do not stop until the PSO reaches its termination criteria, such as completing the specified number of iterations. During each iteration, the schedule information resulting from the process simulation model is constantly updated into simulation entities.

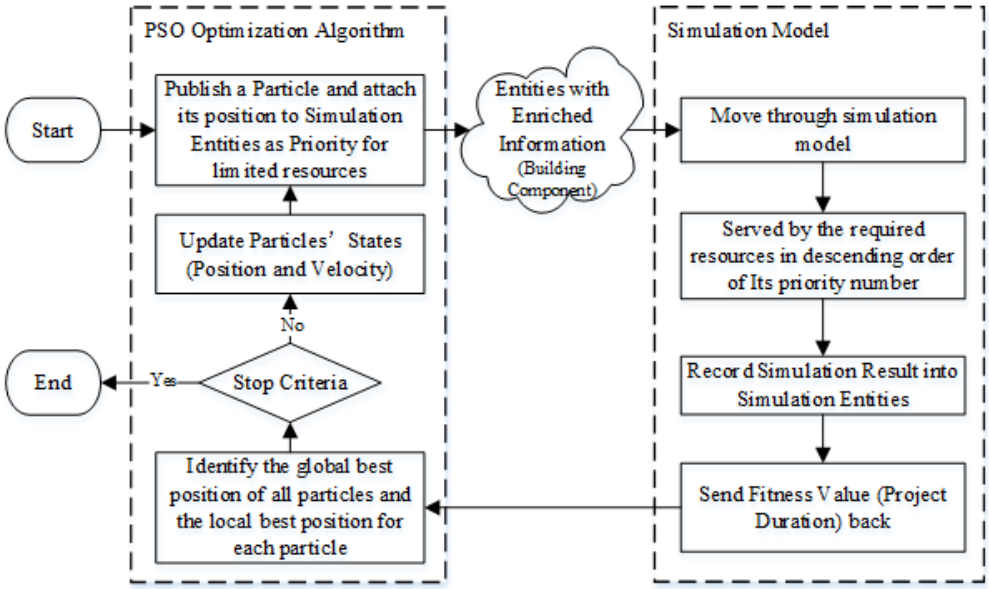


Figure 4.3 Interaction between PSO optimization algorithm and simulation model

4.4 Background of light gauge steel (LGS) construction

The methodology is implemented for panelized construction projects which uses the light gauge steel (LGS) system. In this section, the on-site construction method for an LGS system is described. Presently, in LGS construction practice, structural bearing walls are pre-assembled in the factory and installed on site as wall panels. In the case of non-bearing walls, the steel building materials are delivered to the construction site and the walls are installed in place in a conventional manner. More specifically, each structural wall panel is assembled in four steps: *Lift wall panel, Connect wall panel, Install insulation, and Install drywall*. Non-bearing wall panels, alternatively, are constructed through the following sequence: *Lift wall panel studs, Frame wall panel in place, Install insulation, and Install drywall*. Wall panels, regardless of whether they are structural or non-structural elements, at the first level require another activity, *Survey panel location*, to be completed before they can be lifted. The floor, made up of steel joists, is constructed in the same fashion as the non-bearing walls. It is assembled on site from pieces of steel joists. The main steps to construct the floor system include *Lift floor joists* and

Frame floor joist. It is of interest here to mention that the washrooms are pre-fabricated as module in the factory and are then shipped to the site for on-site installation. A washroom module usually consists of four wall panels, a floor, and a ceiling (which serves as the floor for the level above). All other components in the washroom, such as the tub, are pre- installed prior to shipment to the site.

Additionally, since the washroom is a stand-alone module which does not require an additional temporary bracing system, on-site assembly work for wall panels at the same level as the washroom always begins with the installation of the washroom module. The next components to be lifted are the wall panels, which have connections with the wash-room module. Note that "*Connection"* here refers to building components sharing at least one contact area which is greater than zero. Following the connections among the walls, other wall panels are lifted and framed sequentially. Figure 4.4 illustrates one feasible lifting and framing sequence of an apartment unit in a panelized building. In this figure, "CS" refers to the construction sequence and "Bearing"/"Non-Bearing" indicates the wall panel structural function. In this case, some wall panels have identical construction sequence numbers since technically they can be assembled concurrently, provided that there are sufficient construction resources (e.g., equipment and labour) to allow concurrent activity execution on the construction site. Construction of the steel joists for the floor is divided into zones corresponding to the different apartment units in the residential building, i.e., assembly of floor joists is performed zone -by-zone.

One challenge of detailed scheduling in construction projects, including LGS systems, is that construction activities have dynamic precedence constraints depending on the resource allocation strategy employed. For instance, wall panel #369815 (in the solid rectangle in Figure 4.4) is

connected with wall panels # 515298, #419019, and # 369814 (denoted by the dashed lines in the figure). In accordance with the construction method described earlier, "*Lift wall panel"* for wall panel #369815 hence has "*Lift wall panel"* for any one of the other three wall panels (#515298, #419019, and #369814) as its precedence constraint. For the case in which "*Lift wall panel"* for wall panel #515298 is completed in advance, its precedence constraint could be "*Lift wall panel"* for wall panel #515298, and lifting wall panel #369815 can then be started once the resource constraints are satisfied. This poses a challenge for scheduling in the form of a fixed activity network diagram, such as CPM or traditional simulation model, which needs an explicit network to perform schedule analysis. The existing scheduling tools, including Primavera P6 and S3 (Lu et al., 2008), cannot handle such flexible precedence constraints in carrying out detailed construction scheduling under resource constraints. The integrated approach described in this chapter, on the other hand, can address this problem and yield the schedule automatically by integrating the process simulation with BIM and optimization technology.

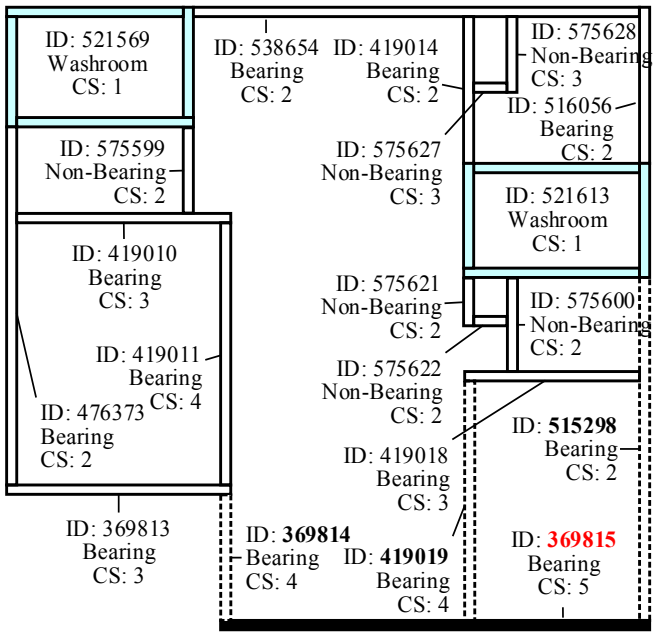


Figure 4.4 Installing sequence of wall panels in a structure

4.5 Implementation

4.5.1 System architecture

An automated scheduling system for panelized construction using LGS has been developed to implement the methodology. The automated system as shown in Figure 4.5 comprises three main components: (1) MS Access, where project resource information and work package information (WBS) are stored; (2) Autodesk Revit, which is used to design the building project; and (3) MS Project, which is employed to display the generated schedule. (Autodesk Revit, it should be noted, includes a Revit add-on which encompasses a structural supporting relationship and spatial relationship analyzer (SSRAnalyzer), a process simulation model in Simphony, and a PSO optimization model to perform the schedule analysis.) The three components are connected through an Autodesk Revit application programming interface (API) in C# language. Figure 4.6 shows the user interface of the developed add-on for Revit. It allows for users to view and edit the WBS information and the process simulation model by clicking corresponding buttons in order to consider different construction methods. Meanwhile, it also shows a portion of the extracted information from the BIM model.

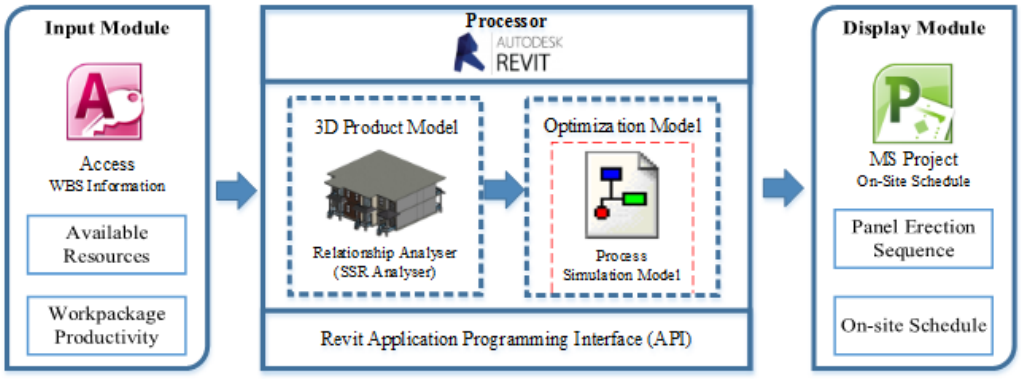


Figure 4.5 Architecture of automated scheduling system

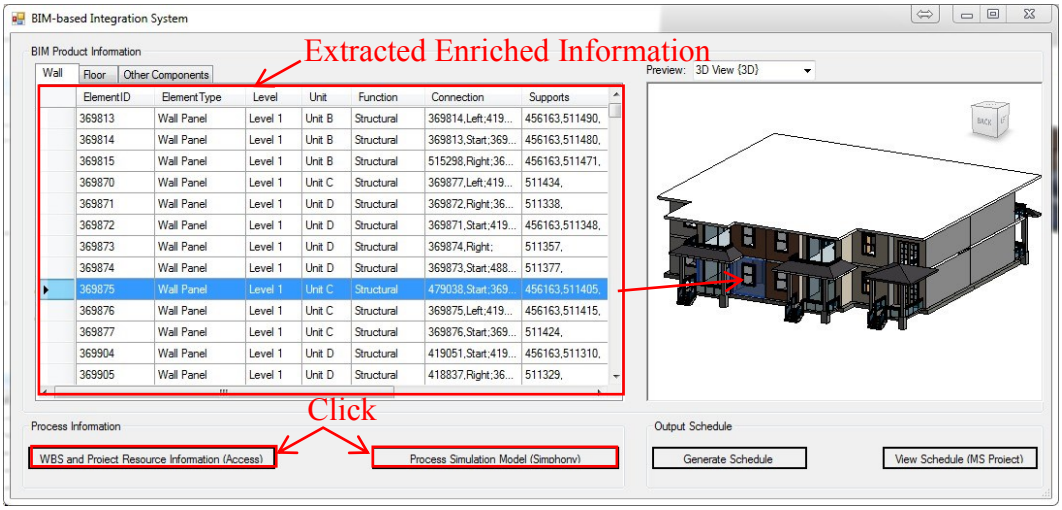


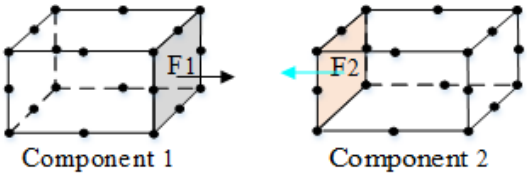
Figure 4.6 Interface of automated scheduling system

4.5.2 Spatial and structural supporting relationship analyzer (SSRAnalyzer)

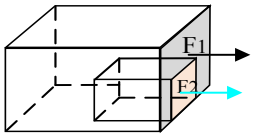
4.5.2.1 Connection relationships among wall panels

Connection relationships of building components are essential to construction scheduling, since they determine the construction logics described in the previous section. A "connection" here refers to the case in which building components share one contact area which is greater than zero, and for which normal vectors of contact faces defined as pointing outward of the solid object are opposite to one another, as shown in Figure 4.7d. Hence, a connection relationship is derived by checking whether two faces (each from one building component) overlap and whether their normal vectors are opposite.

This chapter describes an algorithm by means of which to infer the connection relationships among walls based on geometric information of faces, edges, and points (as depicted in Figure 4.7a). This information is extracted using the Revit API functions of element.get_Geometry(), solid.Faces(), face.EdgeLoops(), Curve.GetEndPoint(), respectively, in the algorithm. The algorithm then takes one face from each component; given that F1 is from component 1 and F2 is from component 2, normal vectors of F1 and F2 are checked to determine whether or not they

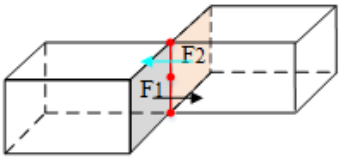


a. Geometric information of building component



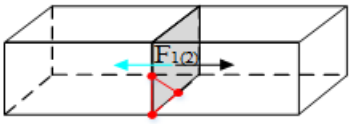
$F1 \cap F2 = F2$

b. Containment



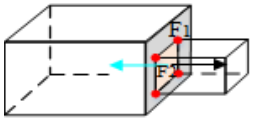
$(F1 \cap F2 \neq \emptyset) \wedge (F1^{o} \cap F2^{o} = \emptyset)$

c. Intersection



$F1^{o} \cap F2^{o} \neq \emptyset$

d. Scenario 1 of connection



$F1^{o} \cap F2^{o} \neq \emptyset$

e. Scenario 2 of Connection

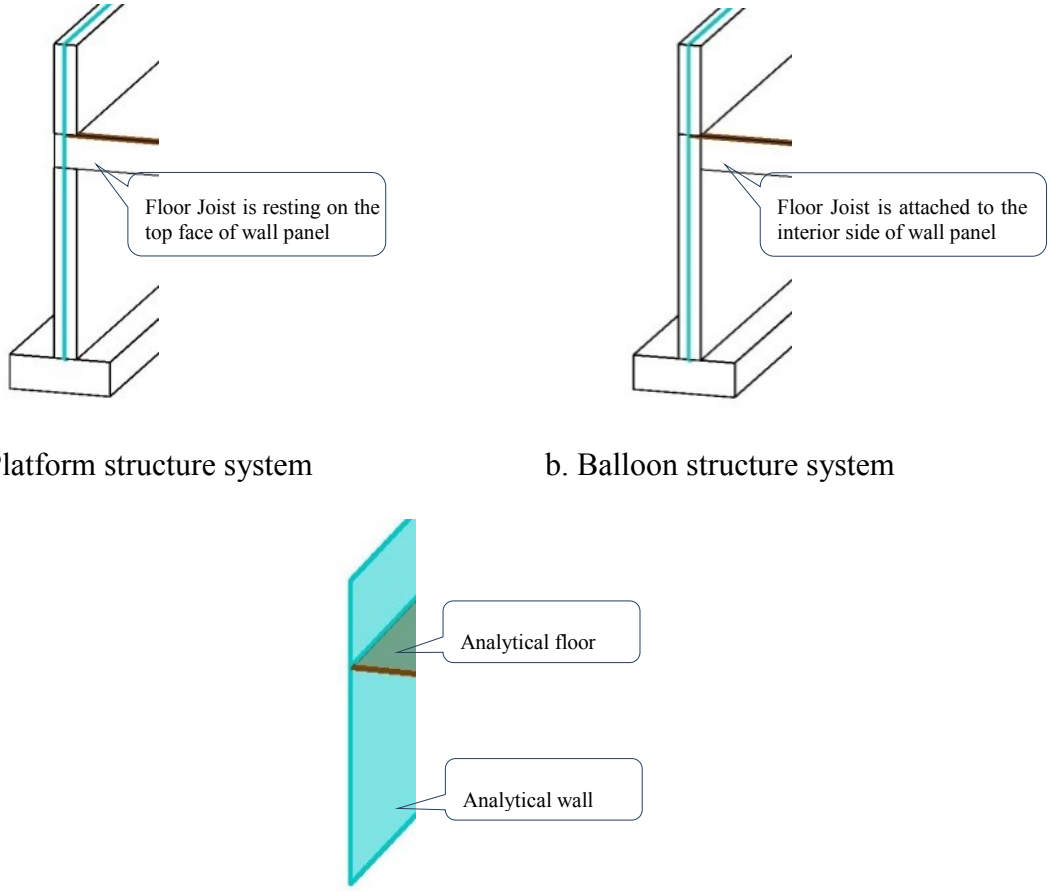
Figure 4.7 Topological relationships among walls

are in opposite directions. This is done in order to exclude the containment relationship as shown in Figure 4.7b. If the vectors are opposite, then all points (including vertices of building components and middle points of edges) of F1 are checked to determine whether or not they lie

in F2. If there are more than 3 points that do not lie in a straight line, the two elements are considered to be connected. The element ID is mutually stored into the connection information of each entity. All points of F2 are also checked against F1 in order to consider the case in which F2 is inside of F1, as shown in Figure 4.7e.

4.5.2.2 Supporting relationship of structural elements

Structural supporting relationships of building components are also important since the construction sequence is subject to the structural behaviour of the building structure under construction. Focusing on spatial relationships (e.g., connection relationship) is not an adequate method to derive supporting relationships and construction sequences. Furthermore, the connection relationships cannot be used to infer the construction sequence between floors and walls in LGS systems. In LGS construction, two different floor systems are commonly adopted in current practice, platform and balloon structure systems, as shown in Figure 4.8. The floor joists are resting on top of the wall panels in Figure 4.8a, whereas the floor system in Figure 4.8b is connected to the interior side of the wall panel. Despite this, both the lower wall panel and floor support the above wall panel, and in the interest of safety in both cases the work of assembling the wall panels on the above floor cannot begin until the floor is assembled.



a. Platform structure system

b. Balloon structure system

c. Structural analytical model for both floor systems

Figure 4.8 Sketch for different floor systems and their structural analytical model

To address these challenges, we propose a new approach in which the structural supporting information of building components is obtained from the simplified structural analytical model, rather than from a 3D geometrical model. As suggested by its name, a structural analytical model is a simplified 3D representation of the structural information model consisting of geometry and material properties of structural components and applied loads, and is used to perform structural analysis. In the simplified structural analytical model, 3D walls and floors are normally represented by 2D shell elements. Figure 4.8 also presents the structural analytical model for walls and floors. Moreover, the analytical model encompasses the supporting information of building elements and is created automatically in Revit when a 3D physical/geometrical model is

developed. Although the supporting information is not accessible to users through the Revit user interface, it can be extracted by means of the Revit API function, "element.GetAnalyticalModelSupport()". With respect to the floor system described above, both the structural wall and floor are recognized by Revit API as supporting elements for the above wall panel. Additionally, supporting information among other structural elements, such as walls and wall foundations, can be extracted in the same manner.

4.5.3 Development of WBS

A LGS building project consists of wall panels, floor system, slabs, stairs, and foundations. Although it also comprises architectural building elements such as windows and doors, these architectural components do not affect the on-site schedule due to the fact that they are preassembled in the factory prior to shipment to the site. Consequently, the work package information stored in MS Access is for wall panels, floor system, slabs, stairs, and foundations. Each type of building component has its own table to store its WBS information. For example, the following are the on-site work packages for "Wall Panel" at the first level: (1) "Survey Panel Location"; (2) "Lift Wall Panel"; (3) "Connect Wall Panel"; (4) "Install Insulation"; and (5) "Install Drywall". All these items, together with the corresponding required resource information and productivity information, are stored in the MS Access table, "Wall Panel", shown in Figure 4.9.

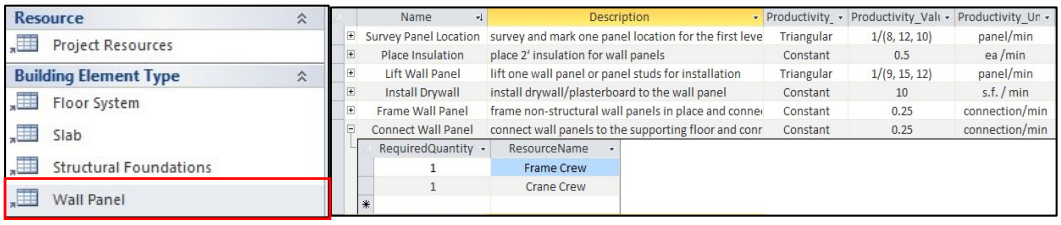


Figure 4.9 Table-based work package information

4.5.4 Development of process simulation model

The research presented in this chapter employs *Simphony*.NET (Mohamed & AbouRizk, 2000) to develop the simulation model since this simulation environment offers several important features, such as entities; calendars; the ability to carry attributes with entities; the capability for the user to write their own code for the purpose of enhancing modelling flexibility; and a simulation engine library (open source library) (AbouRizk, 2010; 2011). These features enable it to handle complex and interactive problems, as well as for it to be easily integrated with other systems in order to generate construction schedules.

During the development of the simulation model, the concept of "Process Pattern" is adopted. Construction knowledge encapsulating construction logic can be generalized as process patterns, which are reusable for similar building elements and building projects (Benevolenskiy et al., 2012; Konig et al., 2012). Process patterns of construction processes for each type of building component in panelized LGS projects are formalized to develop the simulation model in this research. Figure 4.10 presents the typical process pattern for a regular cast-in-place concrete building element, such as a building's foundation. The simulation model for on-site construction of a panelized building project (see Figure 4.12) is further developed based on formalized process patterns. As shown in the figure, a wall panel is usually constructed through four sequential activities: (1) "Lift Wall Panel"; (2) "Connect Wall Panel"; (3) "Install Insulation"; and (4) "Install Drywall". Each activity is represented by a *Composite* element in the simulation model. It is noteworthy that a *Composite* element of Simphony is a "container" housing other simulation elements and has no specific simulation behaviour. Inside the *Composite* element of "Lift Wall Panel", as shown in Figure 4.12.4, simulation entities representing building components capture its required resource (e.g., Crane Crew) based on its resource priority from

the optimization model (see Figure 4.12.3) in order to conduct the work package of "Lift Wall Panel".

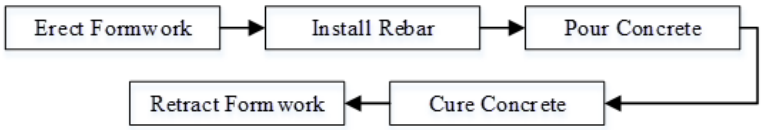


Figure 4.10 Process pattern for cast-in-place building elements

Nevertheless, process pattern can only take into account the local predecessor relationships between activities for one component; the construction logic among different building components discussed in Section 4 cannot be handled by formalized process patterns. For instance, the completion of "Lift Wall Panel" for wall panel # might trigger the activity "Lift Wall Panel" of wall panel #2, and "Erect Formwork Foundation" of foundation #1 might be followed by "Erect Formwork Foundation" of foundation #2. This construction logic is not represented in the above process pattern. In light of such limitation, we develop construction sequence controllers, such as "Controller for Wall Lifting" and "Controller for Floor Assembly", by writing user codes in the "Execute" element. These controllers are in turn placed inside the "Construction Sequencing (Routing Entities)" *Composite* element shown in Figure 4.12. It performs construction sequence reasoning by factoring in the enriched product information such as topological/connection from BIM. Figure 4.11 presents the flowchart of "Controller for Wall Lifting". In general, the controller can manage the construction sequence and route the entities representing building components in accordance with the enriched information it contains. For example, once an activity which would be the potential predecessor of "Lift Wall Panel" for wall panel #N, is completed, such as the "Lift Wall Panel" of its connected wall panel #N-1, "Controller for Wall Lifting" is triggered by the completed activity in order to check all work

package state information and launch subsequent activities, which satisfies the construction conditions. This would include “Lift Wall Panel” for #N wall panel.

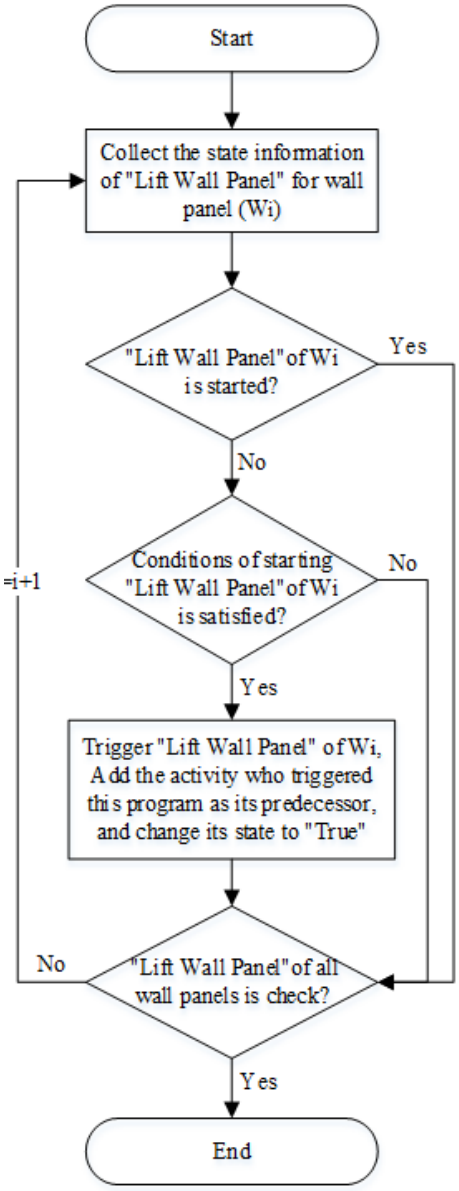
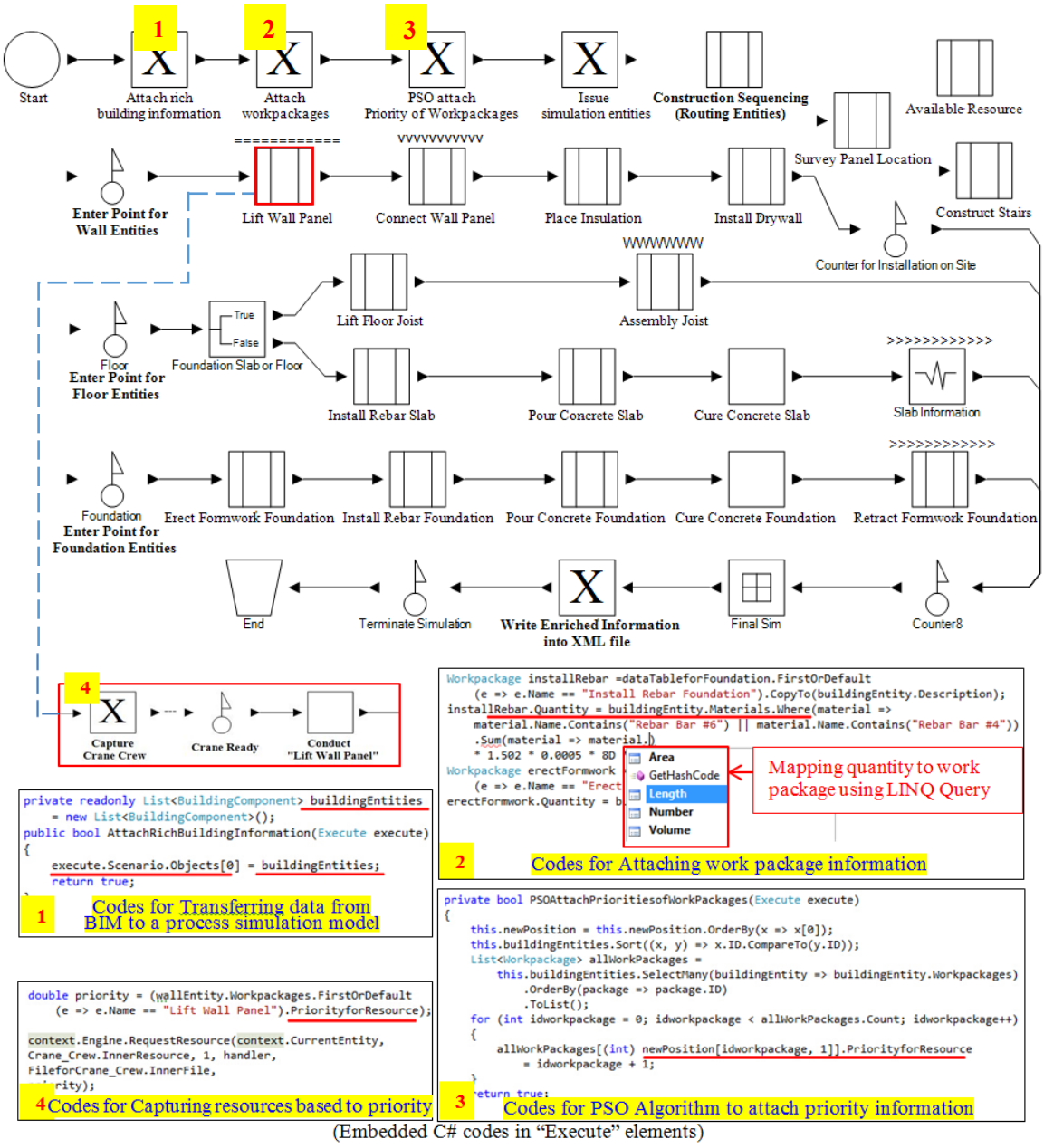


Figure 4.11 Flowchart of “Controller for Wall Lifting”

It should be noted that, for a given activity, construction conditions mainly refer to the completion of corresponding activities of all of its support elements and one of its connected wall panels. The simulation entity standing for #N wall panel is looped into “Enter Point for

Wall Panel", as shown in Figure 4.12, thereby constructing the panel through a series of activities. In this way, the limitation of the fixed activity network diagram explained above and dynamic precedence constraints (e.g., physical constraints) of activities is addressed by the fact that the controller routes entities and triggers construction activities in a manner satisfying precedence constraints without an explicit direction arrow, as indicated by the incomplete network in the simulation model.



(Embedded C# codes in "Execute" elements)

Figure 4.12 Simulation model for on-site construction of panelized building projects

4.5.5 PSO algorithm

The PSO algorithm is employed in this research to optimize the construction schedule by taking advantage of the process simulation in calculating fitness value. Generally speaking, each particle in PSO is represented by a *D*-dimensional vector, $X_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T$ (where D is identical to the dimension of the search space and i represents the index of particles), and its

position is the potential solution to the problem being optimized. PSO is an iteration process, during which particles update their states (velocity and position) to approach the optimum solution. At the outset PSO uses a random population (referred to as a swarm of particles) of potential solutions to the problem in order to explore optimal solutions in the search space. Each particle keeps a record of the best position $\mathrm{P}_i = (p_{\mathrm{i1}}, p_{\mathrm{i2}}, \dots, p_{\mathrm{id}})^T$ it has reached (referred to as the local best solution), the best position $\mathrm{P}_g = (p_{\mathrm{g1}}, p_{\mathrm{g2}}, \dots, p_{\mathrm{gd}})^T$(where g denotes the index of the particle in the swarm) of the best particle (referred to as the global best solution), and its velocity, represented by another *n*-dimensional vector $\mathrm{V}_i = (v_{\mathrm{i1}}, v_{\mathrm{i2}}, \dots, v_{\mathrm{in}})^T$. Once the local best P_i and global best P_g are obtained by calculating the fitness measure of the objective function being studied, each particle in the PSO updates its velocity and position based on its own/local best position, the swarm's best solution, and its previous velocity vector, in accordance with Equation (1) and Equation (2) (Eberhart & Kennedy 1995). Each particle then initiates another iteration process to approach the optimum position gradually.

$$v_{id}^{n+1} = wv_{id}^{n} + c_1 r_1 (p_{id}^{n} - x_{id}^{n}) + c_2 r_2 \left(p_{gd}^{n} - x_{id}^{n}\right) \qquad \text{Eq. (1)}$$

$$x_{id}^{n+1} = x_{id}^{n} + v_{id}^{n+1} \qquad \text{Eq. (2)}$$

where *v* and *x* are the particle's velocity and position, respectively, *n* denotes the n^{th} iteration; c_1 and c_2 are positive constants, called acceleration constants; r_1 and r_2 are random numbers, uniformly distributed in [0,1]; and p_{id} and p_{gd} are, respectively, the local best solution and global best solution mentioned above.

The PSO algorithm is programmed and embedded into Revit as an add-on tool. The detailed implementation of the PSO algorithm in C# is provided in Appendix D. PSO parameters are set as follows, in consideration of the existing research (Eberhart & Kennedy, 1995; Eberhart, 2013;

Lu et al., 2006; Lu et al., 2008; Zhang et al., 2006; Zhang et al., 2006): (1) $c_1 = 1$; (2) $c_2 = 2$; (3) the value of w is 0.9 initially, and then it linearly decreases to 0.4 at the maximum number of iterations; and (4) swarm size = 40. It is anticipated that these parameter settings could reduce the optimizing time and ensure the convergence of the PSO algorithm. Notably, solely applying this algorithm is not the main focus and novelty of the present research, the detailed explanations of PSO parameters would draw attention away from BIM–Simulation integration. Accordingly, detailed explanations of the PSO algorithm parameters are not given in this chapter; instead the reader is directed to previous PSO-related studies.

4.5.6 Information exchanges

An enriched information entity is an object or instance of a class in the object-oriented programming domain. To achieve the proposed integration, two classes, as shown in Figure 4.13, are used as the templates for building components and work packages. These templates are defined in Visual Studio C# and are used as the basis for data exchange among different components of this research. Figure 4.14 presents sample data of one building component extracted from the BIM model and presented in XML format. Revit API is used to extract the rich building element information and feed it to SSRAnalyzer in order to obtain "Connections" and "Supports" information of building components. All enriched product information is then transferred to the process simulation model through an "Attach rich building information" simulation element (see Figure 4.12.1) by means of C# codes embedded in the Revit add-on. Afterward, resource requirements and productivity information of each activity is extracted from MS Access and attached to the enriched information entities by C# codes embedded in the "Attach workpackages" element (see Figure 4.12.2). Subsequently, priority information of work packages from the PSO algorithm is also attached to the enriched information entities by means

of additional C# codes as shown in Figure 4.12.3 Finally, the enriched information entities (objects of the building component and work package classes) will move through the process simulation model in Simphony. The schedule is automatically generated and incorporated into the entities by the process simulation model. At the end of the simulation model, all information carried by the entities is written into an XML file. An add-on tool for MS Project is developed to display the generated schedule by parsing the XML file.

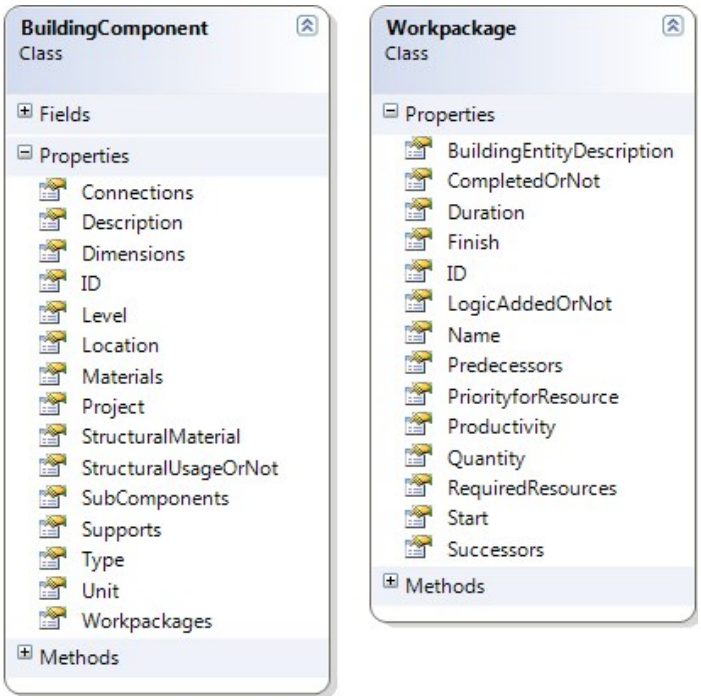


Figure 4.13 Classes for building components and work packages

Quantity take-offs, as part of enriched information extracted from the BIM model, are organized into the enriched information entities under “Materials” and “Dimensions” attributes as shown in Figure 4.13 and Figure 4.14. This research allows for executing queries over defined schemas (information exchanges template) in order to establish the mapping between quantities and activities, using Language-Integrated Query (LINQ). As denoted in Figure 4.12.2, the quantity for the “Install Rebar Foundation” work package is assigned to be the sum of the material

quantities of “Rebar Bar #6” and “Rebar Bar #8” in one building component. The measurement unit of quantities can be selected in order to match the productivity information. Formulating queries for retrieving activity quantity, instead of assigning individual quantity data, can improve accuracy and reduce the workload to update data in the case of design changes. Compared with relational database based quantity take-off assignment in previous research, it avoids the redundant manipulation (e.g., read and write) of external relational database through Structured Query Language (SQL). Meanwhile, LINQ is a common querying syntax that applies across different data storage types such as Objects and SQL Database Tables, thus allowing the execution of queries without the knowledge of specific database languages (e.g., MS Access SQL Language and MS SQL Server Language). Also, querying data in computer memory is superior to querying data in external relational system in terms of the computing performance of developed scheduling system.

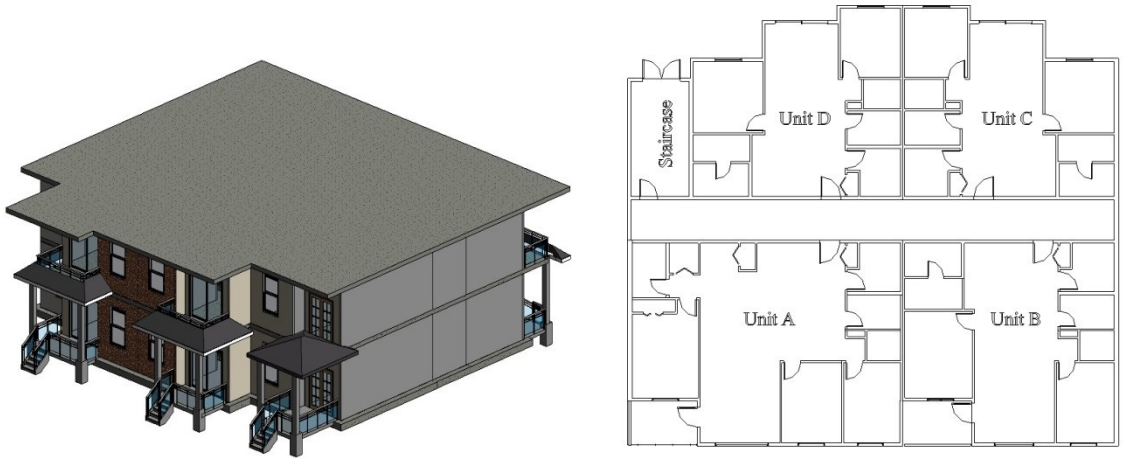
```
<BuildingComponent>
  <ID>511250</ID>
  <Type>Structural Foundations</Type>
  <Unit>NaN</Unit >
  <Level>NaN</Level >
  <Description> Structural Foundation 511250</Description>
  <StructuralUsageOrNot>true</StructuralUsageOrNot>
  <Location>
   <X>-520.710953961512</X>
   <Y>90.286826646650525</Y>
   <Z>2.5</Z>
  </Location>
  <StructuralMaterial>Concrete,
  <Dimensions>        Cast-in-Place gray</StructuralMaterial>
      <string>Volume</string>
      <double>33.16977984762061</double>
      <string>Width</string>
      <double>2.9527559055118111</double>
      <string>Length</string>
      <double>14.822316864990711</double>
      <string>Lateral area</string>
      <double>11.233498775060847</double>
  </Dimensions>
  <Materials>
   <NameandQuantityofMaterial>
    <Name>Concrete, Cast-in-Place gray</Name>
    <Volume>33.16977984762061</Volume>
    <Area>134.30612553700078</Area>
    <Length>NaN</Length>
    <Number>1</Number>
   </NameandQuantityofMaterial>
   <NameandQuantityofMaterial>
    <Name>Structural Rebar #6 : Shape M_00</Name>
    <Volume>0.10362825329567779</Volume>
    <Area>NaN</Area>
    <Length>75.9995078675841</Length>
    <Number>3</Number>
   </NameandQuantityofMaterial>
   <NameandQuantityofMaterial>
    <Name>Structural Rebar #4 : Shape M_00</Name>
    <Volume>0.03802518721792679</Volume>
    <Area>NaN</Area>
    <Length>27.887139107609755</Length>
    <Number>10</Number>
   </NameandQuantityofMaterial>
  </Materials>
  <Connections>
   <int>511528</int>
   <int>536245</int>
  </Connections>
  <Supports />
  <Workpackages />
  <SubComponents />
 </BuildingComponent>
```

Figure 4.14 Sample data extracted from 3D BIM model

4.6 Demonstration

Part of a residential building is selected in order to test the scheduling system embedded in Revit. The building consists of two storeys, each with four apartment units and one staircase. Each apartment unit has two washrooms where the assembly work commences. Additionally, there are 182 panels, including sixty non-bearing walls and 122 bearing walls. The building rests on twenty-nine concrete wall footings. The BIM model for the building is developed in Autodesk

Revit, as shown in Figure 4.15. It is noteworthy that the washroom module, as described above, is a composite of several wall panels and floors, such that the "Group" function in Revit is utilized to model washrooms by grouping the necessary components as one single washroom component. Later, washroom elements can be recognized by the system in order to commence the construction work on each floor. Work package information, such as the resource requirements and productivity of each building component, is listed in Table 4.1, while Table 4.2 tabulates the available resources for the project. All the data has been provided by the construction manager and is used as the inputs (stored in MS Access) to the system. The start date of the project is set as November 6, 2013, and a standard workweek which runs Monday to Friday is assumed in the project, with each day starting at 8 a.m. and ending at 5 p.m.



a. 3D view b. Plan view

Figure 4.15 a two-storey residential building

Table 4.1 Durations, productivity, and resources

Component	*Activities*	*Required Resources*	*Productivity*	*Duration (min)*
Wall Panel	Survey Panel Location	1 Survey Crew	1panel/T(8, 12, 10) min	-
	Lift Wall Panel	1 Crane Crew	1panel/T(9, 15, 12)min	-
	Connect Wall Panel	1 Frame Crew 1 Crane Crew	0.25 connections /min	-
	Place Insulation	1 Insulation Crew	0.5 ea/min	-
	Install Drywall	1 Drywall Crew	10 s.f./min	
	Frame Wall Panel	1 Frame Crew	4 studs/min	-
Slab	Install Rebar Slab	1 Reinforcement Crew	29 C.S.F/day	-
	Pour Concrete Slab	1 Concrete Crew	55 cy^3/day	-
	Cure Concrete Slab			U(50, 56)*60
Foundation	Erect Formwork Foundation	1 Formwork Crew	300 s.f./day	-
	Install Rebar Foundation	1 Reinforcement Crew	2.1 ton/day	-
	Pour Concrete Foundation	1 Concrete Crew	11 cy^3/day	-
	Cure Concrete Foundation		-	U(50, 56)*60
	Retract Formwork Foundation	1 Formwork Crew	300 s.f./day	-
Floor	Lift Joists	1 Crane Crew	8 joists/T(12, 18, 15) min	
	Assembly Joist	1 Frame Crew	1 joist/T(16, 20, 18) min	
Stairs	Lift Landing Pieces	1 Crane Crew	1 landing/T(12, 18, 15) min	
	Frame Landing	1 Frame Crew	1 landing/T(50, 70, 60) min	
	Lift Stairs Panel	1 Crane Crew	1 panel/T(9, 15, 12) min	
	Assemble Stairs Panel	1 Frame Crew 1 Crane Crew	1 panel/T(40, 60, 50) min	

Note: For T(N1,N2,N3), T = Triangular, N1 = lower limit, N2 = upper limit, and N3 = mode value; and for U(N1, N2), U = Uniform, N1 = lower limit, N2 = upper limit.

Table 4.2 Available resources

Resource	*Quantity*	*Resource*	*Quantity*
Frame Crew	1	Drywall Crew	1
Formwork Crew	1	Reinforcement Crew	1
Survey Crew	1	Concrete Crew	1
Crane Crew	1	Insulation Crew	1

It should be pointed out that the prototype system encompassing the WBS database in Access and the DES model in Simphony is developed in accordance with the construction method commonly adopted in panelized construction (using LGS system). In order to apply it to a new project, work package information shown in Table 4.1 and Table 4.2 along with a process

simulation model should be modified through the add-on user interface as inputs of the proposed system (see Figure 4.6) in the case that a different construction method is applied in the new project.

The optimized project duration for the case example is 30.67 calendar days, starting November 06, 2013 and ending December 06, 2013. Figure 4.16 shows part of the generated schedule from the scheduling system, which is automatically exported into MS Project. As illustrated in the figure, the completion of "*Level 1 Floor 456163 Curing Slab*" (marked as "1" in the figure) is followed by the start of "*Level 1 Bathroom 521569 Survey Panel Location*" (shown as "2" in the figure). The completion of "*Level 1 Bathroom 521569 Survey Panel Location*" triggers the start of "*Level 1 Bathroom 521569 Lift Wall Panel (Bathroom)*", "*Level 1 Unit B 538654 Survey Panel Location*", and "*Level 1 Unit B 476373 Survey Panel Location*" (marked as "3' in the figure), and so on. All these replicate the construction logic described in Section 4. For example, the on-site work for wall panels at the same level as the washroom always begins with the installation of the washroom module, and the next components to be constructed are the wall panels, which have a connection with the wash-room module. Hence, the generated schedule demonstrates the feasibility of the prototype system.

Additionally, Figure 4.17 shows the evolutionary process of one particle and the entire swarm in one experiment. As noted in the figures, the maximum project duration for the project in the experiment is 35.37 days (50,933.89 min), and it gradually approaches 30.67 days (44,162.89 min) over 100 optimization iterations, meaning that the project duration is shortened by 4.70 days (15.33%). The selection of parameters discussed in Section 5 ensures the convergence of the PSO algorithm but at the same time may lead the solution toward a local optimum. Therefore, the optimized solution of 30.67 days may not be the global optimum. There are sharp changes in

project duration in the evolutionary process, caused by imposing working calendars. This is due to the fact that the duration is shortened by more than just the two working days in the span from Tuesday to the previous Friday, since this span includes two calendar days of non-working time.

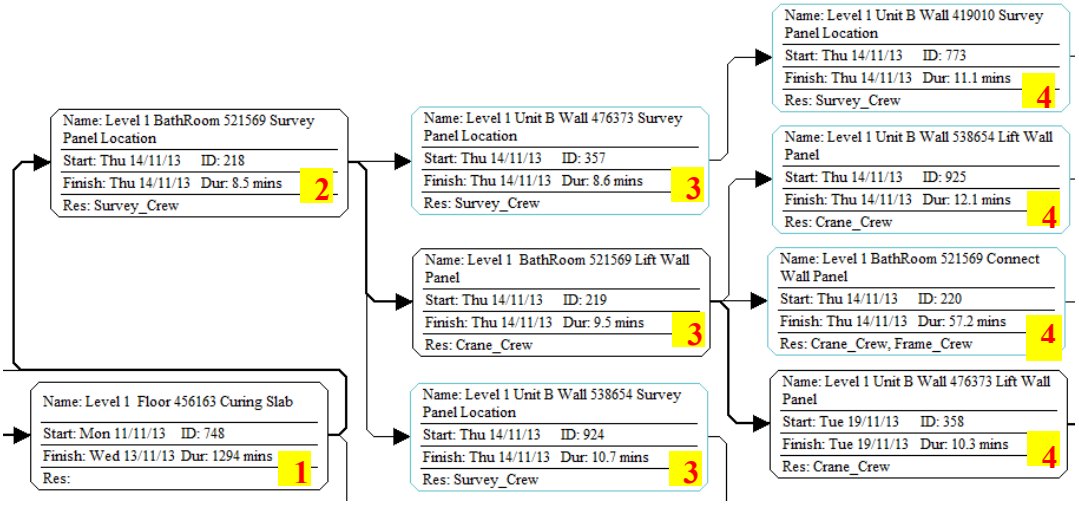


Figure 4.16 Part of generated schedule from scheduling system

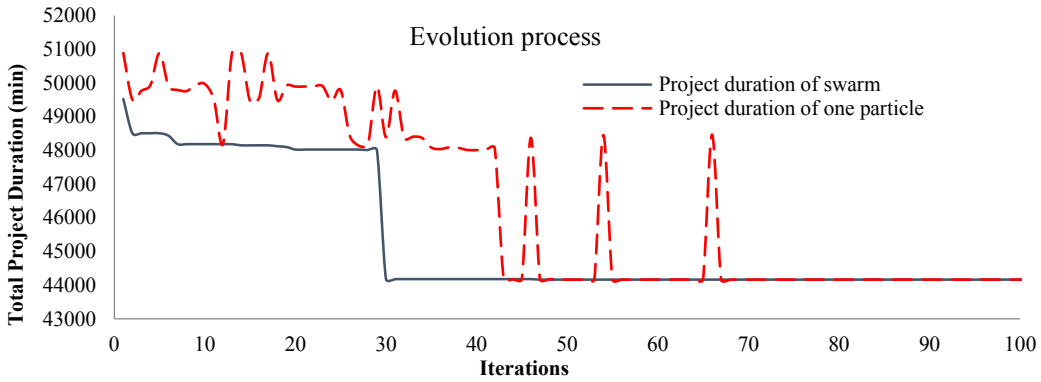


Figure 4.17 Project duration evolution process during optimization

4.7 Discussion

Activities/processes in panelized construction should distinguish each individual building component, instead of distinguishing each construction zone, in that each pre-fabricated component is unique and needs to be installed at its own designed location and be scheduled

individually in order to manage and coordinate factory production and on-site construction processes. This detailed scheduling is defined as component-centric activity level scheduling which is the focus of this research. Establishing a complete activity network manually in component-centric activity level scheduling, is a challenge for current construction planning methodologies (such as critical path method and Primavera P6), due largely to the practical needs of adjusting construction technology under physical and spatial constraints in the field and dynamic precedence constraints between activities caused by resource allocation strategies. This research demonstrates a methodology for the planner to build part of the activity network manually as per constraints that must be accommodated during construction, instead of a complete activity network as in previous research. Those dynamic precedence constraints on activities are derived at run time of DES, whereas resource-induced precedence constraints are addressed by using priority dispatching rules to allocate limited resources. As a result, the simulation model in this research does not have a complete network; in other word, there are no explicit direction arrows navigating simulation entities through simulation modelling blocks. As such, the flexible and dynamic precedence constraints caused by different resource allocation strategies are considered in construction planning.

In short, the proposed methodology addresses the difficulty in manually building a complete activity network through in-depth BIM-Simulation integration and overcomes the limitation of defining a fixed activity network in construction project planning. This work also demonstrates that (1) enriched information exchange between the BIM model and the process simulation model is feasible via the development of an information exchange template; (2) enriched product information from the 3D BIM model is indispensable for the simulation model to mimic construction processes in the real world of panelized building construction."

4.8 Conclusion

Since BIM is increasingly utilized within the construction industry, this research has proposed a BIM-based integrated approach for detailed construction scheduling under resource constraints. This capitalizes on the benefits of rich building information in BIM and the capability of DES to mimic the construction operation logic and investigate the allocation of available resources among activities. In-depth integration among a BIM/project model in Revit, WBS information in MS Access, process simulation model in Simphony, and an evolutionary optimization algorithm has been achieved in the proposed methodology in order to automatically generate an optimized construction schedule. Furthermore, a prototype scheduling system for panelized LGS construction has been developed as an Autodesk Revit add-on which is able to produce MS Project-based schedules in order to facilitate communication among project stakeholders and support project management on site. The scheduling system is generally able to produce expected schedules for panelized construction, and assists project managers in effectively planning on-site assembly work by reducing the human error in scheduling for panelized construction, which also validates the integrated methodology. However, the current prototype system has limitations in the following respects: (1) duration and productivity estimates for work packages are made based on project managers' experience; (2) other factors affecting construction schedule, such as weather and work space limitations, are not taken into consideration; and (3) part of the simulation network still needs to be established manually. In order to improve the performance of the current system and achieve fully automated schedule generation, the following directions can be pursued in the future:

1) Time studies on construction processes can be conducted, and data mining technology such as Artificial Neuron Network (ANN) can be integrated into the proposed system to predict method productivity based on historical data.
2) Ontology-based construction knowledge modelling can be studied in order to fully automate simulation model generation with the support of enriched building information from a BIM model.
3) Optimization of construction schedules with respect to time, cost, resource use and material logistics based on a sufficient BIM-simulation integrated project model can be explored.

CHAPER 5: CONCLUSIONS

5.1 Summary

Panelization has emerged as a popular, more efficient approach to constructing residential projects with the rise of Building Information Modelling (BIM). In order to advance the current planning practice of panelized construction, this research incorporates construction-oriented intelligence (i.e., trades know-how) into current BIM design models in order to facilitate automated panelized construction planning.

First, this research automates the building design and modelling in terms of boarding design in order to achieve manufacturing-centric BIM and to adapt discipline-specific BIM design models (e.g., architectural model) for use by construction trades. The prototyped design application eliminates the guesswork and saves a large amount of time in boarding design, building information modelling and raw material cutting planning for construction engineers. The automated approach can generate optimized boarding layout design, which reduces construction material waste in that mathematical algorithms and design-rules (e.g., trades know-how) are integrated with BIM design models. Along with the optimized layout design, the prototyped application can also formulate the material cutting plan that is instrumental in guiding field engineers to perform their work.

Also, this research investigates semantic technologies to extract domain-specific data from a common BIM repository, thereby expediting the QTO process. In the proposed approach, the implicit BIM data crucial to construction practitioners is derived and extracted such that it enables QTO professionals to take off the implicit BIM features on the basis of existing design-oriented BIM models, which improves the QTO efficiency. All relevant BIM data are transformed into the ontology-enhanced BIM model in an RDF file. The resulting RDF model

conceptually functions as a domain-specific "model view" of the given BIM model while enabling semantic queries to facilitate construction-oriented QTO. Hence, construction practitioners can semantically query a BIM design model in order to generate QTO for construction activities by using their domain vocabularies, without the need to understand the technical structure of the underlying complex BIM schema. In addition, the proposed semantic QTO approach sheds light on other research endeavors in terms of semantic enrichment for BIM to support domain-specific tasks and semantic interoperability among BIM applications.

Lastly, this research develops an automated on-site scheduling application for panelized construction by achieving an in-depth integration between BIM, DES, and evolutionary optimization. It demonstrates a methodology for the planner to build part of the activity network manually as per constraints that must be accommodated during construction, instead of a complete activity network as in previous research. Those dynamic precedence constraints on activities are derived at run time of DES, whereas resource-induced precedence constraints are addressed by using priority dispatching rules to allocate limited resources. The proposed scheduling approach is able to schedule each building panel individually. Consequently, it not only serves as the base for project managers to arrange the on-site construction, but also provides the guides for them to plan the factory production in order to deliver individual building panels as expected.

This research enriches the application of BIM technology in light-frame building construction in terms of three aspects. Three long-standing, ill-structured problems in the construction industry—boarding layout design, quantity take-off, and panelized scheduling—are formulated into structured ones. Three novel approaches are then proposed to solve three practical problems in a scientific manner. Although these three practical problems are addressed individually,

proposed methodologies and prototyped computer systems are used together to create an approach that is more robust than the applying the component systems individually. For instance, other domain terms can be further formalized into the proposed product ontology in order to store the construction process and process pattern information as part of the ontology-enhanced BIM model. As a result, construction practitioners can retrieve this process and process pattern information using their vocabularies in a straightforward manner, resulting in better communication among project stakeholders and fully automated construction scheduling.

5.2 Research Contributions

The primary contributions of this research are summarized as follows:

1) Automation of the process in optimizing board layout design and planning board cutting by taking advantage of rich building information in BIM models and integrating comprehensively formalized industry know-how in terms of boarding practice and mathematical algorithms with BIM models.
2) Introduction of an ontology-based semantic framework for construction-oriented QTO that enables construction practitioners to retrieve the QTO information in a flexible manner.
3) Establishment of a construction-oriented product ontology, which extends current design BIM models by adding domain terms and their properties and interrelationships without changing the original BIM schema (i.e., supplementing domain semantic into BIM design models), and aligns design BIM models with construction process oriented QTO.
4) Development of in-depth integration between BIM, DES, and evolutionary optimization for panelized construction on-site scheduling. The methodology addresses the existing challenges with respect to automatic detailed construction planning under resource

constraints. Rich building information extracted from the BIM model supports not only the activity duration calculation, but more importantly the simulation logic.

5) Addressing dynamic precedence constraints due to physical and spatial constraints in the field and resource allocation strategies through an in-depth BIM-simulation integration for on-site scheduling in panelized construction.
6) Development of three add-ons of Autodesk Revit to automate the processes of manufacturing-centric BIM (e.g., boarding layout design), construction-oriented QTO, and panelized construction scheduling, respectively.

5.3 Limitations and Future Research

In order to improve the performance of the proposed method and prototyped system, the following directions can be pursued in the future:

1) Other combinatorial algorithms, instead of greedy algorithms, can be investigated and incorporated into the boarding design prototype system in order to optimize boarding design more efficiently.
2) SWRL-based ontology reasoning can be investigated to further provide semantics to the QTO prototype application in the future.
3) Time studies on construction processes can be conducted, and data mining technology such as Artificial Neuron Network (ANN) can be integrated into the proposed scheduling system to predict method productivity based on historical data.
4) Ontology-based construction knowledge modelling can be studied in order to automate simulation model generation for the purpose of fully automated construction scheduling with the support of rich building information from a BIM model.

5) On-site oriented building panel production line planning and management system can be explored to facilitate the production line planning and management in panelized construction. This system is expected to assist the practitioners to match the productivity of the factory production to the on-site construction productivity, which leads to an entire Just-in-Time management structure for the industry company.

REFERENCES

AbouRizk, S. (2010). Role of simulation in construction engineering and management. Journal of Construction Engineering and Management, 136(10), 1140-1153.

AbouRizk, S., Halpin, D., Mohamed, Y., and Hermann, U. (2011). Research in modeling and simulation for improving construction engineering operations. Journal of Construction Engineering and Management, 137, SPECIAL ISSUE: Construction Engineering: Opportunity and Vision for Education, Practice, and Research, 843-852.

Alwisy, A., and Al-Hussein, M. (2010). Automation in drafting and design for modular construction manufacturing utilizing 2D CAD and parametric modeling. In Proc., Computing in Civil and Building Engineering, Proc., Int. Conf.

Alwisy, A., Al-Hussein, M., and Al-Jibouri, S. H. (2012). BIM approach for automated drafting and design for modular construction manufacturing. Computing in civil engineering (2012), 221-228.

Aryanezhad, M. B., Hashemi, N. F., Makui, A., and Javanshir, H. (2012). A simple approach to the two-dimensional guillotine cutting stock problem. Journal of Industrial Engineering International, 8(1), 1-10.

Association of School Business Officials (2013). BIM Resource Guide: A Guide for Implementing Building Information Modeling in State of Maryland and Washington DC Public School Construction Projects.

Autodesk Ltd. (2015). Autodesk Revit solution. Available at: http://www.autodesk.com/products/revit-family/overview. (Aug. 21, 2015)

Autodesk Ltd. (2015). Autodesk Revit help. Available at: http://help.autodesk.com/view/RVT/2015/ENU/?guid=GUID-22D24055-61A2-40BB-A2F7-A37990300B2B. (Aug. 21, 2015)

Autodesk Ltd. (2014). Autodesk Revit API help. Available at: http://help.autodesk.com/view/RVT/2014/ENU/?guid=GUID-B5E019A8-02F1-49A2-9EB8-449FB99D1E7C. (Aug. 21, 2015)

Beetz, J., Van Leeuwen, J., and De Vries, B. (2009). IfcOWL: A case of transforming EXPRESS schemas into ontologies. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 23(01), 89-101.

Benevolenskiy, A., Roos, K., Katranuschkov, P., and Scherer, R. J. (2012). Construction processes configuration using process patterns. Advanced Engineering Informatics, 26(4), 727-736.

Borrmann, A., and Rank, E. (2009a). Topological analysis of 3D building models using a spatial query language. Advanced Engineering Informatics, 23(4), 370-385.

Borrmann, A., Schraufstetter, S., and Rank, E. (2009b). Implementing metric operators of a spatial query language for 3D building models: octree and B-rep approaches. Journal of Computing in Civil Engineering, 23(1), 34-46.

California Integrated Waste Management Board (2007). Wallboard (drywall) recycling. <www.ciwmb.ca.gov> (March 17, 2009).

Cavieres, A., Gentry, R., and Al-Haddad, T. (2011). Knowledge-based parametric tools for concrete masonry walls: Conceptual design and preliminary structural analysis. Automation in Construction, 20, 716-728.

Chau, K.W., Anson, M., and Zhang, J.P. (2005). 4D dynamic construction management and visualization software: 1. Development. Automation in Construction, 14, 512-524.

Chen, S.M., Griffis, F.H., and Chen, P.H., and Chang, L.M. (2013). A framework for an automated and integrated project scheduling and management system. Automation in Construction, 35, 89-110.

Cheng, J. C., and Ma, L. Y. (2013). A BIM-based system for demolition and renovation waste estimation and planning. Waste management, 33(6), 1539-1551.

Choi, J., Kim, H., and Kim, I. (2015). Open BIM-based quantity take-off system for schematic estimation of building frame in early design stage. Journal of Computational Design and Engineering, 2(1), 16-25.

COAA. (2014). About Advanced Work Packaging (AWP). Available at: http://www.coaa.ab.ca/construction/AWPWFP/AWPWFPOverviewandDefinitions.aspx. (Aug. 16, 2016)

Da Costa, F. M., and Sassi, R. J. (2012). Application of a hybrid bio-inspired meta-heuristic in the optimization of two-dimensional guillotine cutting in a glass industry. In Intelligent Data Engineering and Automated Learning-IDEAL 2012 (pp. 802-809). Springer Berlin Heidelberg.

Daum, S., and Borrmann, A. (2014). Processing of Topological BIM Queries using Boundary Representation Based Methods. Advanced Engineering Informatics, 28(4), 272-286.

De Vries, B. and Harink, J. (2007). Generation of a construction planning from a 3D CAD model. Automation in Construction, 16(1), 13-18.

DuCharme, B. (2011). Learning SPARQL. Sebastopol, CA 95472, USA, 2011.

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

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. Proceedings, Sixth International Symposium on Micro Machine and Human Science (MHS'95). Nagoya, Japan, 39-43.

Elbeltagi, E. (2013). 20-swarm intelligence for large-scale optimization in construction management. Metaheuristic Applications in Structures and Infrastructures, 479-495.

Esparza, J. (2003). Greedy algorithms. Available at http://www.dcs.ed.ac.uk/teaching/cs1/CS1/Bh/Notes/Greedy.pdf. (Apr. 30, 2016)

Fischer, M., and Aalami, F. (1995). Scheduling with computer-interpretable construction method models. Journal of Construction Engineering and Management, 122(4), 337-347.

Formoso, C. T., Soibelman, L., De Cesare, C., and Isatto, E. L. (2002). Material waste in building industry: main causes and prevention. Journal of construction engineering and management, 128(4), 316-325.

Gane, V., and Haymaker, J. (2012). Design Scenarios: Enabling transparent parametric design spaces. Advanced Engineering Informatics, 26(3), 618-640.

Gilmore, P. C., and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. Operations research, 9(6), 849-859.

Gordian Group (2015), RS Means online, Available at: https://www.rsmeansonline.com/. (Aug. 17, 2015)

Gray, C. (1986). "Intelligent" construction time and cost analysis. Construction Management and Economics, 4(2), 135-150.

Gruber, T. R. (1993). A translation approach to portable ontology specifications, Knowledge acquisition 5.2: 199-220. Available at: http://www.dbis.informatik.huberlin.de/dbisold/lehre/WS0203/SemWeb/lit/KSL-92-17.pdf. (Apr. 22, 2015)

Holm, L., Schaufelberger, J., Griffin, D., and Cole, T. (2005). Construction Cost Estimating Process and Practices. Prentice Hall.

Home Innovation Research Labs (2001). Residential Construction Waste: From Disposal to Management. Available at: http://www.toolbase.org/Best-Practices/Construction - Waste/residential-construction-waste. (Jan. 26, 2015)

Hong, T., Cho, K., Hyun, C., and Han, S. (2011). Simulation-based schedule estimation model for ACS-based core wall construction of high-rise building. Journal of Construction Engineering and Management, 137(6), 393-402.

Howard, R. and Björk B.C. (2008). Building information modelling: Experts' views on standardisation and industry deployment. Advanced Engineering Informatics, 28(2), 271-280.

Hu, D. and Mohamed, Y. (2010). State-based simulation mechanism for facilitating project schedule updating. Proceedings, Construction Research Congress, ASCE, Banff, AB, Canada, May 8-10, 369-378.

Hu, Z., Zhang, J., and Zhang, X. (2010). 4D construction safety information model-based safety analysis approach for scaffold system during construction. Engineering Mechanics. 27(12), 192-200 (in Chinese).

Jeong, S. K., and Ban, Y. U. (2011). Computational algorithms to evaluate design solutions using Space Syntax. Computer-Aided Design, 43(6), 664-676.

Kaner, I., Sacks, R., Kassian, W., and Quitt, T. (2008). Case studies of BIM adoption for precast concrete design by mid-sized structural engineering firms.

Kataoka, M. (2008). Automated generation of construction plans from primitive geometries. Journal of Construction Engineering and Management, 134(8), 592-600.

Kim H., Anderson, K., Lee, S., and Hildreth, J. (2013). Generating construction schedules through automatic data extraction using open BIM (building information modeling) technology. Automation in Construction, 35, 285-295.

Kim, S.A., Chin S., Yoon S.W., Shin T.H., Kim Y.H., and Choi C. (2009). Automated building information modeling system for building interior to improve productivity of BIM-based quantity take-off. In Proceedings of the 26th International Symposium on Automation and Robotics in Construction, Austin, TX, USA., pp. 492-496.

Kim, K., and Cho, Y. K. (2015). Construction-specific spatial information reasoning in Building Information Models. Advanced Engineering Informatics, 29(4), 1012-1027.

Kim, H., and Grobler, F. (2009). Design coordination in building information modeling (BIM) using ontological consistency checking. Computing in Engineering, 410-420.

Kim, K., and Teizer, J. (2014). Automatic design and planning of scaffolding systems using building information modeling. Advanced Engineering Informatics, 28(1), 66-80.

Konig, M., Habenicht, I., Koch, C., and Spieckermann, S. (2012). Intelligent BIM-based construction scheduling using discrete event simulation. Proceedings, Winter Simulation Conference, Berlin, Germany, Dec. 9-12, 662-673.

Laquatra, J., and Pierce, M. (2004). Managing waste at the residential construction site. Journal of the solid waste technology and management, 30 (2), 67-74.

Lawrence, M., Pottinger, R., Staub-French, S., and Nepal, M. P. (2014). Creating flexible mappings between Building Information Models and cost information. Automation in Construction, 45, 107-118.

Lee, S.-K., Kim, K.-R., and Yu, J.-H. (2014). BIM and ontology-based approach for building cost estimation. Automation in Construction, 41, 96-105.

Lee, G., Sacks, R., and Eastman, C.M. (2006), Specifying parametric building object behavior (BOB) for a building information modeling system. Automation in Construction 15, 758-776.

Leite, F., Akcamete, A., Akinci, B., Atasoy, G., and Kiziltas, S. (2011). Analysis of modeling effort and impact of different levels of detail in building information models. Automation in Construction, 20(5), 601-609.

Li, H., Chen, Z., and Wong, C. T. (2003). Barcode technology for an incentive reward program to reduce construction wastes. Computer-Aided Civil and Infrastructure Engineering, 18(4), 313-324.

Liu, H., Al-Hussein M., and Lu M. (2015a), BIM-based integrated approach for detailed construction scheduling under resource constraints, Automation in Construction, 53, 29-43.

Liu, H., Altaf, M. S., Lei, Z., Lu, M., and Al-Hussein, M. (2015b). Automated production planning in panelized construction enabled by integrating discrete-event simulation and BIM. Proceedings, International Construction Specialty Conference, pp. 8-10.

Liu, H., Lei, Z., Li, H.X., and Al-Hussein, M. (2014). An automatic scheduling approach: Building information modeling-based on-site scheduling for panelized construction. In Proceedings of the Construction Research Congress, pp. 1666-1675.

Liu, H., Singh, G., Lu, M., and Al-Hussein, M. (2015c). BIM-enabled boarding design optimization for residential buildings. Proceedings, International Conference on Construction Applications of Virtual Reality (CONVR), Banff, AB, Canada, Oct. 5-7.

Liu, H., Lu, M., and Al-Hussein, M. (2016). Ontology-based semantic approach for construction-oriented quantity take-off from BIM models in the light-frame building industry. Advanced Engineering Informatics, 30(2), 190-207.

Liu, Z., Osmani, M., Demian, P., and Baldwin, A. (2015). A BIM-aided construction waste minimisation framework. Automation in Construction, 59, 1-23.

Lu, M. (2003). Simplified discrete-event simulation approach for construction simulation. Journal of Construction Engineering and Management, ASCE, 129(5), 537-546.

Lu, M., Lam, H., and Dai, F. (2008). Resource-constrained critical path analysis based on discrete event simulation and particle swarm optimization. Automation in Construction, 17(6), 670-681.

Lu, M., Wu, D., and Zhang, J. (2006). A particle swarm optimization-based approach to tackling simulation optimization of stochastic, large-scale and complex systems. Advances in Machine Learning and Cybernetics Lecture Notes in Computer Science, 3930, 528-537.

Lu, M., Zhang, Y., Zhang, J., Hu, Z., and Li, J. (2009). Integration of four-dimensional computer-aided design modeling and three-dimensional animation of operations simulation for visualizing construction of the main stadium for the Beijing 2008 Olympic games. Canadian Journal of Civil Engineering, 36(3), 473-479.

Ma, Z., Liu, Z., and Wei, Z. (2015). Formalized Representation of Specifications for Construction Cost Estimation by Using Ontology. Computer-Aided Civil and Infrastructure Engineering, 31(1), 4-17.

Ma, Z., Wei, Z., and Zhang, X. (2013). Semi-automatic and specification-compliant cost estimation for tendering of building projects based on IFC data of design model. Automation in Construction, 30, 126-135.

Ma, Z., Wei, Z., Song, W., and Lou, Z. (2011). Automation in construction application and extension of the IFC standard in construction cost estimating for tendering in China. Automation in Construction, 20(2), 196-204.

Manrique, J. D., Al-Hussein, M., Bouferguene, A., and Nasseri, R. (2015). Automated generation of shop drawings in residential construction. Automation in Construction, 55, 15-24.

Manrique, J. D., Al-Hussein, M., Bouferguene, A., Safouhi, H., and Nasseri, R. (2009). Combinatorial Algorithm for Optimizing Wood Waste in Framing Designs. Journal of Construction Engineering and Management, 137(3), 188-197.

Martinez-Cruz, C., Blanco, I. J., and Vila, M. A. (2011). Ontologies versus relational databases: are they so different? A comparison. Artificial Intelligence Review, 38(4), 271-290.

Mohamed, Y. and AbouRizk, S. (2000). Simphony: An integrated environment for construction simulation. Proceedings, Winter Simulation Conference, Orlando, FL, USA, Dec. 10-13, 1907-1914.

Monteiro, A., and Martins, J. P. (2013) A survey on modeling guidelines for quantity takeoff-oriented BIM-based design. Automation in Construction, 35, 238-253.

Montibelli, A. (2014). Application for solving Bin Packing and Cutting Stock problem. Available at http://www.codeproject.com/Articles/706136/Csharp-Bin-Packing-Cutting-Stock-Solver. (Aug. 09, 2016)

Moon, H., Kim, H., Kamat, V., and Kang, L. (2013). BIM-based construction scheduling method using optimization theory for reducing activity overlaps. Journal of Computing in Civil Engineer-ing, 10.1061/(ASCE) CP.1943-5487.0000342 (Jul. 10, 2013).

National Association of Home Builders. (2009). "Fast facts for panelized homes." Available at: http://www.nahb.org/generic.aspx?genericContentID=10310 (Oct. 24, 2013).

National BIM Standard, United States. National Building Information Model Standard Project Committee. Available at: http://www.national-bimstandard.org/faq.php#faq1. (Nov. 20, 2013).

National Association of Home Builders (1999), Construction Waste Estimate of a Typical 2000-Sq-Ft House. Available at http://www.calrecycle.ca.gov/Publications/Documents/GreenBuilding/43199009D.doc. (Jun. 15, 2016)

Nepal, M. P., Staub-French, S., Pottinger, R., and Webster, A. (2012). Querying a building information model for construction-specific spatial information. Advanced Engineering Informatics, 26(4), 904-923.

Nepal, M. P., Staub-French, S., Pottinger, R., and Zhang, J. (2013). Ontology-based feature modeling for construction information extraction from a Building Information Model. Journal of Computing in Civil Engineering, 27(5), 555-569.

Noy, N. F., and McGuinness, D. L. (2000). Ontology development 101: a guide to creating your first ontology, 1-25.

Oh, M., Lee, J., Hong, S. W., and Jeong, Y. (2015). Integrated system for BIM-based collaborative design. Automation in Construction, 58, 196-206.

Porwal, A., and Hewage, K. N. (2011). Building Information Modeling-Based Analysis to Minimize Waste Rate of Structural Reinforcement. Journal of construction engineering and management, 138(8), 943-954.

Powers, S. (2003). Practical rdf. "O'Reilly Media, Inc."

Protégé (2014). A free, open-source ontology editor and framework for building intelligent systems. Available at: http://protege.stanford.edu/. (Nov. 23, 2015)

Ramaji, I. J., and Memari, A. M. (2016). Product Architecture Model for Multistory Modular Buildings. Journal of Construction Engineering and Management, 04016047.

Sacks, R., and Barak, R. (2008). Impact of three-dimensional parametric modeling of buildings on productivity in structural engineering practice. Automation in Construction, 17(4), 439-449.

Sacks, R., Eastman, C.M., and Lee, G. (2004). Parametric 3D modeling in building construction with examples from precast concrete, Automation in Construction 13, 291-312.

Sacks, R., Eastman, C. M., Lee, G., and Orndorff, D. (2005). A target benchmark of the impact of three-dimensional parametric modeling in precast construction. PCI journal, 50(4), 126.

Salman, A., Ahmad, I., and Al-Madani, S. (2002). Particle swarm optimization for task assignment problem. Microprocessors and Microsystems, 26(8), 363-371.

Sattineni, A., and Bradford II, R. H. (2011). Estimating with BIM: a survey of US construction companies, Proceedings of the 28th International Symposium on Automation and Robotics in Construction and Mining, 564-569.

Siricharoen, W. V. (2007). Ontologies and object models in object oriented software engineering. IAENG International Journal of Computer Science, 33(1), 19-24.

Staub-French S., Fischer M., and Spradlin M. (1999). Into the fourth dimension. Civil Engineering, 69(5), 44-47.

Staub-French, S., Fischer, M., and Kunz, J. (2002). An ontology for relating features of building product models with construction activities to support cost estimating, Center for Integrated Facility Engineering Working Paper No. 70.

Staub-French, S., Fischer, M., Kunz, J., Ishii, K., and Paulson, B. (2003). A feature ontology to support construction cost estimating. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 17(02), 133-154.

StrucSoft Solutions Ltd. (2014). Metal wood framer (MWF) add-on. Available at: http://www.strucsoftsolutions.com/. (Aug. 31, 2015)

StrucSoft Solutions Ltd. (2015). Metal Wood Framer (MWF) add-on. Available at: http://www.strucsoftsolutions.com/. (Apr. 25, 2016)

Taghaddos, H., AbouRizk, S., Mohamed, Y., and Hermann, R. (2009). Integrating simulation-based scheduling for module assembly yard. Proceedings, Construction Research Congress, Seattle, WA, USA, Apr. 5-7, 1270-1279.

Taghaddos, H., Hermann, U., AbouRizk, S., and Mohamed, Y. (2012). A simulation-based multi-agent approach for scheduling modular construction. Journal of Computing in Civil Engineering, 28(2), 263-274.

Tauscher, E., Mikulakova, E., Beucke, K., and König, M. (2009). Automated generation of construction schedules based on the IFC object model. In International workshop on computing in civil engineering. ASCE, Austin (pp. 666-675).

Tulke, J. and Hanff, J. (2007). 4D Construction sequence planning: New process and data model. Available at: http://www.inpro-project.eu/media/4d_jantulke.pdf. (Mar., 2014).

Venugopal, M., Eastman, C. M., Sacks, R., and Teizer, J. (2012). Semantics of model views for information exchanges using the industry foundation class schema. Advanced Engineering Informatics, 26(2), 411-428.

Venugopal, M., Eastman, C. M., and Teizer, J. (2015). An ontology-based analysis of the industry foundation class schema for building information model exchanges. Advanced Engineering Informatics, 29(4), 940-957.

Vesse R., Zettlemoyer R.M., Ahmed K., Moore G., Pluskiewicz T. (2014). dotNetRDF, an open source .Net library for RDF. Available at: http://www.dotnetrdf.org/default.asp. (Sep. 11, 2015)

W3C Semantic Web (2015). RDF. Available at http://www.w3.org/RDF. (Dec. 12, 2015)

Wang W.C., Weng S.W., Wang S.H., and Chen C.Y. (2014). Integrating building information models with construction process simulations for project scheduling support. Automation in Construction, 37, 68-80.

Webster, D. (2014). What BIM can be: LOD 300: Optimizing your model for construction documentation. Available at http://www.mastergraphics.com/ wordpress/2013/what-bim-can-be-lod-300-optimizing-your-model-for-construction-documentation/. (Mar. 23, 2015)

Weygant, R. S. (2011). BIM content development: standards, strategies, and best practices. Wiley, Hoboken, NJ, USA.

Won, J., Cheng, J. C., and Lee, G. (2016). Quantification of construction waste prevented by BIM-based design validation: Case studies in South Korea. Waste Management.

Yang, Q. Z., and Zhang, Y. (2006). Semantic interoperability in building design: Methods and tools. Computer-Aided Design, 38(10), 1099-1112.

Yu, S. (2010). The prevention and recycling of clean drywall waste. Available at http://www.recyclingproductnews.com/article/14816/the-prevention-and-recycling-of-clean-drywall-waste. (Aug. 18, 2016)

Zhang, H., Li, X., Li, H., and Huang, F. (2006). Particle swarm optimization-based schemes for resource-constrained project scheduling. Automation in Construction, 14(3), 393-404.

Zhang, H., Tam, C. M., and Shi, J. J. (2002). Simulation-based methodology for project scheduling. Construction Management and Economics, 20, 667-678.

Zhang, L., and Issa, R. R. (2013). Ontology-based partial Building Information Model extraction. Journal of Computing in Civil Engineering, 27(6), 576-584.

Zhang, S., Teizer, J., Lee, J. K., Eastman, C. M., and Venugopal, M. (2013). Building information modeling (BIM) and safety: Automatic safety checking of construction models and schedules. Automation in Construction, 29, 183-195.

Zhao, H., Liu, H., and Al-Hussein, M. (2015). Automation of quantity takeoff for modular construction. Proceedings, 2015 Modular and Offsite Construction (MOC) Summit and 1st International Conference on the Industrialization of Construction (ICIC), Edmonton, AB, Canada, May 19-21, 458-465.

Zheng, C., and Lu, M. (2016). Optimized Reinforcement Detailing Design for Sustainable Construction: Slab Case Study. Procedia Engineering, 145, 1478-1485.

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

Zhong, B. T., Ding, L. Y., Love, P. E., and Luo, H. B. (2015). An ontological approach for technical plan definition and verification in construction. Automation in Construction, 55, 47-57.

APPENDIX A

Glossary

Boarding design: Boarding design refers to the layout design of sheathing and drywall sheets on walls and floors according to design principles and construction best practice.

Semantic QTO: Semantic QTO is an ontology-based semantic approach for construction-oriented quantity take-off (QTO) that allows users to semantically query the BIM model using domain vocabularies in order to retrieve the quantity take-off information for construction processes.

Panelized construction scheduling: Panelized construction scheduling is defined as detailed scheduling where activities should distinguish each individual building component, instead of distinguishing each construction zone, and request different resources in accordance with specific construction methods in order to build individual building components.

APPENDIX B

Excerpt from the C#.NET codes for the wall boarding layout design algorithm:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Serialization;
using Autodesk.Revit.DB;

namespace HexuLibrary.Revit
{
    public class WallLayer : Geometry
    {
        private const double LengthDrywallSheet;
        private const double HeightDrywallSheet;
        private const double StudThickness = 1.5 / 12;
        private const double TopGap = 0.5 / 12;
        private const double BottomGap = 5 / 8.0 / 12.0;

        private readonly Wall wall;
        private readonly Part currentPart;

        public WallLayer(Element partElement, Element wallElement, XYZ startP, XYZ
endP)
            : base(partElement, wallElement)
        {
            this.wall = this.HostElement as Wall;
            this.currentPart = partElement as Part;
            this.ExteriorFaceNormalVector = this.wall != null? this.wall.Orientation :
new XYZ(0,0,0);
            this.Direction = startP.GetVector(endP).Normalize();
            this.SetGeometryReference(startP, endP);
        }

        [XmlIgnore]
        public XYZ Direction { get; private set;}

        private Face InteriorFace { get; set; }

        private void GetInteriorFace()
        {
            try
            {
                var facesEnumerator = this.Faces.GetEnumerator();
                while (facesEnumerator.MoveNext())
                {
                    var face = facesEnumerator.Current as Face;
                    if (face == null) continue;
                    if (!face.IsVertical()) continue;
                    var interiorDirection = -this.ExteriorFaceNormalVector;
                    //(this.wall).GetExteriorFaceNormalVector();
```

```
                    if (face.ComputeNormal(new UV(0,
0)).IsAlmostEqualTo(interiorDirection))
                    {
                        this.InteriorFace = face;
                        break;
                    }
                }
            }
            catch (Exception ex)
            {
                throw new UofAException("Drywall.GetInteriorFace",
this.HostElement.Id.ToString(), ex);
            }
        }

        public void OffsetDrywall()
        {
            if (this.currentPart.CanOffsetFace(this.TopFace))
                this.currentPart.SetFaceOffset(this.TopFace, -0.5);
        }

        public IEnumerable<Drywall> DividePart(IList<Curve> curveArray)
        {
            try
            {
                if (curveArray.Any())
                {
                    this.GetInteriorFace();
                    using (SketchPlane sketchPlane = SketchPlane.Create(this.mDoc,
this.InteriorFace.Reference))
                    {
                        ICollection<ElementId> partsToBeCutted = new List<ElementId>
{ this.GeometryElement.Id };
                        if (PartUtils.ArePartsValidForDivide(this.mDoc,
partsToBeCutted))
                        {
                            PartMaker partMaker = PartUtils.DivideParts(this.mDoc,
partsToBeCutted, new List<ElementId>(), curveArray, sketchPlane.Id);
                            this.mDoc.Regenerate();
                            var division =
partMaker.get_Parameter(BuiltInParameter.PARTMAKER_PARAM_DIVISION_GAP);
                            division.Set(0 / 12.0);
                        }
                    }
                    this.mDoc.ActiveView.PartsVisibility =
PartsVisibility.ShowPartsOnly;
                    IEnumerable<Element> itself = new List<Element>();
                    return
this.GeometryElement.GetAssociatePartsforElement(true).Except(itself.Add(this.Geometr
yElement)).
                        Select(dry => new Drywall(this.HostElement, dry)).ToList();
                }
            }
            catch (Exception ex)
            {
```

```
                throw new UofAException("Drywall Error",
this.HostElement.Id.ToString(), ex);
            }
            return null;
        }

        public List<Curve> GetDesignLayout(Wall_Hexu hostedWall, double length,
double HeightDrywallSheet)//GetCuttingPattern
        {
            this.LengthDrywallSheet = length;
            this.HeightDrywallSheet = height;
            var curves = new List<Curve>();
            if (!hostedWall.SubComponents.Openings.Any())
            {
                curves.AddRange(GetCurveArrayforSingleArea(this.StartPoint,
this.EndPoint, this.Height, true, true));
            }
            else
            {
                var openings = hostedWall.SubComponents.Openings.ToList();
                openings.Sort();
                var firstOpening = openings.FirstOrDefault();
                var lastOpening = openings.LastOrDefault();
                if (firstOpening != null)
                {
                    curves.AddRange(GetCurveArrayforSingleArea(this.StartPoint,
firstOpening.StartPointXY.Add(XYZ.BasisZ.Multiply(this.StartPoint.Z)),//- diff
                        this.Height, true, false));


curves.AddRange(hostedWall.SubComponents.Openings.SelectMany(opening =>
opening.GetDrywallEdges()));

                    if (openings.Count > 1)
                    {
                        for (int i = 0; i < openings.Count - 1; i++)
                        {

curves.AddRange(GetCurveArrayforSingleArea(openings[i].EndPointXY.Add(XYZ.BasisZ.Mult
iply(this.StartPoint.Z)),
                                openings[i +
1].StartPointXY.Add(XYZ.BasisZ.Multiply(this.StartPoint.Z)),
                                this.Height, false, false));
                        }
                    }

                    if (lastOpening != null)

curves.AddRange(GetCurveArrayforSingleArea(lastOpening.EndPointXY.Add(XYZ.BasisZ.Mult
iply(this.StartPoint.Z)), this.EndPoint,
                                this.Height, false, true));
                }
            }
            return curves;
        }
```

```
        private static IEnumerable<Curve> GetCurveArrayforSingleArea(XYZ areaOrigin,
XYZ areaEnd,
            double areaHeight, bool isLeft, bool isRight)
        {
            List<Curve> curveArray = new List<Curve>();
            XYZ areaXAxis = areaEnd.Subtract(areaOrigin).Normalize();
            double areaLength = areaOrigin.DistanceTo(areaEnd);
            Transform honrizontalTran =
                Transform.CreateTranslation(areaXAxis.Multiply(LengthDrywall));
            Curve verticalCut = Line.CreateBound(areaOrigin.Add(XYZ.BasisZ.Multiply(-
1)),
                areaOrigin.Add(XYZ.BasisZ.Multiply(areaHeight + 1)));
            if (areaLength >= 6.0)
            {
                for (int i = 1; i < areaLength / LengthDrywall; ++i)
                {
                    verticalCut = verticalCut.CreateTransformed(honrizontalTran);
                    curveArray.Add(verticalCut);
                }
            }

            Transform verticalTran = Transform.CreateTranslation(
                    XYZ.BasisZ.Multiply(HeightDrywall));
            // areaLength < 6.0 && areaLength > 4.0 ? LengthDrywall : HeightDrywall
            var start = isLeft ? areaOrigin.Add(areaXAxis.Multiply(-5)) : areaOrigin;
            var end = isRight
                ? areaOrigin.Add(areaXAxis.Multiply(areaLength + 5))
                : areaOrigin.Add(areaXAxis.Multiply(areaLength));
            Curve honrizontalCut = Line.CreateBound(start, end);
            for (int i = 1; i < areaHeight / (areaLength < 6.0 && areaLength > 4.0 ?
LengthDrywall : HeightDrywall); ++i)
            {
                honrizontalCut = honrizontalCut.CreateTransformed(verticalTran);
                curveArray.Add(honrizontalCut);
            }
            return curveArray;
        }

        public List<Curve> GetStaggerCuttingPattern(StudFramedWall hostedWall)
        {
            this.CheckSharedParameter(hostedWall);
            List<Curve> curveArray = new List<Curve>();
            var horizontalCurves = this.GetHorizontalCuttingCurves().ToList();
            curveArray.AddRange(horizontalCurves);
            var verticalCurves = this.GetVerticalCuttingCurves(horizontalCurves,
hostedWall);
            curveArray.AddRange(verticalCurves);
            foreach (var opening in hostedWall.SubComponents.Openings)
            {
                curveArray.AddRange(opening.GetWindowRoughEdges(this.StartPoint,
this.EndPoint));
            }
            return curveArray;
        }
```

```
        private IEnumerable<Curve> GetVerticalCuttingCurves(IList<Curve>
horizontalCurves, StudFramedWall hostedWall)
        {
            List<Curve> curveArray = new List<Curve>();
            XYZ areaXAxis = EndPoint.Subtract(StartPoint).Normalize();
            Dictionary<int, List<XYZ>> edges = new Dictionary<int, List<XYZ>>();
            for (int rowindex = 1; rowindex <= horizontalCurves.Count; rowindex++)
            {
                XYZ startPoint_Up, startPoint_Bot, endPoint_Up, endPoint_Bot;
                this.CalculateEndPoints(horizontalCurves, rowindex, out startPoint_Up,
                    out startPoint_Bot, out endPoint_Up, out endPoint_Bot);

                Curve verticalCut = !rowindex.IsOdd() ?
Line.CreateBound(startPoint_Up, startPoint_Bot) :
                    Line.CreateBound(endPoint_Up, endPoint_Bot);
                var edgesPerRow = new List<XYZ>();
                do
                {
                    var startPoint = verticalCut.GetEndPoint(0);
                    var endPoint =
startPoint.Add(areaXAxis.Multiply(!rowindex.IsOdd() ? 1 : -
1).Normalize().Multiply(LengthDrywall));
                    var farthestStudLocation =
hostedWall.GetCoveredFarthestStud(startPoint, endPoint);

                    farthestStudLocation =
this.StaggerToNearestStudLocation(startPoint, farthestStudLocation, hostedWall,
rowindex, edges);
                    farthestStudLocation =
MoveVerticalCutAwayfromOpeningEdges(startPoint, farthestStudLocation, hostedWall);

                    verticalCut = verticalCut.CreateTransformed(
                            Transform.CreateTranslation(
                            areaXAxis.Multiply((!rowindex.IsOdd() ? 1 : -1) *
verticalCut.GetEndPoint(0).DistanceTo(farthestStudLocation))));
                    //verticalCut = this.MoveVerticalCutAwayfromOpening(verticalCut,
rowindex, areaXAxis, hostedWall);
                    if (!IsNearWallEdge(verticalCut.GetEndPoint(0)))
                    {
                        if ((rowindex != 2 || rowindex == 2
&& !IsBetweenDoors(verticalCut.GetEndPoint(0), hostedWall)))
                        {

edgesPerRow.Add(verticalCut.GetEndPoint(0).SetZValue(this.StartPoint.Z));
                            curveArray.Add(verticalCut);
                        }
                    }
                }
                while
(verticalCut.GetEndPoint(0).DistanceTo(!rowindex.IsOdd() ?endPoint_Up:startPoint_Up) >
LengthDrywall);
                edges.Add(rowindex, edgesPerRow);
            }
            return curveArray;
```

```
        }

        private bool IsNearWallEdge(XYZ cuttingPoint)
        {
            return
(cuttingPoint.SetZValue(this.StartPoint.Z).DistanceTo(this.StartPoint) < 1 / 12.0) ||
                   (cuttingPoint.SetZValue(this.EndPoint.Z).DistanceTo(this.EndPoint)
< 1 / 12.0)|| this.Length < 8;
        }

        private bool IsBetweenDoors(XYZ cuttingPoint, StudFramedWall hostedWall)
        {
            var doors = hostedWall.SubComponents.Doors.ToList();
            var leftDoors =
               doors.Where(
                   door =>

cuttingPoint.Subtract(door.LocationXYZ.SetZValue(cuttingPoint.Z))
                            .Normalize()
                            .DotProduct(hostedWall.Core.Direction) > 0).ToList();

            var rightDoors =
              doors.Where(
                  door =>

cuttingPoint.Subtract(door.LocationXYZ.SetZValue(cuttingPoint.Z))
                           .Normalize()
                           .DotProduct(hostedWall.Core.Direction) < 0).ToList();

            leftDoors.Sort((door1, door2) =>
            {
                if (null == door1 || null == door2)
                {
                    return -1;
                }
                return
                    door1.LocationXYZ.DistanceTo(cuttingPoint)
                        .CompareTo(door2.LocationXYZ.DistanceTo(cuttingPoint));
            });

            rightDoors.Sort((door1, door2) =>
            {
                if (null == door1 || null == door2)
                {
                    return -1;
                }
                return
                    door1.LocationXYZ.DistanceTo(cuttingPoint)
                        .CompareTo(door2.LocationXYZ.DistanceTo(cuttingPoint));
            });
            var leftDoor = leftDoors.FirstOrDefault();
            var rightDoor = rightDoors.FirstOrDefault();

            if (leftDoor != null && rightDoor != null)
            {
```

```
                if
(cuttingPoint.DistanceTo(leftDoor.EndPoint.SetZValue(cuttingPoint.Z)) +

cuttingPoint.DistanceTo(rightDoor.StartPoint.SetZValue(cuttingPoint.Z)) <
LengthDrywall)
                {
                    return true;
                }
            }

            if (leftDoor != null)
            {
                if
(cuttingPoint.DistanceTo(leftDoor.EndPoint.SetZValue(cuttingPoint.Z)) +
                    cuttingPoint.DistanceTo(this.EndPoint.SetZValue(cuttingPoint.Z))
< LengthDrywall)
                {
                    return true;
                }
            }

            if (rightDoor != null)
            {
                if (cuttingPoint.DistanceTo(this.StartPoint.SetZValue(cuttingPoint.Z))
+

cuttingPoint.DistanceTo(rightDoor.StartPoint.SetZValue(cuttingPoint.Z)) <
LengthDrywall)
                {
                    return true;
                }
            }
            return false;
        }

        private XYZ StaggerToNearestStudLocation(XYZ startPoint, XYZ
farthestStudLocation, StudFramedWall hostedWall,
            int rowindex, IDictionary<int, List<XYZ>> takenEdges)
        {
            if (rowindex == 1) return farthestStudLocation;
            XYZ location = farthestStudLocation;
            if (takenEdges[rowindex - 1].Select(edge =>
edge.DistanceTo(location.SetZValue(this.StartPoint.Z))).
                Any(distance => Math.Round(distance, 2) <= 2 * StudThickness))
            {
                farthestStudLocation = hostedWall.GetCoveredFarthestStud(startPoint,
farthestStudLocation);
            }
            return farthestStudLocation;
        }

        private static XYZ MoveVerticalCutAwayfromOpeningEdges(XYZ startPoint, XYZ
farthestStudLocation, StudFramedWall hostedWall)
        {
            XYZ previousNearest;
```

```
            do
            {
                previousNearest = new XYZ(farthestStudLocation.X,
farthestStudLocation.Y, farthestStudLocation.Z);
                if (
                    hostedWall.SubComponents.Openings.Any(
                        openning =>

openning.StartPoint.SetZValue(farthestStudLocation.Z).DistanceTo(farthestStudLocation)
<
                            2 * StudThickness ||

openning.EndPoint.SetZValue(farthestStudLocation.Z).DistanceTo(farthestStudLocation)
<
                            2 * StudThickness))
                {
                    farthestStudLocation =
hostedWall.GetCoveredFarthestStud(startPoint, farthestStudLocation);
                }
            } while (!farthestStudLocation.IsEqual(previousNearest));


            return farthestStudLocation;
        }

        private Curve MoveVerticalCutAwayfromOpening(Curve verticalCut, int i, XYZ
areaXAxis, StudFramedWall hostedWall)
        {
            foreach (var openning in hostedWall.SubComponents.Openings)
            {
                if (
                    openning.GetWindowEdges(this.StartPoint, this.EndPoint)
                        .Any(windowEdge => { return verticalCut.Intersect(windowEdge)
== SetComparisonResult.Overlap; }))
                {
                    verticalCut = verticalCut.CreateTransformed(!i.IsOdd()
                        ? Transform.CreateTranslation(areaXAxis.Multiply(-
LengthDrywall / 3.0))
                        :
Transform.CreateTranslation(areaXAxis.Multiply(LengthDrywall / 3.0)));
                }
            }
            return verticalCut;
        }

        private void CalculateEndPoints(IList<Curve> horizontalCurves, int i, out XYZ
startPoint_Up, out XYZ startPoint_Bot,
            out XYZ endPoint_Up, out XYZ endPoint_Bot)
        {
            if (i == 1)
            {
                startPoint_Up = StartPoint.Add(XYZ.BasisZ.Multiply(BottomGap));
                startPoint_Bot = StartPoint.Add(XYZ.BasisZ.Multiply(-1));
                endPoint_Up = EndPoint.Add(XYZ.BasisZ.Multiply(BottomGap));
                endPoint_Bot = EndPoint.Add(XYZ.BasisZ.Multiply(-1));
```

```
            }
            else if (i == horizontalCurves.Count + 1)
            {
                startPoint_Up = StartPoint.Add(XYZ.BasisZ.Multiply(Height + 1));
                startPoint_Bot = StartPoint.SetZValue(horizontalCurves[i -
2].GetEndPoint(0).Z);
                endPoint_Up = EndPoint.Add(XYZ.BasisZ.Multiply(Height + 1));
                endPoint_Bot = EndPoint.SetZValue(horizontalCurves[i -
2].GetEndPoint(0).Z);
            }
            else
            {
                startPoint_Up = StartPoint.SetZValue(horizontalCurves[i -
1].GetEndPoint(0).Z);
                startPoint_Bot = StartPoint.SetZValue(horizontalCurves[i -
2].GetEndPoint(0).Z);
                endPoint_Up = EndPoint.SetZValue(horizontalCurves[i -
1].GetEndPoint(0).Z);
                endPoint_Bot = EndPoint.SetZValue(horizontalCurves[i -
2].GetEndPoint(0).Z);
            }
        }

        private IEnumerable<Curve> GetHorizontalCuttingCurves(bool fromBot, bool
isLeft = true, bool isRight = true)
        {
            try
            {
                List<Curve> curveArray = new List<Curve>();
                XYZ areaXAxis = EndPoint.Subtract(StartPoint).Normalize();
                double areaLength = StartPoint.DistanceTo(EndPoint);

                var start = isLeft ? StartPoint.Add(areaXAxis.Multiply(-5)) :
StartPoint;
                var end = isRight
                    ? StartPoint.Add(areaXAxis.Multiply(areaLength + 5))
                    : StartPoint.Add(areaXAxis.Multiply(areaLength));

                Curve honrizontalCut = Line.CreateBound(start, end);

                Transform verticalTran =
Transform.CreateTranslation(XYZ.BasisZ.Multiply(HeightDrywall));
                double remaining = Height;
                for (int i = 1; i < Height / HeightDrywall; ++i)
                {
                    if (remaining > 1.33 * HeightDrywall)
                    {
                        honrizontalCut =
honrizontalCut.CreateTransformed(verticalTran);
                        curveArray.Add(honrizontalCut);
                        remaining = remaining - HeightDrywall;
                    }
                }
                double lastTansform =
```

```
honrizontalCut.GetEndPoint(0).DistanceTo(start.Add(XYZ.BasisZ.Multiply(this.Height -
TopGap)));
                honrizontalCut =
honrizontalCut.CreateTransformed(Transform.CreateTranslation(XYZ.BasisZ.Multiply(last
Tansform)));
                curveArray.Add(honrizontalCut);
                return curveArray;
            }
            catch (Exception ex)
            {
                throw new UofAException("Cannot get horizontal cutting curves",
this.HostElement.Id.IntegerValue.ToString(), ex);
            }
        }

        private IEnumerable<Curve> GetHorizontalCuttingCurves(bool isLeft = true,
bool isRight = true)
        {
            try
            {
                List<Curve> curveArray = new List<Curve>();
                XYZ areaXAxis = EndPoint.Subtract(StartPoint).Normalize();
                double areaLength = StartPoint.DistanceTo(EndPoint);

                var start = isLeft ? StartPoint.Add(areaXAxis.Multiply(-5)) :
StartPoint;
                var end = isRight
                    ? StartPoint.Add(areaXAxis.Multiply(areaLength + 5))
                    : StartPoint.Add(areaXAxis.Multiply(areaLength));

                Curve honrizontalCut_BGap =
Line.CreateBound(start.Add(XYZ.BasisZ.Multiply(BottomGap)),
                     end.Add(XYZ.BasisZ.Multiply(BottomGap)));
                curveArray.Add(honrizontalCut_BGap);

                if (this.Length > 4.5)
                {
                    Curve honrizontalCut_B =
Line.CreateBound(start.Add(XYZ.BasisZ.Multiply(BottomGap + 4.5)),
                        end.Add(XYZ.BasisZ.Multiply(BottomGap + 4.5)));
                    curveArray.Add(honrizontalCut_B);

                }

                Curve honrizontalCut =
Line.CreateBound(start.Add(XYZ.BasisZ.Multiply(this.Height - TopGap)),
                    end.Add(XYZ.BasisZ.Multiply(this.Height - TopGap)));
                curveArray.Add(honrizontalCut);
                return curveArray;
            }
            catch (Exception ex)
            {
                throw new UofAException("Cannot get horizontal cutting curves",
this.HostElement.Id.IntegerValue.ToString(), ex);
```

```
            }

        }

        private void CheckSharedParameter(StudFramedWall hostedWall)
        {
            Parameter hostParameter =
this.GeometryElement.FindParameterByName("PartHost");
            Parameter isExteriorParameter =
this.GeometryElement.FindParameterByName("IsExterior");
            Parameter isNotWaste =
this.GeometryElement.FindParameterByName("IsNotWaste");
            Parameter nameParameter =
this.GeometryElement.FindParameterByName("Name");

            if (hostParameter != null && isExteriorParameter != null &&
nameParameter != null && isNotWaste != null)
            {
                hostParameter.Set(hostedWall.NameofWallPanel);

isExteriorParameter.Set(this.StartPoint.SetZValue(hostedWall.Core.StartPoint.Z)
                    .IsOnBottomSide(hostedWall.Core.StartPoint,
                    hostedWall.Core.EndPoint) ? 0 : 1);
            }
            else
            {
                this.CreateSharedParameterforPart();
                Parameter newHostParameter =
this.GeometryElement.FindParameterByName("PartHost");
                Parameter newIsExteriorParameter =
this.GeometryElement.FindParameterByName("IsExterior");
                newHostParameter.Set(hostedWall.NameofWallPanel);

newIsExteriorParameter.Set(this.StartPoint.SetZValue(hostedWall.Core.StartPoint.Z)
                    .IsOnBottomSide(hostedWall.Core.StartPoint,
                    hostedWall.Core.EndPoint) ? 0 : 1);
            }
        }

        private void CreateSharedParameterforPart()
        {
            var partHost =
ParameterUtil.GetOrCreateSharedParameter(this.mDoc.Application, "PartHost", "Star",
ParameterType.Text,
                true);
            var isExterior =
ParameterUtil.GetOrCreateSharedParameter(this.mDoc.Application, "IsExterior", "Star",
ParameterType.YesNo,
                true);
            var isNotWaste =
ParameterUtil.GetOrCreateSharedParameter(this.mDoc.Application, "IsNotWaste", "Star",
ParameterType.YesNo,
                true);
            var name = ParameterUtil.GetOrCreateSharedParameter(this.mDoc.Application,
"Name", "Star", ParameterType.Text,
```

```
                true);

            CategorySet categorySetPart =
this.mDoc.Application.Create.NewCategorySet();

categorySetPart.Insert(this.mDoc.Settings.Categories.get_Item(BuiltInCategory.OST_Par
ts));

            ParameterUtil.AttachSharedParameter(this.mDoc.Application, partHost,
categorySetPart,
                BuiltInParameterGroup.PG_IDENTITY_DATA, true);
            ParameterUtil.AttachSharedParameter(this.mDoc.Application, isExterior,
categorySetPart,
                BuiltInParameterGroup.PG_IDENTITY_DATA, true);
            ParameterUtil.AttachSharedParameter(this.mDoc.Application, isNotWaste,
categorySetPart,
                BuiltInParameterGroup.PG_IDENTITY_DATA, true);
            ParameterUtil.AttachSharedParameter(this.mDoc.Application, name,
categorySetPart,
                BuiltInParameterGroup.PG_IDENTITY_DATA, true);
        }
    }
}



using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Serialization;
using Autodesk.Revit.DB;
using HexuLibrary.Revit.Analyzor;

namespace HexuLibrary.Revit
{
    [Serializable]
    public class Geometry
    {
        protected Document mDoc { get; private set; }
        private XYZ refStart;
        private XYZ refEnd;

        private List<XYZ> fourVerticesBottom;

        internal Geometry()
        {
            this.ExteriorFaceNormalVector = new XYZ();
        }

        public Geometry(Element partGeoElement, Element hostElement)
        {
            this.GeometryElement = partGeoElement;
            this.mDoc = partGeoElement.Document;
            this.HostElement = hostElement;
            this.ExteriorFaceNormalVector = new XYZ();
```

```
        }

        public readonly Element GeometryElement;
        protected Element HostElement { get; private set; }

        public virtual double Length
        {
            get { return this.StartPoint.DistanceTo(this.EndPoint); }
        }

        private double width;
        public double Width
        {
            get
            {
                if (Math.Abs(this.width) < Utility.Tolerance)
                {
                    if (this.GeometryElement is Wall)
                    {
                        var wall = this.GeometryElement as Wall;
                        this.width = wall.Width;
                    }
                    else
                    {
                        this.width =
this.GeometryElement.FindParameterByName("Thickness").AsDouble();
                    }
                }
                return this.width;
            }
        }

        private double height;
        public double Height
        {
            get
            {
                if (Math.Abs(this.height) < Utility.Tolerance)
                {
                    if (this.GeometryElement is Wall)
                    {
                        var wall = this.GeometryElement as Wall;
                        this.height =
wall.get_Parameter(BuiltInParameter.WALL_USER_HEIGHT_PARAM).AsDouble();
                    }
                    else
                    {
                        this.height =
this.GeometryElement.FindParameterByName("Height").AsDouble();
                    }
                }
                return this.height;
            }
        }
```

```
        private double area;
        public double Area
        {
            get
            {
                if (Math.Abs(this.area) < Utility.Tolerance)
                {
                    this.area =
this.GeometryElement.FindParameterByName("Area").AsDouble();
                }
                return this.area;
            }
        }

        [XmlIgnore]
        public XYZ ExteriorFaceNormalVector { get; protected set; }
        //this.exteriorFaceNormalVector = this.exteriorFaceNormalVector ??
this.GetExteriorFaceNormalVector();

        private FaceArray faces;
        [XmlIgnore]
        public FaceArray Faces
        {
            get
            {
                this.faces = this.faces ?? this.GeometryElement.GetFacesforElement();
                return this.faces;
            }
        }

        private List<Face> verticalFaces;
        public IEnumerable<Face> VerticalFaces
        {
            get
            {
                this.verticalFaces = this.verticalFaces ??
this.Faces.Cast<Face>().Where(e => e.IsVertical()).ToList();
                return this.verticalFaces;
            }
        }

        private List<Face> horizontalFaces;
        public IEnumerable<Face> HorizontalFaces
        {
            get
            {
                this.horizontalFaces = this.horizontalFaces ??
this.Faces.Cast<Face>().Where(e => e.IsHorizontal()).ToList();
                return this.horizontalFaces;
            }
        }

        private Face topFace;
        [XmlIgnore]
        public Face TopFace
```

```
        {
            get
            {
                this.topFace = this.topFace ??
this.Faces.GetFaceByComparing_ZElevation((ideal, each) => ideal < each); //Bottom:
(ideal, each) => ideal > each
                return this.topFace;
            }
        }

        private IEnumerable<Face> bottomFaces;
        public IEnumerable<Face> BottomFaces
        {
            get
            {
                if (this.bottomFaces != null) return this.bottomFaces;
                IGrouping<double, Face> bottomGroup = this.HorizontalFaces
                    .GroupBy(face =>
face.GetElevationOfFaceinAxisDirection(XYZ.BasisZ))
                    .OrderBy(g => g.Key).FirstOrDefault();
                if (bottomGroup != null)
                    this.bottomFaces = this.bottomFaces ?? bottomGroup.ToList();
                return this.bottomFaces;
            }
        }

        private IEnumerable<Face> topFaces;
        public IEnumerable<Face> TopFaces
        {
            get
            {
                if (this.topFaces != null) return this.topFaces;
                if (this.HorizontalFaces != null)
                {
                    this.topFaces = this.HorizontalFaces
                                        .GroupBy(face =>
face.GetElevationOfFaceinAxisDirection(XYZ.BasisZ))
                                        .OrderByDescending(g => g.Key)
                                        .FirstOrDefault();
                }
                return this.topFaces;
            }
        }

        /// <summary>
        /// Vertical face of at the right of the core layer for the wall
        /// </summary>
        [XmlIgnore]
        public Face RightFace
        {
            get
            {
                return
                    this.VerticalFaces
```

```
                        .GetFacesBySpecifiedNormalVector(this.ExteriorFaceNormalVecto
r.Multiply(-1)).FirstOrDefault();
            }
        }

        /// <summary>
        ///     Vertical face of at the left of the core layer for the wall
        /// </summary>
        [XmlIgnore]
        public Face LeftFace
        {
            get
            {
                return
                    this.VerticalFaces
                        .GetFacesBySpecifiedNormalVector(this.ExteriorFaceNormalVecto
r).FirstOrDefault();
            }
        }

        private Face startFace;
        /// <summary>
        ///     Vertical face of at the start point of the core layer for the wall
        /// </summary>
        [XmlIgnore]
        public Face StartFace
        {
            get
            {
                try
                {
                    this.startFace = this.startFace ??
this.Faces.GetVerticalFace_HostingPoint(StartPoint);
                }
                catch (Exception exception)
                {
                    throw new UofAException(exception.Message,
this.GeometryElement.Id.IntegerValue.ToString(), exception);
                }
                return this.startFace;
            }
        }

        private Face endFace;
        /// <summary>
        ///     Vertical face of at the end point of the core layer for the wall
        /// </summary>
        [XmlIgnore]
        public Face EndFace
        {
            get
            {
                try
                {
```

```
                    this.endFace = this.endFace ??
this.Faces.GetVerticalFace_HostingPoint(EndPoint);
                }
                catch (Exception exception)
                {
                    throw new UofAException(exception.Message,
this.GeometryElement.Id.IntegerValue.ToString(), exception);
                }
                return this.endFace;
            }
        }

        public XYZ StartPoint_Top
        {
            get
            {
                this.coreGeometyStartEndPoints = this.coreGeometyStartEndPoints ??
                                                 this.GetPartGeometyEndPoints(new
List<Face> {this.TopFace}); //this.GetPartGeometyEndPoints_Top();
                return this.coreGeometyStartEndPoints.Any() ?
this.coreGeometyStartEndPoints[0]
                    : new XYZ(0, 0, 0);
            }
        }

        public XYZ EndPoint_Top
        {
            get
            {
                this.coreGeometyStartEndPoints = this.coreGeometyStartEndPoints ??
                                                 this.GetPartGeometyEndPoints(new
List<Face>{this.TopFace}); //this.GetPartGeometyEndPoints_Top();
                return this.coreGeometyStartEndPoints.Any() ?
this.coreGeometyStartEndPoints[1]
                    : new XYZ(0, 0, 0);
            }
        }

        /// <summary>
        /// Start point of the core geometry of the wall, no matter that the wall is
flipped or not
        /// Note: it is the middle point of core layer at wall elevation
        /// </summary>
        private List<XYZ> coreGeometyStartEndPoints;
        [XmlIgnore]
        public XYZ StartPoint
        {
            get
            {
                this.coreGeometyStartEndPoints = this.coreGeometyStartEndPoints ??
this.GetPartGeometyEndPoints(this.BottomFaces);//GetPartGeometyEndPoints_Top();
                return this.coreGeometyStartEndPoints.Any() ?
this.coreGeometyStartEndPoints[0]
                    : new XYZ(0, 0, 0);
            }
```

```
        }

        /// <summary>
        /// End point of the core geometry of this wall, no matter that the wall is
flipped or not
        /// Note: it is the middle point of core layer at wall elevation
        /// </summary>
        [XmlIgnore]
        public XYZ EndPoint
        {
            get
            {
                this.coreGeometyStartEndPoints = this.coreGeometyStartEndPoints ??
this.GetPartGeometyEndPoints(this.BottomFaces);//.GetPartGeometyEndPoints_Top();
                return this.coreGeometyStartEndPoints.Any() ?
this.coreGeometyStartEndPoints[1]//.Add(XYZ.BasisZ.Multiply(-this.Height))
                    : new XYZ(0, 0, 0);
            }
        }

        [XmlIgnore]
        public XYZ StartPointExterior
        {
            get
            {
                this.fourVerticesBottom = this.fourVerticesBottom ??
                                          this.GetFourVerticesforBottomFaces();
                return
                    this.fourVerticesBottom.GetNearByPoints(this.StartPoint, 2)
                        .FirstOrDefault(e => !e.IsOnBottomSide(this.StartPoint,
this.EndPoint));
            }
        }

        [XmlIgnore]
        public XYZ EndPointExterior
        {
            get
            {
                this.fourVerticesBottom = this.fourVerticesBottom ??
                                          this.GetFourVerticesforBottomFaces();
                return
                    this.fourVerticesBottom.GetNearByPoints(this.EndPoint, 2)
                        .FirstOrDefault(e => !e.IsOnBottomSide(this.StartPoint,
this.EndPoint));
            }
        }

        [XmlIgnore]
        public XYZ StartPointInterior
        {
            get
            {
                this.fourVerticesBottom = this.fourVerticesBottom ??
```

```
                                        this.GetFourVerticesforBottomFaces();
//Utility.GetFourVerticesforFace(this.BottomFace, this.startRef, this.endRef);
                return
                    this.fourVerticesBottom.GetNearByPoints(this.StartPoint, 2)
                        .FirstOrDefault(e => e.IsOnBottomSide(this.StartPoint,
this.EndPoint));
            }
        }

        [XmlIgnore]
        public XYZ EndPointInterior
        {
            get
            {
                this.fourVerticesBottom = this.fourVerticesBottom ??
                                          this.GetFourVerticesforBottomFaces();
//Utility.GetFourVerticesforFace(this.BottomFace, this.startRef, this.endRef);
                return
                    this.fourVerticesBottom.GetNearByPoints(this.EndPoint, 2)
                        .FirstOrDefault(e => e.IsOnBottomSide(this.StartPoint,
this.EndPoint));
            }
        }

        protected void SetGeometryReference(XYZ start, XYZ end)
        {
            this.refStart = start;
            this.refEnd = end;
        }

        private XYZ GetExteriorFaceNormalVector()
        {
            Transform rotate90 = Transform.CreateRotationAtPoint(new XYZ(0, 0, 1),
Math.PI / 2, refStart);
            Line directionLine = Line.CreateBound(this.refStart, this.refEnd);
            var rotatedCurve = directionLine.CreateTransformed(rotate90);
            return
rotatedCurve.GetEndPoint(1).Subtract(rotatedCurve.GetEndPoint(0)).Normalize();
        }

        private List<XYZ> GetFourVerticesforBottomFaces()
        {
            return this.BottomFaces.SelectMany(face =>
face.GetFourVerticesforFace(this.refStart, this.refEnd)).ToList();
        }

        private List<XYZ> GetPartGeometyEndPoints(IEnumerable<Face>
facesToGetGeometyEndPoints)
        {
            try
            {
                var interactionLine =
this.GetPartCenterLine(facesToGetGeometyEndPoints);

                List<Curve> bottomEdges = new List<Curve>();
```

```
                foreach (var bottomFace in facesToGetGeometyEndPoints)
                {
                    bottomEdges.AddRange(bottomFace.GetEncloseEdges());
                }
                //this.currentElement.GetEncloseEdgesforFaceWithNormalVector(-
XYZ.BasisZ);
                List<XYZ> twoInteractPoints = (from bottomEdge in bottomEdges
                                               let edgeLine = bottomEdge as Line
                                               let result =
interactionLine.Intersect(bottomEdge)
                                               where result ==
SetComparisonResult.Overlap
                                               select
interactionLine.GetIntersectionPointBetweenTwoCurves(edgeLine)).ToList();
                return twoInteractPoints.Distinct(new DistinctXYZComparer()).ToList()
                        .GetTwoPointsNearSpecifiedPoints(interactionLine.GetEndPoint(
0), interactionLine.GetEndPoint(1)); //this.refStart, this.refEnd;
            }
            catch (Exception ex)
            {
                throw new Exception("Cannot get geometrical ends for the wall core
part ("+ this.GeometryElement.Id.IntegerValue+") due to "+ ex.Message, ex);
            }
        }

        private Line GetPartCenterLine(IEnumerable<Face> facesToGetCenterLine)
        {
            try
            {
                List<XYZ> faceBoundingPoints = new List<XYZ>();
                foreach (var bottomFace in facesToGetCenterLine)
                {

faceBoundingPoints.AddRange(bottomFace.GetEdgeMiddlePointsforFaceOnBoundingBox());
                }

                // when walls have door, the wall will have two parts.
                List<XYZ> twoBottomEndPoints =
faceBoundingPoints.GetPointsParallelToDirection(this.refStart, this.refEnd);
                var pointElevation = twoBottomEndPoints.FirstOrDefault();
                if (pointElevation == null) return null;
                if (twoBottomEndPoints.Count() == 2)
                {
                    var dir =
twoBottomEndPoints.FirstOrDefault().GetVector(twoBottomEndPoints.LastOrDefault());
                    if (dir.IsSameDirection(this.refStart.GetVector(this.refEnd)))
                    {
                        return Line.CreateBound(twoBottomEndPoints.FirstOrDefault(),
twoBottomEndPoints.LastOrDefault());
                    }
                    return Line.CreateBound(twoBottomEndPoints.LastOrDefault(),
twoBottomEndPoints.FirstOrDefault());
                }
                var elevation = pointElevation.Z;
```

```
                var leftPoints =
                    twoBottomEndPoints.Where(
                        point =>
                            point.IsOnLeftSide(this.refStart.SetZValue(elevation),
this.refEnd.SetZValue(elevation))).ToList();
                var rightPoints =
                   twoBottomEndPoints.Where(
                       point =>
                           point.IsOnRightSide(this.refStart.SetZValue(elevation),
this.refEnd.SetZValue(elevation))).ToList();

                var startP =
leftPoints.Any()?leftPoints.GetFarthestByPoints(this.refStart.SetZValue(elevation),
1).FirstOrDefault():

twoBottomEndPoints.GetNearByPoints(this.refStart.SetZValue(elevation),
1).FirstOrDefault();
                var endP = rightPoints.Any() ?
rightPoints.GetFarthestByPoints(this.refEnd.SetZValue(elevation),
1).FirstOrDefault() :

twoBottomEndPoints.GetNearByPoints(this.refEnd.SetZValue(elevation),
1).FirstOrDefault();

                if (startP.IsEqual(endP)) throw new ArgumentException("Cannot get two
distinct end points for geometry");
                Line interactionLine = Line.CreateBound(startP, endP);
                return interactionLine; // the line going through all the wall
            }
            catch (Exception ex)
            {
                throw new Exception("Cannot get bottom center line for Part", ex);
            }
        }
    }
}
```

APPENDIX C

Excerpt from the C#.NET codes for generating ontology-enhanced BIM models:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using HexuLibrary;
using HexuLibrary.Revit;
using VDS.RDF;
using VDS.RDF.Nodes;
using VDS.RDF.Parsing;
using VDS.RDF.Writing;
using BuildingComponent = HexuLibrary.Revit.BuildingComponent;
using Stud = GeneralTemplateExtension.Stud;

namespace Addin_Fortis.RDF
{
    internal class KnowledgeBaseGenerator
    {
        private IUriNode primary;
        private IUriNode second;
        private IUriNode third;
        private IUriNode height;
        private IUriNode length;
        private IUriNode thickness;
        private IUriNode volume;
        private IUriNode hostID;
        private IUriNode iD;
        private IUriNode isAcoustic;
        private IUriNode isBalloonSystem;
        private IUriNode isExterior;
        private IUriNode isFireRated;
        private IUriNode isHonrizontal;
        private IUriNode isMechanical;
        private IUriNode isPartition;
        private IUriNode isStraight;
        private IUriNode isStructural;
        private IUriNode isWashroomPanel;
        private IUriNode isSubIntersection;
        private IUriNode hasDoubleTopPlate;
        private IUriNode level;
        private IUriNode material;
        private IUriNode name;
        private IUriNode stud_Spacing;
        private IUriNode stud_Size;
        private IUriNode size;
        private IUriNode hostWall;
        private IUriNode weight;
        private IUriNode workZone;

        private IUriNode typeOf;
        private IUriNode studFramedWall;
```

```
        private IUriNode concreteWall;
        private IUriNode window;
        private IUriNode door;
        private IUriNode cripple;
        private IUriNode jack;
        private IUriNode joint;
        private IUriNode king;
        private IUriNode normal;
        private IUriNode blocking;
        private IUriNode bottomPlate;
        private IUriNode header;
        private IUriNode sill;
        private IUriNode topPlate;
        private IUriNode lConnection;
        private IUriNode doubleTConnection;
        private IUriNode tConnection;

        private IUriNode connectedWith;
        private IUriNode hasIntersection;
        private IUriNode hasSubIntersection;
        private IUriNode hasDoor;
        private IUriNode hasWindow;
        private IUriNode hasSubComponent;
        private IUriNode hostedBy;
        private IUriNode isInstanceOf;
        private IUriNode isMadeOf;
        private IUriNode isPartOf;
        private IUriNode hasPattern;
        private IUriNode supportedBy;

        private readonly ILiteralNode booleanFalse;
        private readonly ILiteralNode booleanTrue;

        public readonly Graph Graph = new Graph();

        public KnowledgeBaseGenerator(string nameOfKnowledgeBase)
        {
            this.NameOfKnowledgeBase = nameOfKnowledgeBase;
            this.Graph.BaseUri = new
Uri("http://www.semanticweb.org/hexu/ontologies/2014/6/BuildingProcessOntology");
            this.Graph.LoadFromFile(nameOfKnowledgeBase);
            this.AddNamespace();
            this.SetUriNodesforTerm();
            this.SetUriNodesforProperty();
            this.SetUriNodesforInterrelationship();
            this.booleanTrue = Graph.CreateLiteralNode("true",
                UriFactory.Create(XmlSpecsHelper.XmlSchemaDataTypeBoolean));
            this.booleanFalse = Graph.CreateLiteralNode("false",
                UriFactory.Create(XmlSpecsHelper.XmlSchemaDataTypeBoolean));
        }

        private void SetUriNodesforTerm()
        {
            this.typeOf = Graph.CreateUriNode("rdf:type");
            this.studFramedWall = Graph.CreateUriNode("proOnto:StudFramedWall");
```

```
        this.concreteWall = Graph.CreateUriNode("proOnto:Wall");
        this.window = Graph.CreateUriNode("proOnto:Window");
        this.door = Graph.CreateUriNode("proOnto:Door");
        this.cripple = Graph.CreateUriNode("proOnto:Cripple");
        this.jack = Graph.CreateUriNode("proOnto:Jack");
        this.joint = Graph.CreateUriNode("proOnto:Joint");
        this.king = Graph.CreateUriNode("proOnto:King");
        this.normal = Graph.CreateUriNode("proOnto:Normal");
        this.blocking = Graph.CreateUriNode("proOnto:Blocking");
        this.bottomPlate = Graph.CreateUriNode("proOnto:BottomPlate");
        this.header = Graph.CreateUriNode("proOnto:Header");
        this.sill = Graph.CreateUriNode("proOnto:Sill");
        this.topPlate = Graph.CreateUriNode("proOnto:TopPlate");
        Graph.CreateUriNode("proOnto:IConnection");
        this.lConnection = Graph.CreateUriNode("proOnto:LConnection");
        this.doubleTConnection = Graph.CreateUriNode("proOnto:DoubleTConnection");
        this.tConnection = Graph.CreateUriNode("proOnto:TConnection");
    }

    private void SetUriNodesforProperty()
    {
        this.primary = Graph.CreateUriNode("proOnto:Primary");
        this.second = Graph.CreateUriNode("proOnto:Second");
        this.third = Graph.CreateUriNode("proOnto:Third");
        this.height = Graph.CreateUriNode("proOnto:Height");
        this.length = Graph.CreateUriNode("proOnto:Length");
        this.thickness = Graph.CreateUriNode("proOnto:Thickness");
        this.volume = Graph.CreateUriNode("proOnto:Volume");
        this.hostID = Graph.CreateUriNode("proOnto:HostID");
        this.iD = Graph.CreateUriNode("proOnto:ID");
        this.isAcoustic = Graph.CreateUriNode("proOnto:IsAcoustic");
        this.isBalloonSystem = Graph.CreateUriNode("proOnto:IsBalloonSystem");
        this.isExterior = Graph.CreateUriNode("proOnto:IsExterior");
        this.isFireRated = Graph.CreateUriNode("proOnto:IsFireRated");
        this.isHonrizontal = Graph.CreateUriNode("proOnto:IsHonrizontal");
        this.isMechanical = Graph.CreateUriNode("proOnto:IsMechanical");
        this.isPartition = Graph.CreateUriNode("proOnto:IsPartition");
        this.isStraight = Graph.CreateUriNode("proOnto:IsStraight");
        this.isStructural = Graph.CreateUriNode("proOnto:IsStructural");
        this.isWashroomPanel = Graph.CreateUriNode("proOnto:IsWashroomPanel");
        this.isSubIntersection = Graph.CreateUriNode("proOnto:IsSubIntersection");
        this.hasDoubleTopPlate = Graph.CreateUriNode("proOnto:HasDoubleTopPlate");
        this.level = Graph.CreateUriNode("proOnto:Level");
        this.material = Graph.CreateUriNode("proOnto:Material");
        this.name = Graph.CreateUriNode("proOnto:Name");
        this.stud_Spacing = Graph.CreateUriNode("proOnto:StudSpacing");
        this.stud_Size = Graph.CreateUriNode("proOnto:StudSize");
        this.size = Graph.CreateUriNode("proOnto:Size");
        this.hostWall = Graph.CreateUriNode("proOnto:HostWall");
        this.weight = Graph.CreateUriNode("proOnto:Weight");
        this.workZone = Graph.CreateUriNode("proOnto:ID");
    }

    private void SetUriNodesforInterrelationship()
    {
```

```
            this.connectedWith = Graph.CreateUriNode("proOnto:connectedWith");
            this.hasIntersection = Graph.CreateUriNode("proOnto:hasIntersection");
            this.hasSubIntersection =
Graph.CreateUriNode("proOnto:hasSubIntersection");
            this.hasDoor = Graph.CreateUriNode("proOnto:hasDoor");
            this.hasWindow = Graph.CreateUriNode("proOnto:hasWindow");
            this.hasSubComponent = Graph.CreateUriNode("proOnto:hasSubComponent");
            this.hostedBy = Graph.CreateUriNode("proOnto:hostedBy");
            this.isInstanceOf = Graph.CreateUriNode("proOnto:isInstanceOf");
            this.isMadeOf = Graph.CreateUriNode("proOnto:isMadeOf");
            this.isPartOf = Graph.CreateUriNode("proOnto:isPartOf");
            this.hasPattern = Graph.CreateUriNode("proOnto:hasPattern");
            this.supportedBy = Graph.CreateUriNode("proOnto:supportedBy");
        }

        private string NameOfKnowledgeBase { get; set; }

        private void AddNamespace()
        {
            Graph.NamespaceMap.AddNamespace("proOnto",

UriFactory.Create("http://www.semanticweb.org/hexu/ontologies/2014/6/BuildingProcessO
ntology#"));
            Graph.NamespaceMap.AddNamespace("rdf",
UriFactory.Create("http://www.w3.org/1999/02/22-rdf-syntax-ns#"));
            Graph.NamespaceMap.AddNamespace("rdfs",
UriFactory.Create("http://www.w3.org/2000/01/rdf-schema#"));
            Graph.NamespaceMap.AddNamespace("xsd",
UriFactory.Create("http://www.w3.org/2001/XMLSchema#"));
            Graph.NamespaceMap.AddNamespace("owl",
UriFactory.Create("http://www.w3.org/2002/07/owl#"));
        }

        public void AddNewEntitiesintoRDF<T>(IEnumerable<T> entities)
            where T : BuildingComponent
        {
            foreach (var wall in entities.OfType<Wall_Hexu>()) //SteelFramingWall
            {
                try
                {
                    SteelFramingWall wall1 = wall as SteelFramingWall;
                    var wallNode = AssertWallEntitiesToRDF<T>(wall);
                    AssertWindowEntitiesToRDF<T>(wall.SubComponents.Windows,
wallNode);
                    AssertWindowEntitiesToRDF<T>(wall.SubComponents.Doors, wallNode);
                    if (wall1 != null) AssertStudEntitiesToRDF<T>(wall1.Studs,
wallNode);
                    if (wall1 != null) AssertPlateEntitiesToRDF<T>(wall1.Plates,
wallNode);
                }
                catch (Exception ex)
                {
                    throw new UofAException("Triple cannot be created.",
wall.ElementID.ToString("F0"), ex);
                }
```

```
            }
            RdfXmlWriter rdfxmlWriter = new RdfXmlWriter();
            rdfxmlWriter.Save(Graph, NameOfKnowledgeBase);
        }

        public void AddNewSingleIntersectionsintoRDF(IEnumerable<Intersection>
intersections)
        {
            foreach (var intersection in intersections)
            {
                try
                {
                    IUriNode intersectionSubject =this.Graph.CreateUriNode
                        ("proOnto:Intersections" + ("_"+intersection.Name));
                    switch (intersection.Type)
                    {
                        case (IntersectionType.T):
                            this.Graph.Assert(new Triple(intersectionSubject, typeOf,
this.doubleTConnection));
                            break;
                        case (IntersectionType.L):
                            this.Graph.Assert(new Triple(intersectionSubject, typeOf,
this.lConnection));

this.AssertLiteralNode<BuildingComponent>(intersectionSubject, this.isSubIntersection,
                                intersection.IsSubIntersection ? this.booleanTrue :
this.booleanFalse);
                            break;
                        default:
                            this.Graph.Assert(new Triple(intersectionSubject, typeOf,
this.lConnection));
                            break;
                    }


this.AssertLiteralNodeWithDataType<BuildingComponent>(intersectionSubject, this.iD,
                            intersection.ID,
                            XmlSpecsHelper.XmlSchemaDataTypeString);


this.AssertLiteralNodeWithDataType<BuildingComponent>(intersectionSubject,
this.hostWall,
                           intersection.Host.ToString(),
                           XmlSpecsHelper.XmlSchemaDataTypeString);


this.AssertLiteralNodeWithDataType<BuildingComponent>(intersectionSubject,
this.primary,
                            intersection.GetPrimaryAngle().ToString("F2"),
                            XmlSpecsHelper.XmlSchemaDataTypeDouble);

                    foreach (var wallLocationPair in intersection.ContainedElements)
                    {
                        var wallHostNode =  this.Graph.GetUriNode("proOnto:" +
```

```
(wallLocationPair.Key.Family.Add(wallLocationPair.Key.ElementID.ToString("F0"))));
                        //var hostWall_1 = this.Graph.GetTriplesWithSubjectPredicate
                        //(wallHostNode, this.hasIntersection).FirstOrDefault();
                        //if (hostWall_1 != null) this.Graph.Retract(hostWall_1);
                        this.Graph.Assert(new Triple(wallHostNode,
this.hasIntersection, intersectionSubject));
                        //this.Graph.Assert(new Triple(intersectionSubject,
this.hostedBy, wallHostNode));
                    }
                }
                catch (Exception ex)
                {
                    throw new UofAException("Triple cannot be created.",
intersection.ID, ex);
                }
            }
            RdfXmlWriter rdfxmlWriter = new RdfXmlWriter();
            rdfxmlWriter.Save(Graph, NameOfKnowledgeBase);
        }

        public void AddNewMultipleIntersectionsintoRDF(IEnumerable<Intersection>
multiIntersections)
        {
            foreach (var multiIntersection in multiIntersections)
            {
                try
                {
                    var first = multiIntersection.SubIntersections.FirstOrDefault();
                    if (first == null) return;

                    IUriNode mIntersectionSubject = this.Graph.CreateUriNode
                        ("proOnto:MIntersections" + ("_" + first.Name));

                    this.Graph.Assert(new Triple(mIntersectionSubject, typeOf,
this.doubleTConnection));


this.AssertLiteralNodeWithDataType<BuildingComponent>(mIntersectionSubject, this.iD,
                        first.ID,
                        XmlSpecsHelper.XmlSchemaDataTypeString);


this.AssertLiteralNodeWithDataType<BuildingComponent>(mIntersectionSubject,
this.second,
                        multiIntersection.SecondaryAngle.ToString("F2"),
                        XmlSpecsHelper.XmlSchemaDataTypeDouble);


this.AssertLiteralNodeWithDataType<BuildingComponent>(mIntersectionSubject,
this.third,
                        multiIntersection.ThirdAngle.ToString("F2"),
                        XmlSpecsHelper.XmlSchemaDataTypeDouble);
```

```
                    foreach (Intersection subIntersection in
multiIntersection.SubIntersections)
                    {
                        var subIntersectionObject =
this.Graph.GetUriNode("proOnto:Intersections" + ("_" + subIntersection.Name));

                        this.Graph.Assert(new Triple(mIntersectionSubject,
this.hasSubIntersection, subIntersectionObject));
                    }

                }
                catch (Exception ex)
                {
                    throw new UofAException("Triple cannot be created.",
multiIntersection.ID, ex);
                }
            }
            RdfXmlWriter rdfxmlWriter = new RdfXmlWriter();
            rdfxmlWriter.Save(Graph, NameOfKnowledgeBase);
        }

        private void AssertStudEntitiesToRDF<T>(IEnumerable<HexuLibrary.Revit.Stud>
studs, IUriNode hostNode) where T : BuildingComponent
        {
            foreach (var stud in studs)
            {
                IUriNode studSubject =
                    this.Graph.CreateUriNode("proOnto:Studs" +
(stud.ElementID.ToString("F0")));//stud.Family.Add()

                switch (stud.StudFunction)
                {
                    case (Stud.StudFunction.Blocking):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.blocking));
                        break;
                    case (Stud.StudFunction.BottomPlate):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.bottomPlate));
                        break;
                    case (Stud.StudFunction.TopPlate):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.topPlate));
                        break;
                    case (Stud.StudFunction.Cripple):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.cripple));
                        break;
                    case (Stud.StudFunction.Header):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.header));
                        break;
                    case (Stud.StudFunction.Jack):
                        this.Graph.Assert(new Triple(studSubject, typeOf, this.jack));
                        break;
```

```
                    case (Stud.StudFunction.King):
                        this.Graph.Assert(new Triple(studSubject, typeOf, this.king));
                        break;
                    case (Stud.StudFunction.OC):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.normal));
                        break;
                    case (Stud.StudFunction.RoughSill):
                        this.Graph.Assert(new Triple(studSubject, typeOf, this.sill));
                        break;
                    case (Stud.StudFunction.SJoin):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.joint));
                        break;
                    default:
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.normal));
                        break;
                }

                this.Graph.Assert(new Triple(hostNode, this.hasSubComponent,
studSubject));

                this.Graph.Assert(new Triple(studSubject, this.isPartOf, hostNode));

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.iD,
                    stud.ElementID.ToString("F0"),
                    XmlSpecsHelper.XmlSchemaDataTypeString);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.hostID,
                    stud.HostElement.ElementID.ToString("F0"),
                    XmlSpecsHelper.XmlSchemaDataTypeInteger);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.height,
                    stud.Height.ToString("F2"),
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.length,
                    stud.b.ToString("F2"),
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.thickness,
                    stud.h.ToString("F2"),
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.size,
                    stud.Type.ToString(),
                    XmlSpecsHelper.XmlSchemaDataTypeString);
            }
        }

        private void AssertPlateEntitiesToRDF<T>(IEnumerable<Plate> plates, IUriNode
hostNode) where T : BuildingComponent
        {
            foreach (var stud in plates)
```

```
            {
                IUriNode studSubject =
                    this.Graph.CreateUriNode("proOnto:Plates" +
(stud.ElementID.ToString("F0")));//stud.Family.Add()

                switch (stud.PlateFunction)
                {
                    case (Stud.StudFunction.Blocking):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.blocking));
                        break;
                    case (Stud.StudFunction.BottomPlate):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.bottomPlate));
                        break;
                    case (Stud.StudFunction.TopPlate):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.topPlate));
                        break;
                    case (Stud.StudFunction.Cripple):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.cripple));
                        break;
                    case (Stud.StudFunction.Header):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.header));
                        break;
                    case (Stud.StudFunction.Jack):
                        this.Graph.Assert(new Triple(studSubject, typeOf, this.jack));
                        break;
                    case (Stud.StudFunction.King):
                        this.Graph.Assert(new Triple(studSubject, typeOf, this.king));
                        break;
                    case (Stud.StudFunction.OC):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.normal));
                        break;
                    case (Stud.StudFunction.RoughSill):
                        this.Graph.Assert(new Triple(studSubject, typeOf, this.sill));
                        break;
                    case (Stud.StudFunction.SJoin):
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.joint));
                        break;
                    default:
                        this.Graph.Assert(new Triple(studSubject, typeOf,
this.normal));
                        break;
                }

                this.Graph.Assert(new Triple(hostNode, this.hasSubComponent,
studSubject));

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.iD,
                    stud.ElementID.ToString("F0"),
```

```
                    XmlSpecsHelper.XmlSchemaDataTypeString);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.hostID,
                    stud.HostElement.ElementID.ToString("F0"),
                    XmlSpecsHelper.XmlSchemaDataTypeInteger);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.height,
                    stud.Height.ToString("F2"),
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.length,
                    stud.Length.ToString("F2"),
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);

                this.AssertLiteralNodeWithDataType<T>(studSubject, this.thickness,
                    stud.Width.ToString("F2"),
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);
            }
        }

        private void AssertWindowEntitiesToRDF<T>(IEnumerable<Openning> openings,
IUriNode hostNode)
            where T : BuildingComponent
        {
            foreach (var openning in openings)
            {
                IUriNode openingSubject =
                    this.Graph.CreateUriNode("proOnto:" +
(openning.Family.Add(openning.ElementID.ToString("F0"))));

                this.Graph.Assert(openning is Window_Hexu ? new Triple(openingSubject,
typeOf, this.window):
                    new Triple(openingSubject, typeOf, this.door));

                //this.Graph.Assert(new Triple(openingSubject, this.hostedBy,
hostNode));
                this.Graph.Assert(openning is Window_Hexu ? new Triple(hostNode,
this.hasWindow, openingSubject) :
                    new Triple(hostNode, this.hasDoor, openingSubject));


                this.AssertLiteralNodeWithDataType<T>(openingSubject, this.iD,
                    openning.ElementID.ToString("F0"),
                    XmlSpecsHelper.XmlSchemaDataTypeString);

                this.AssertLiteralNodeWithDataType<T>(openingSubject, this.hostID,
                    openning.HostID.ToString("F0"),
                    XmlSpecsHelper.XmlSchemaDataTypeInteger);

                this.AssertLiteralNodeWithDataType<T>(openingSubject, this.height,
                    openning.Height.ToString("F2"),
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);

                this.AssertLiteralNodeWithDataType<T>(openingSubject, this.length,
                    openning.Length.ToString("F2"),
```

```
                    XmlSpecsHelper.XmlSchemaDataTypeDouble);
            }
        }

        private IUriNode AssertWallEntitiesToRDF<T>(Wall_Hexu wall) where T :
BuildingComponent
        {
            SteelFramingWall wall1 = wall as SteelFramingWall;
            IUriNode buildingEntitySubject =
                this.Graph.CreateUriNode("proOnto:" +
(wall.Family.Add(wall.ElementID.ToString("F0"))));

            this.Graph.Assert(new Triple(buildingEntitySubject, typeOf, wall1 !=
null ? this.studFramedWall : this.concreteWall));//this.studFramedWall

            this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject, this.height,
                wall.Dimensions["Height"].ToString("F2"),
                XmlSpecsHelper.XmlSchemaDataTypeDouble);

            this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject, this.length,
                wall.Dimensions["Length"].ToString("F2"),
                XmlSpecsHelper.XmlSchemaDataTypeDouble);

            this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject,
this.thickness,
                wall.Dimensions["Width"].ToString("F2"),
                XmlSpecsHelper.XmlSchemaDataTypeDouble);

            this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject, this.volume,
                wall.Dimensions["Volume"].ToString("F2"),
                XmlSpecsHelper.XmlSchemaDataTypeDouble);

            this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject, this.iD,
                wall.ElementID.ToString("F0"),
                XmlSpecsHelper.XmlSchemaDataTypeString);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isAcoustic,
                wall.IsAcoustic ? this.booleanTrue : this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isExterior,
                wall.IsExterior ? this.booleanTrue : this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isFireRated,
                wall.IsFireRated ? this.booleanTrue : this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isMechanical,
                wall.IsMechanical ? this.booleanTrue : this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isPartition,
                wall.IsPartition ? this.booleanTrue : this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isStraight,
                wall.IsStraight ? this.booleanTrue : this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isStructural,
```

```
                wall.IsStructuralUsage ? this.booleanTrue : this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.isWashroomPanel,
                wall1 != null && wall1.IsBathroomWall ? this.booleanTrue :
this.booleanFalse);

            this.AssertLiteralNode<T>(buildingEntitySubject, this.hasDoubleTopPlate,
                wall1 != null && wall1.HasDoubleTopPlate ? this.booleanTrue :
this.booleanFalse);

            this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject, this.level,
                wall.Level,
                XmlSpecsHelper.XmlSchemaDataTypeString);

            this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject,
this.material,
               wall.StructuralMaterial,
               XmlSpecsHelper.XmlSchemaDataTypeString);

            if (wall1 != null)
            {
                double studSpacing = wall1.GetStudSpacing();
                this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject,
this.stud_Spacing,
                    studSpacing.ToString("F0"),
                    XmlSpecsHelper.XmlSchemaDataTypeString);

                 this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject,
this.name,
                    wall1.NameofWallPanel,
                XmlSpecsHelper.XmlSchemaDataTypeString);
            }

            if (wall1 != null)
            {
                string objectValue = wall1.GetStudSize();
                this.AssertLiteralNodeWithDataType<T>(buildingEntitySubject,
this.stud_Size,
                    objectValue,
                    XmlSpecsHelper.XmlSchemaDataTypeString);
            }

            return buildingEntitySubject;
        }

        private void AssertLiteralNodeWithDataType<T>(IUriNode subject, IUriNode
predict,
            string objectValue, string dataType)where T : BuildingComponent
        {
            var id = Graph.CreateLiteralNode(objectValue,
UriFactory.Create(dataType));
            this.AssertLiteralNode<T>(subject, predict, id);
        }

        private void AssertLiteralNode<T>(IUriNode subject, IUriNode predict,
```

```
            ILiteralNode objectValue) where T : BuildingComponent
        {
            var isExteriorTriple =
                this.Graph.GetTriplesWithSubjectPredicate(subject,
predict).FirstOrDefault();
            if (isExteriorTriple != null)
                this.Graph.Retract(isExteriorTriple);
            this.Graph.Assert(new Triple(subject, predict, objectValue));
        }
    }
}
```

APPENDIX D

C#.NET code of the PSO algorithm for schedule optimization:

```
namespace Addin_Fortis
{
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Text;
    using System.Windows.Forms;
    using HexuLibrary;
    using Simphony;
    using Simphony.General;
    using Simphony.Modeling;
    using Simphony.Simulation;
    using Autodesk.Revit.UI;
    using File = Simphony.General.File;


    /// <summary>
    ///     Initialized an instance of the class PSO_Optimizer_OnSite
    /// </summary>
    public class PSO_Optimizer_OnSite : IDisposable
    {
        /// <summary>
        ///     inertia weight 0.729
        /// </summary>
        private const double InertiaWeight = 0.9;

        /// <summary>
        ///     cognitive/local weight  1.49445
        /// </summary>
        private const double LocalWeight1 = 1;

        /// <summary>
        ///     social/global weight
        /// </summary>
        private const double LocalWeight2 = 2;

        /// <summary>
        ///     Evolutionary process of best global fitness
        /// </summary>
        private readonly List<double> bestGlobalFitness_History = new List<double>();

        /// <summary>
        ///     Evolutionary process of each particle, the key of dictionary is id of
each particle
        /// </summary>
        private readonly Dictionary<int, List<double>> bestLocalFitness_History = new
Dictionary<int, List<double>>();
```

```
        /// <summary>
        ///     File address to store the evolutionary process of best global fitness
        /// </summary>
        private readonly string best_GlobalFitness =
@"C:\BIM_UofA\Temp\Best_GlobalFitness" +
                                                    DateTime.Now.ToString()
                                                        .Replace("/", "_")
                                                        .Replace(":", "_")
                                                        .Replace(" ", "_") + ".txt";

        /// <summary>
        ///     Building components from Revit Model
        /// </summary>
        private readonly List<BuildingComponent> buildingEntities = new
List<BuildingComponent>();

        /// <summary>
        ///     Dimension of the optimized problem or the number of variables or
activities
        /// </summary>
        private readonly int dimension;

        /// <summary>
        ///     velocity boundary
        /// </summary>
        private readonly double maxV;

        /// <summary>
        ///     variable upper bounder
        /// </summary>
        private readonly double maxX;

        /// <summary>
        ///     velocity boundary
        /// </summary>
        private readonly double minV;

        /// <summary>
        ///     variable lower bounder
        /// </summary>
        private readonly double minX;

        /// <summary>
        ///     Simulation model (Object function calculator)
        /// </summary>
        private readonly Model modelforOptimization = new Model();

        private readonly int numberIterations;
        private readonly int numberParticles;

        /// <summary>
        ///     File address to store information of each particle in each iteration
        /// </summary>
        private readonly string optimizationResults =
@"C:\BIM_UofA\Temp\Optimization_Result" +
```

```
                                                    DateTime.Now.ToString()
                                                        .Replace("/", "_")
                                                        .Replace(":", "_")
                                                        .Replace(" ", "_") + ".txt";

        /// <summary>
        ///     a random number generator for updating particle velocity or position
        /// </summary>
        private readonly Random ran = new Random();

        /// <summary>
        ///     The value of object function
        /// </summary>
        private readonly Counter termination = new Counter();

        /// <summary>
        ///     XML file for Building components from Revit Model
        /// </summary>
        private readonly string xmlPath_Component = PSO_Interface.PathofXMLFile;

        private double bestGlobalFitness;

        /// <summary>
        ///     Array to store the best global position (best position of swarm)
        /// </summary>
        private double[,] bestGlobalPosition;

        private Execute controller_For_Location_Surv_Wall = new Execute();
        private Execute controller_For_Assembly_Floor = new Execute();
        private Execute controller_For_Lifting_Wall = new Execute();
        private int currentIteration;

        /// <summary>
        ///     Flag to indicating the dispose from outside of the class itself
        /// </summary>
        private bool disposed;

        /// <summary>
        ///     Start date of the construction project
        /// </summary>
        private DateTime projectStartDate;

        /// <summary>
        ///     cognitive and social randomizations
        /// </summary>
        private double socialRandomization_1, socialRandomization_2;

        private Particle[] swarm;

        /// <summary>
        ///     Initializes a new instance of the PSO_Optimizer_OnSite class
        ///     Initialize parameters such as number of particles, number of
iterations, dimension (number) of solution,
        ///     deserialize the simulation model.
        /// </summary>
```

```
        /// <param name="numberParticles">Number of Particles</param>
        /// <param name="numberIterations">Number of iterations</param>
        /// <param name="dim">number of variables - "Number of activities"</param>
        /// <param name="minVariable">variable min boundary</param>
        /// <param name="maxVariable">variable max boundary</param>
        /// <param name="componentsfromRevit">building components from Revit</param>
        public PSO_Optimizer_OnSite(
            int numberParticles,
            int numberIterations,
            int dim,
            double minVariable,
            double maxVariable,
            List<BuildingComponent> componentsfromRevit)
        {
            buildingEntities = componentsfromRevit;
            this.numberParticles = numberParticles;
            this.numberIterations = numberIterations;
            dimension = dim;    //// dimensions
            minX = minVariable; ////variable boundary  -100.0;
            maxX = maxVariable; ////variable boundary
            minV = -1*maxX;     ////velocity boundary
            maxV = maxX;        ////velocity boundary   this.Dim

            using (
                var stream =
                    new FileStream(
@"C:\Users\hexu.MODULAR\Dropbox\VS_2010\Revit\trunk\WFA_PSO\WFA_PSO\bin\Debug\Resourc
es\On_Site_Scheduling_2013_11_09_ActivityPriority.sim",
                        FileMode.Open))
            {
                modelforOptimization.Deserialize(stream);
                termination =
modelforOptimization.Scenarios[0].GetElement<Counter>("Terminate Simulation");
                projectStartDate = modelforOptimization.Scenarios[0].StartDate;

            }
        }

        /// <summary>
        ///     Gets or sets Best global fitness
        /// </summary>
        public double BestGlobalFitness
        {
            get { return bestGlobalFitness; }

            set { bestGlobalFitness = value; }
        }

        /// <summary>
        ///     Dispose the class instance
        /// </summary>
        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
```

```
        }

        /// <summary>
        ///     Finalizes an instance of the PSO_Optimizer_OnSite class
        ///     Destructor to release manageable and unmanageable resource
        /// </summary>
        ~PSO_Optimizer_OnSite()
        {
            Dispose(false);
        }

        /// <summary>
        ///     The priority setter for each component
        /// </summary>
        /// <param name="particlePosition">Priority array (position of
particle)</param>
        public void SetPriority_Component_Level(double[,] particlePosition)
        {
            buildingEntities.Sort((x, y) => x.ID.CompareTo(y.ID));
            for (int i = 0; i < buildingEntities.Count; i++)
            {
                buildingEntities[i].PriorityforResource = particlePosition[i, 0];
            }
            ////Sort the components according to the construction priority
            buildingEntities.Sort((x, y) =>
x.PriorityforResource.CompareTo(y.PriorityforResource));
            buildingEntities.Reverse();
            XMLHelper.SerializeToXML(xmlPath_Component, buildingEntities);
            ////End the components according to the construction priority
        }

        /// <summary>
        ///     The priority setter for each activity
        /// </summary>
        /// <param name="particlePosition">Priority array (position of
particle)</param>
        public void SetPriority_Workpackage_Level(double[,] particlePosition)
        {
            double[,] sortedIndex = new double[dimension, 2];
            particlePosition.CopyTo(sortedIndex);
            sortedIndex = sortedIndex.OrderBy(x => x[0]);
            buildingEntities.Sort((x, y) => x.ID.CompareTo(y.ID));
            List<Workpackage> workpackages =
                buildingEntities.SelectMany(
                    buildingEntity =>
                        buildingEntity.Workpackages.Where(package
=> !package.ID.Contains("Curing"))
                            .OrderBy(package => package.ID)).ToList();
            ////(from component in this.components
            ////from workPackage in component.Workpackages
            ////where (!workPackage.ID.Contains("Curing"))
            ////select workPackage).OrderBy(package => package.ID).ToList();

            // workpackages[(int)sortedIndex[idworkpackage, 1]] ---- sort work
package
```

```
            for (int idworkpackage = 0; idworkpackage < workpackages.Count;
idworkpackage++)
            {
                workpackages[(int) sortedIndex[idworkpackage, 1]].PriorityforResource
= idworkpackage + 1;
            }

            ////Sort the components according to the construction priority
            buildingEntities.Sort(
                (x, y) =>
x.Workpackages[0].PriorityforResource.CompareTo(y.Workpackages[0].PriorityforResource
));
            buildingEntities.Reverse();
            XMLHelper.SerializeToXML(xmlPath_Component, buildingEntities);
            ////End the components according to the construction priority
        }

        /// <summary>
        ///     Initialize each particle
        /// </summary>
        public void Initialize_Particles()
        {
            try
            {
                ////this.ran = new Random();  No seed
                currentIteration = 0;
                swarm = new Particle[numberParticles];
                bestGlobalPosition = new double[dimension, 2];
                    // best solution found by any particle in the swarm. implicit
initialization to all 0.0

                ////double[,] sortedIndex = bestGlobalPosition.OrderBy(x => x[0]);
                ////Position is only one dimension of the above array
                bestGlobalFitness = double.MaxValue; ////smaller values better
                bestGlobalFitness_History.Add(bestGlobalFitness);
                //// initialize each Particle in the swarm
                for (int i = 0; i < swarm.Length; ++i)
                {
                    double[,] randomPosition =
RandomNumberUti.RandomPriorityPositionGenerator(dimension, ran, minX, maxX);
                        ////OrderBy(x => x[0]);
                    this.SetPriority_Workpackage_Level(randomPosition);

                    ////initializeSimulation.Expression.Function =
InitializeSimulationModel;
                    modelforOptimization.Simulate();

                    double fitness = termination.LastTime.Mean;

                    double[] randomVelocity = new double[dimension];
                    for (int j = 0; j < randomVelocity.Length; ++j)
                    {
                        double lo = -1.0*Math.Abs(maxX - minX);
                        double hi = Math.Abs(maxX - minX);
```

```
                    randomVelocity[j] = ((hi - lo)*ran.NextDouble()) + lo;
                    ////randomVelocity[j] = ran.Next(-(maxX - 1), maxX);
                }

                swarm[i] = new Particle(i, randomPosition, fitness,
randomVelocity, randomPosition, fitness);
                swarm[i].BestLocalFitness_History.Add(fitness);
                //// does current Particle have global best position/solution?
                if (swarm[i].Fitness < bestGlobalFitness)
                {
                    bestGlobalFitness = swarm[i].Fitness;
                    ////bestGlobalPosition = swarm[i].position;
                    swarm[i].Position.CopyTo(bestGlobalPosition);
                }

                TextHelper.Write(swarm[i].ToString(), optimizationResults);
            }
            //// initialization
            TextHelper.Write(
                bestGlobalFitness + ",   " + projectStartDate.Add(new TimeSpan(0,
0, (int) bestGlobalFitness, 0)),
                best_GlobalFitness);
            bestGlobalFitness_History.Add(bestGlobalFitness);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, @"Fatal error in Initialization: ");
            //// each Particle
            for (int i = 0; i < swarm.Length; ++i)
            {
                bestLocalFitness_History.Add(i + 1,
swarm[i].BestLocalFitness_History);
            }
            bestLocalFitness_History.ExportToExcel();
        }
    }

    /// <summary>
    ///     Initialize particles when continue the optimization
    /// </summary>
    public void Initialize_Particles_Continue()
    {
        try
        {
            currentIteration = 0;

            swarm = new Particle[numberParticles];
            bestGlobalPosition = new double[dimension, 2];
            //// best solution found by any particle in the swarm. implicit
initialization to all 0.0

            ////double[,] sortedIndex = bestGlobalPosition.OrderBy(x => x[0]);
            ////Position is only one dimension of the above array
            bestGlobalFitness = double.MaxValue; ////smaller values better
            bestGlobalFitness_History.Add(bestGlobalFitness);
```

```
                List<Particle> activities =
XMLHelper.DeserializeFromXML_Particle(@"C:\BIM_UofA\Temp\Particles.xml");
                //// initialize each Particle in the swarm
                for (int i = 0; i < swarm.Length; ++i)
                {
                    var id = activities[i].ID;
                    double[] temp_Position = new
double[activities[i].Position_Serializable.GetLength(0)];
                    activities[i].Position_Serializable.CopyTo(temp_Position, 0);

                    double fitness = activities[i].Fitness;
                    double[] temp_Velocity = new
double[activities[i].Position_Serializable.GetLength(0)];
                    activities[i].Velocity.CopyTo(temp_Velocity, 0);
                    double[] temp_BestPosition = new
double[activities[i].Position_Serializable.GetLength(0)];
                    activities[i].BestPosition_Serializable.CopyTo(temp_BestPosition,
0);
                    double fitness_Best = activities[i].BestFitness;
                    swarm[i] = new Particle(
                        id,
                        temp_Position.AddIndexasLastColunm(),
                        fitness,
                        temp_Velocity,
                        temp_BestPosition.AddIndexasLastColunm(),
                        fitness_Best);

                    swarm[i].BestLocalFitness_History.Add(swarm[i].Fitness);
                    //// does current Particle have global best position/solution?
                    if (swarm[i].Fitness < bestGlobalFitness)
                    {
                        bestGlobalFitness = swarm[i].Fitness;
                        ////bestGlobalPosition = swarm[i].position;
                        swarm[i].Position.CopyTo(bestGlobalPosition);
                    }

                    TextHelper.Write(swarm[i].ToString(), optimizationResults);
                }
                //// initialization
                ////TextWritor.TxtWritor(bestGlobalFitness.ToString() + ",    " +
fitness_DateTime+"\r\n++++++++++++++++++++++++++++++++++++++++++++++++++",
OptResults);
                TextHelper.Write(
                    bestGlobalFitness + ",    " + projectStartDate.Add(new TimeSpan(0,
0, (int) bestGlobalFitness, 0)),
                    best_GlobalFitness);
                bestGlobalFitness_History.Add(bestGlobalFitness);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Fatal error in Initialization: ");
                //// each Particle
                for (int i = 0; i < swarm.Length; ++i)
                {
```

```
                    bestLocalFitness_History.Add(i + 1,
swarm[i].BestLocalFitness_History);
                }

                bestLocalFitness_History.ExportToExcel();
            }
        }

        /// <summary>
        ///     Iteration of PSO
        /// </summary>
        public void PSO_Iteration()
        {
            try
            {
                ////enter into the mail loop for optimization
                ////bool isNotImproving = false;
                ////this.ran = new Random();
                while (currentIteration < numberIterations)
                {
                    ++currentIteration;

                    double[] newVelocity = new double[dimension];
                    double[,] newPosition = new double[dimension, 2];
                    //// each Particle
                    foreach (Particle currP in swarm)
                    {
                        currP.IterationID = currentIteration;
                        //// each x value(dimension) of the velocity
                        for (int j = 0; j < currP.Velocity.Length; ++j)
                        {
                            socialRandomization_1 = ran.NextDouble();
                            socialRandomization_2 = ran.NextDouble();

                            newVelocity[j] = ((InertiaWeight -
(0.2/numberIterations*currentIteration))*
                                              currP.Velocity[j]) +
                                             (LocalWeight1*socialRandomization_1*
                                              (currP.BestPosition[j, 0] -
currP.Position[j, 0])) +
                                             (LocalWeight2*socialRandomization_2*
                                              (bestGlobalPosition[j, 0] -
currP.Position[j, 0]));

                            if (newVelocity[j] < minV)
                            {
                                newVelocity[j] = minV;
                            }
                            else if (newVelocity[j] > maxV)
                            {
                                newVelocity[j] = maxV;
                            }
                        }

                        newVelocity.CopyTo(currP.Velocity, 0);
```

```
                        for (int j = 0; j < currP.Position.GetLength(0); ++j)
                        {
                            newPosition[j, 0] = currP.Position[j, 0] + newVelocity[j];
                            newPosition[j, 1] = j;
                            if (newPosition[j, 0] < minX)
                            {
                                newPosition[j, 0] = minX;
                            }
                            else if (newPosition[j, 0] > maxX)
                            {
                                newPosition[j, 0] = maxX;
                            }
                        }

                        newPosition.CopyTo(currP.Position);
                        ////Set sequence in simulation and run simulation (OB)
                        this.SetPriority_Workpackage_Level(newPosition);

                        modelforOptimization.Simulate();
                        double newFitness = termination.LastTime.Mean;
                        ////end simulation

                        ////fitness_DateTime = Termination.Engine.DateNow;
                        currP.Fitness = newFitness;
                        currP.BestLocalFitness_History.Add(newFitness);

                        if (newFitness < currP.BestFitness)
                        {
                            ////currP.bestPosition = newPosition;
                            newPosition.CopyTo(currP.BestPosition);
                            currP.BestFitness = newFitness;
                        }

                        if (newFitness < bestGlobalFitness)
                        {
                            ////this.bestGlobalPosition = newPosition;
                            newPosition.CopyTo(bestGlobalPosition);
                            bestGlobalFitness = newFitness;
                        }
                        //currP.Position_Serialization();

////XMLWritor.SerializeToXML(@"C:\BIM_UofA\Temp\Best_Position.xml",
this.bestGlobalFitness );
                        if (currentIteration < 5)
                        {
                            TextHelper.Write(currP.ToString(), optimizationResults);
                        }
                    }
                    //// each Particle
                    TextHelper.Write(
                        bestGlobalFitness + ",   " +
                        projectStartDate.Add(new TimeSpan(0, 0, (int)
bestGlobalFitness, 0)), best_GlobalFitness);
                    bestGlobalFitness_History.Add(bestGlobalFitness);
                    ////if (this.iteration == 75)
```

```
                    ////{
                    ////    this.Write_Best_LocalFitness_Excel();
                    ////    this.bestLocalFitness_History.Clear();
                    ////}
                    //// each iteration
                }
                //// while
                SetPriority_Workpackage_Level(bestGlobalPosition);
                XMLHelper.SerializeToXML(@"C:\BIM_UofA\Temp\Particles.xml",
swarm.ToList());
                Write_Best_LocalFitness_Excel();
                Write_Best_LocalFitness_TXT();
            }
            catch (ModelExecutionException ex)
            {
                if (bestLocalFitness_History.Count != 0)
                {
                    Write_Best_LocalFitness_Excel();
                }

                Write_Best_LocalFitness_TXT();
                MessageBox.Show(ex.Message + "\r\n" + ex.Context, "Simulation Error
in Iteration: ");
                throw;
            }
        }

        /// <summary>
        ///     Protected implementation of Dispose pattern.
        /// </summary>
        /// <param name="disposing">Flag for outside calling or inside calling</param>
        protected virtual void Dispose(bool disposing)
        {
            if (disposed)
            {
                return;
            }

            if (disposing)
            {
                //// Free any other managed objects here.
                termination.Dispose();
                ////this.initializeSimulation.Dispose();
            }
            //// Free any unmanaged objects here.

            disposed = true;
        }

        private void Write_Best_LocalFitness_Excel()
        {
            for (int i = 0; i < swarm.Length; ++i)
            {
                bestLocalFitness_History.Add(i + 1,
swarm[i].BestLocalFitness_History);
```

```
            }

            bestLocalFitness_History.ExportToExcel();
        }

        private void Write_Best_LocalFitness_TXT()
        {
            StringBuilder result = new StringBuilder();
            foreach (KeyValuePair<int, List<double>> item in bestLocalFitness_History)
            {
                foreach (double value in item.Value)
                {
                    result.Append(value + ",");
                }

                result.Append("\r\n");
            }

            TextHelper.Write(result.ToString(),
                @"C:\BIM_UofA\Temp\Best_LocalFitness" +
                DateTime.Now.ToString().Replace("/", "_").Replace(":", "_").Replace("
", "_") + ".txt");
        }

        #region Execute code
        private bool Controller_for_Lifting_Wall_Formula(Execute context)
        {
            Counter counter_Lift_Wall =
context.Scenario.GetElement<Counter>("Counter_Lift_Wall");

            BuildingComponent currentEntity = context.CurrentEntity.Objects[0] as
BuildingComponent;
            List<BuildingComponent> components = context.Scenario.Objects[0] as
List<BuildingComponent>;

            List<BuildingComponent> bathRooms = components.Where(e => e.Type ==
"BathRoom").ToList();
            foreach (BuildingComponent bathRoom in bathRooms)
            {
                var survey_Predecessor =
                    bathRoom.Workpackages.FirstOrDefault(e => e.Name == "Survey Panel
Location");
                bool survey_Predecessor_HasOrNot = survey_Predecessor != null ? true :
false;
                bool survey_Predecessor_Done = survey_Predecessor_HasOrNot &&
survey_Predecessor.CompletedOrNot;
                Workpackage toBeStartedActivity =
                    bathRoom.Workpackages.FirstOrDefault(e => e.Name == "Lift Wall
Panel");
                //only applied the below codes to unfinished bathroom Unit
                if (toBeStartedActivity.LogicAddedOrNot != true)
                {
                    //Check wheather all the supporting elements are done or not
                    if (CheckAllSupportsisDone(components, bathRoom) &&
                        (survey_Predecessor_Done || !survey_Predecessor_HasOrNot))
```

```
                    {
                        //Update the completed list
                        toBeStartedActivity.LogicAddedOrNot = true;
                        if (currentEntity.Type == "BathRoom")
                        {

AddWallorBathRoomasPredecessorforLiftingWall(currentEntity, toBeStartedActivity,
                                survey_Predecessor_HasOrNot);
                        }

                        if (currentEntity.Type == "Floor")
                        {
                            AddFloorasPredecessorforLiftingWall(currentEntity,
toBeStartedActivity);
                        }

                        //context.Scenario.Objects[0] = components;
                        //End Update the completed list
                        GeneralEntity newEntity = new GeneralEntity();
                        newEntity.Objects[0] = bathRoom;
                        counter_Lift_Wall.InputPoint.TransferIn(newEntity);
                    }
                }
            }
            //End deal with bathroom

            //Deal with wall panel
            List<BuildingComponent> walls = components.Where(e => e.Type ==
"Wall").ToList();
            foreach (BuildingComponent wall in walls)
            {
                var survey_Predecessor =
                    wall.Workpackages.FirstOrDefault(e => e.Name == "Survey Panel
Location");
                bool survey_Predecessor_HasOrNot = survey_Predecessor != null;
                bool survey_Predecessor_Done = survey_Predecessor_HasOrNot &&
survey_Predecessor.CompletedOrNot;

                Workpackage toBeStartedActivity =
                    wall.Workpackages.FirstOrDefault(e => e.Name == "Lift Wall
Panel");
                //only applied the below codes to unfinished wall panel
                if (toBeStartedActivity.LogicAddedOrNot != true)
                {
                    if (CheckAllSupportsisDone(components, wall) &&
                        CheckOneConnectionisDone(components, wall, "Lift Wall Panel")
&&
                        (survey_Predecessor_Done || !survey_Predecessor_HasOrNot))
                    {
                        //Update the completed list
                        toBeStartedActivity.LogicAddedOrNot = true;
                        if (currentEntity.Type == "Foundation")
                        {
                            toBeStartedActivity.Predecessors.Add(
```

```
                                currentEntity.Workpackages.Where(e => e.Name ==
"Curing Foundation").FirstOrDefault().ID);
                        }

                        if (currentEntity.Type == "Floor")
                        {
                            AddFloorasPredecessorforLiftingWall(currentEntity,
toBeStartedActivity);
                        }

                        if (currentEntity.Type == "Wall" || currentEntity.Type ==
"BathRoom")
                        {

AddWallorBathRoomasPredecessorforLiftingWall(currentEntity, toBeStartedActivity,
                                survey_Predecessor_HasOrNot);
                        }
                        //context.Scenario.Objects[0] = components;
                        //End Update the completed list
                        GeneralEntity newEntity = new GeneralEntity();
                        newEntity.Objects[0] = wall;
                        counter_Lift_Wall.InputPoint.TransferIn(newEntity);
                    }
                }
            }

            return true;
        }

        private static void
AddWallorBathRoomasPredecessorforLiftingWall(BuildingComponent currentEntity,
            Workpackage toBeStartedActivity, bool survey_Predecessor_HasOrNot)
        {
            if (currentEntity.Workpackages.FirstOrDefault(e => e.Name == "Lift Wall
Panel").CompletedOrNot)
            {
                toBeStartedActivity.Predecessors.Add(
                    currentEntity.Workpackages.FirstOrDefault(e => e.Name == "Lift
Wall Panel").ID);
            }
            else
            {
                if (survey_Predecessor_HasOrNot)
                {
                    toBeStartedActivity.Predecessors.Add(
                        currentEntity.Workpackages.FirstOrDefault(e => e.Name ==
"Survey Panel Location").ID);
                }
            }
        }

        private static void AddFloorasPredecessorforLiftingWall(BuildingComponent
currentComponent,
            Workpackage toBeStartedActivity)
        {
```

```
            if (currentComponent.StructuralMaterial.Contains("Concrete"))
            {
                toBeStartedActivity.Predecessors.Add(
                    currentComponent.Workpackages.Where(e => e.Name == "Curing
Slab").FirstOrDefault().ID);
            }
            else
            {
                toBeStartedActivity.Predecessors.Add(
                    currentComponent.Workpackages.Where(e => e.Name == "Assembly
Joist").FirstOrDefault().ID);
            }
        }

        /// <summary>
        ///     Add predecessors and trigger next activities
        /// </summary>
        /// <param name="context">The Controller Execute Element</param>
        /// <param name="currentComponent">The component triggered the
controller</param>
        /// <param name="components">All building components</param>
        /// <param name="toBeStartedComponent">To be launched component</param>
        /// <param name="tech_Condition">Technical condiction</param>
        private static void AddPredecessorsforSurveyandTiggerNextActivities(Execute
context, BuildingComponent currentComponent,
            IEnumerable<BuildingComponent> components, BuildingComponent
toBeStartedComponent, bool tech_Condition)
        {
            if (!tech_Condition) return;
            //Update the completed list
            Workpackage surveyActivity =
                components.FirstOrDefault(e => e.ID ==
toBeStartedComponent.ID).Workpackages.FirstOrDefault(e => e.Name == "Survey Panel
Location");
            surveyActivity.LogicAddedOrNot = true;
            if (currentComponent.Type == "Foundation")
            {
                surveyActivity.Predecessors.Add(
                    currentComponent.Workpackages.FirstOrDefault(e => e.Name ==
"Curing Foundation").ID);
            }

            if (currentComponent.Type == "Floor")
            {
                if (currentComponent.StructuralMaterial.Contains("Concrete"))
                {
                    surveyActivity.Predecessors.Add(
                        currentComponent.Workpackages.FirstOrDefault(e => e.Name ==
"Curing Slab").ID);
                }
                else
                {
                    surveyActivity.Predecessors.Add(
                        currentComponent.Workpackages.FirstOrDefault(e => e.Name ==
"Assembly Joist").ID);
```

```
                }
            }

            //context.Scenario.Objects[0] = components;
            //End Update the completed list
            Counter wallCounter = context.Scenario.GetElement<Counter>("Wall");
            GeneralEntity newEntity = new GeneralEntity();
            newEntity.Objects[0] = toBeStartedComponent;
            wallCounter.InputPoint.TransferIn(newEntity);
        }

        private static bool CheckOneConnectionisDone(List<BuildingComponent>
components, BuildingComponent wall,
            string toBeChecedActivity)
        {
            bool oneConnectionisDone = false;
            foreach (int connection in wall.Connections)
            {
                BuildingComponent connectionComponent = components.FirstOrDefault(e =>
e.ID == connection);

                if (
                    connectionComponent.Workpackages.FirstOrDefault(e => e.Name ==
toBeChecedActivity).CompletedOrNot)
                {
                    oneConnectionisDone = true;
                }
            }

            return oneConnectionisDone;
        }

        private static bool CheckAllSupportsisDone(List<BuildingComponent> components,
BuildingComponent bathRoomOrWall)
        {
            //Check wheather all the supporting elements are done or not
            bool allSupportsisDone = true;
            foreach (int support in bathRoomOrWall.Supports)
            {
                BuildingComponent supportingComponent = components.FirstOrDefault(e =>
e.ID == support);
                if (supportingComponent.Type == "Floor")
                {
                    //Second floor wash room
                    if (!supportingComponent.StructuralMaterial.Contains("Concrete"))
                    {
                        var firstOrDefault =
supportingComponent.Workpackages.FirstOrDefault(e => e.Name == "Assembly Joist");
                        if (firstOrDefault != null && firstOrDefault.CompletedOrNot
== false)
                        {
                            allSupportsisDone = false;
                        }
                    }
                    else
```

```
                    {
                        var firstOrDefault =
supportingComponent.Workpackages.FirstOrDefault(e => e.Name == "Curing Slab");
                        if (
                            firstOrDefault != null && firstOrDefault.CompletedOrNot
== false)
                        {
                            allSupportsisDone = false;
                        }
                    }
                }

                if (supportingComponent.Type == "Foundation")
                {
                    var firstOrDefault =
supportingComponent.Workpackages.FirstOrDefault(e => e.Name == "Curing Foundation");
                    if (
                        firstOrDefault != null && firstOrDefault.CompletedOrNot ==
false)
                    {
                        //MessageBox.Show(bathRoom.Description);
                        allSupportsisDone = false;
                    }
                }
            }

            return allSupportsisDone;
        }

        private bool Controller_for_Assembly_Floor_Formula(Execute context)
        {
            Counter counter_Floor = context.Scenario.GetElement<Counter>("Floor");

            BuildingComponent currentComponent = context.CurrentEntity.Objects[0] as
BuildingComponent;
            List<BuildingComponent> components = context.Scenario.Objects[0] as
List<BuildingComponent>;

            //First Level Floor
            foreach (BuildingComponent slab in components.Where(e => e.Type ==
"Floor" && e.StructuralMaterial.Contains("Concrete")))
            {
                Workpackage curingActivity = slab.Workpackages.Where(e => e.Name ==
"Curing Slab").FirstOrDefault();
                if (curingActivity.LogicAddedOrNot != true &&
                    curingActivity.CompletedOrNot != true)
                {
                    bool supportsisDone = true;
                    foreach (int support in slab.Supports)
                    {
                        BuildingComponent supportingComponent = components.Where(e =>
e.ID == support).FirstOrDefault();
                        if (supportingComponent == null)
                        {
```

```
                            throw new Exception("Cannot find supporting element for
the slab");
                        }

                        if (
                            supportingComponent.Workpackages.Where(e => e.Name ==
"Frame Wall Panel")
                                .FirstOrDefault().CompletedOrNot == false)
                        {
                            supportsisDone = false;
                        }
                    }

                    if (supportsisDone)
                    {
                        //Update the completed list
                        curingActivity.LogicAddedOrNot = true;
                        //context.Scenario.Objects[0] = components;
                        //End Update the completed list
                        //Simphony.General.Counter wallCounter =
context.Scenario.GetElement<Simphony.General.Counter>("Counter_Lift_Wall");
                        GeneralEntity newEntity = new GeneralEntity();
                        newEntity.Objects[0] = slab;
                        counter_Floor.InputPoint.TransferIn(newEntity);
                    }
                }
            }
            //Deal with higher floors
            foreach (BuildingComponent floor in components.Where(e => e.Type ==
"Floor" && !e.StructuralMaterial.Contains("Concrete")))
            {
                Workpackage liftingActivity = floor.Workpackages.Where(e => e.Name ==
"Lift Joists").FirstOrDefault();

                if (liftingActivity.LogicAddedOrNot != true &&
                    liftingActivity.CompletedOrNot != true)
                {
                    bool supportsisDone = true;
                    foreach (int support in floor.Supports)
                    {
                        BuildingComponent supportingComponent = components.Where(e =>
e.ID == support).FirstOrDefault();
                        if (supportingComponent == null)
                        {
                            throw new Exception("Cannot find supporting element for
the floor");
                        }

                        if (
                            supportingComponent.Workpackages.Where(e => e.Name ==
"Frame Wall Panel")
                                .FirstOrDefault().CompletedOrNot == false)
                        {
                            supportsisDone = false;
                        }
```

```
                    }

                    if (supportsisDone)
                    {
                        //MessageBox.Show("Start Floor Assembly");
                        //Update the completed list
                        liftingActivity.LogicAddedOrNot = true;
                        liftingActivity.Predecessors.Add(
                            currentComponent.Workpackages.Where(e => e.Name == "Frame
Wall Panel").FirstOrDefault().ID);
                        //End Update the completed list
                        //Simphony.General.Counter wallCounter =
context.Scenario.GetElement<Simphony.General.Counter>("Counter_Lift_Wall");
                        GeneralEntity newEntity = new GeneralEntity();
                        newEntity.Objects[0] = floor;
                        counter_Floor.InputPoint.TransferIn(newEntity);
                    }
                }
            }

            return true;
        }

        private bool Controller_for_Location_Surv_Wall_Formula(Execute context)
        {
            try
            {
                BuildingComponent currentComponent = context.CurrentEntity.Objects[0]
as BuildingComponent;
                List<BuildingComponent> components = context.Scenario.Objects[0] as
List<BuildingComponent>;

                //End deal with bathroom
                //List<Component> bathRooms = .ToList();
                foreach (BuildingComponent bathRoom in components.Where(e => e.Type
== "BathRoom"))
                {
                    //only applied the below codes to unfinished bathroom Unit
                    if (bathRoom.Workpackages.Where(e => e.Name == "Survey Panel
Location").FirstOrDefault() == null)
                    {
                        continue;
                    }

                    if (
                        bathRoom.Workpackages.Where(e => e.Name == "Survey Panel
Location")
                            .FirstOrDefault().LogicAddedOrNot != true)
                    {
                        bool allSupportsisDone = CheckAllSupportsisDone(components,
bathRoom);

                        AddPredecessorsforSurveyandTiggerNextActivities(context,
currentComponent, components, bathRoom,
                            allSupportsisDone);
```

```
                    }
                }

                //Deal with wall panel
                //List<Component> walls = .ToList();
                foreach (BuildingComponent wall in components.Where(e => e.Type ==
"Wall"))
                {
                    //only applied the below codes to unfinished wall panel
                    if (wall.Workpackages.Where(e => e.Name == "Survey Panel
Location").FirstOrDefault() == null)
                    {
                        continue;
                    }

                    if (
                        wall.Workpackages.Where(e => e.Name == "Survey Panel
Location").FirstOrDefault().LogicAddedOrNot !=
                        true)
                    {
                        bool allSupportsisDone = CheckAllSupportsisDone(components,
wall);

                        bool oneConnectionisDone =
CheckOneConnectionisDone(components, wall, "Survey Panel Location");

                        AddPredecessorsforSurveyandTiggerNextActivities(context,
currentComponent, components, wall,
                            allSupportsisDone && oneConnectionisDone);
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Simulation Error");
                throw;
            }

            return true;
        }
        #endregion
    }
}
```