A BIM COMPATIBLE SCHEMA FOR ARCHITECTURAL PROGRAMMING INFORMATION

A Dissertation

by

EHSAN BAREKATI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2016

Major Subject: Architecture

ABSTRACT

Having access to architectural programming information that documents the pre-design requirements for the building throughout the lifecycle of a building can add value to design evaluation, facility management, renovation and extension. Building Information Modeling (BIM) is a new approach to manage the information that represents the characteristics of a building throughout its lifecycle. Architectural programming, although a key part of Architecture/Engineering/Construction/Operations (AECO) processes, has not been well integrated into Building Information Modeling (BIM) standards. Although research for many years has established computational data models for a BIM, there is not yet a comprehensive and standard data model to store architectural programming information that is compatible with BIM data modeling standards. This study investigated the possibility for a Universal Format for an Architectural Program Of Requirements (UFPOR) that can connect the architectural programming information to the BIM.

Three well-known formats for architectural programming were analyzed to produce data models representing each format. The data models were further analyzed and compared to form a common data model that can bring together all three formats, producing a Universal Format for Architectural Program Of Requirements (UFPOR). The capabilities of UFPOR in representing PORs from the industry was further analyzed by modeling a POR excerpt using UFPOR.

In the next phase, the data schema for the industry standard International

Foundation Classes (IFC), which represents Building Information Models, was analyzed to explore its capabilities in supporting architectural programming information. Previous attempts to use IFC in documenting architectural programming information were reviewed and analyzed. The findings were compared with UFPOR to evaluate the capabilities of IFC in supporting a universal format for architectural programming. The result was a subset of IFC that can partially represent UFPOR (UFPOR-IFC). The limitations of IFC to fully support UFPOR are also discussed.

In the next phase, a computer application prototype (Target) was developed based on the findings of the previous phase to demonstrate how a limited subset of architectural programming requirements can be represented in IFC based on UFPOR-IFC. The output of the application was tested to ensure the IFC physical files produced by the application accurately reflect the data inserted by the application's user. UFPOR-IFC was tested by modeling two architectural programming documents to assess the external credibility of the findings. The devised system offers a blueprint for creating a link between real-world Architectural Programs and BIM models through IFC physical files.

For future work, the findings of the study can be further analyzed by testing the extensibility of the UFPOR-IFC in modeling architectural programming requirements in different subdomains such as lighting, earthquake resilience, and acoustics. Enhancement of IFC to fully support UFPOR could also be investigated and devised. The capabilities of UFPOR-IFC to create a link between UFPOR information and other BIM models should be further tested and analyzed.

DEDICATION

The list of people who made producing this dissertation a possible task is not limited to the individuals whose names have been cited to acknowledge their academic and scientific contributions. It also includes family members and friends who created a supporting environment for me to finish this task.

Over the past seven years, my wife Jenny created an environment where I could focus on finishing this dissertation. It was not always easy and took a lot of sacrifice and dedication on her part. Before that, my parents taught me to dream big and stood by me even when my dreams created a physical distance between us.

I dedicate this dissertation to my wife and parents without whom finishing this task was not possible.

ACKNOWLEDGEMENTS

I would like to thank Dr. Mark Clayton, the chair of my committee, not only for his exemplary depth of knowledge and his commitment to elevate the final product to its full extent, but also for being a friend and like an older brother over the past seven years. Without his support and dedication finishing this dissertation would not have been possible.

I would like to thank my committee members (Dr. Miranda, Dr. Yan, and Dr. Caverlee), who provided a wide range of expertise and knowledge without which this dissertation would have been severely lacking.

I would like to thank Texas A&M University and the College of Architecture faculty and staff who provided an environment where learning and exploring became my second nature.

I would like to thank the great people of College Station who embraced me upon my arrival and helped me in all forms and shapes expecting nothing in return.

NOMENCLATURE

AECO	Architecture/ Engineering/ Construction/ Operations
BIM	Building Information Modeling
BPie	Building Programming Information Exchange
CAD	Computer Aided Design (Drafting)
COBie	Construction Operations Building Information Exchange
IFC	Industry Foundation Classes
OOP	Object Oriented Programming
POR	Program of Requirements
SCie	Spatial Compliance Information Exchange
UFPOR	Universal Format for Program of Requirements

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF CODES

1. INTRODUCTION

The Program Of Requirements (POR) is the key document that defines the goals for a building design project and thus guides all subsequent design and construction activities. Current practices in the Architecture/Engineering/Construction/Operations (AECO) industry do not effectively document and store POR information to facilitate seamless compliance of a design and a building with the POR (Kiviniemi, 2005). Building Information Modeling (BIM) is widely expected to provide an information repository that can support many, if not all, of the information retrieval and processing needs of design, construction and operations (Eastman, 2008). Adoption of BIM within the United States building industry reached over 60% of architectural firms and construction companies by 2012 (Bernstein et al., 2012). This research investigates what extensions to BIM are needed to adequately incorporate and integrate POR information for use in later stages in building design, construction, and operations.

1.1. MOTIVE

The current state of the construction industry is far away from being a healthy practice (LePatner, 2008). The level of non-value added effort and waste is severely high in the business model (National Institute of Building Sciences, 2007). BIM has proven to be a path toward a more efficient AECO industry. BIM supports all of the stakeholders in the AECO industry to make fact-supported decisions more quickly by providing access to the required data at any given time throughout the building lifecycle (Eastman, 2008).

The emergence of Building Information Modeling has been very influential to the AECO industry. One of the cornerstones of BIM is the product modeling techniques and standards that are a necessity to accomplish the premises of BIM (Eastman, 2008). Based on that, different standards have been created for the AECO industry (e.g. IFC) (Howard & Björk, 2008).

As a sub-domain within the AECO industry, architectural programming needs to be connected to the standards developed for the industry. Without this connection the standards of the AECO industry are missing one of the fundamental parts involved in any AECO project.

To provide an example, the absence of architectural programming information on the property inventory records for any organization reduces the reliability of such information. In April 2008, The Government Accountability Office (GAO) reported that the Army and the Navy had not verified the accuracy of about 39 and 59 percent of their real estate property inventory records, respectively (GAO, 2008). Such inaccuracy makes budget allocation for the newly proposed infrastructure very difficult since there is no way to evaluate the capacity of the existing infrastructures.

American Institute of Architects (AIA) defines a POR as

> *"An organized collection of the specific information about the client's requirements which the architect needs in order to design a particular facility. This encompasses not only the expressed requirements of the client, but all of the human, physical and external factors which will influence the design. A program is communication. It transmits and interprets the needs of the client to the designer." (Palmer, 1981).*

There are different interpretations of this definition in the AECO industry. Some architectural firms tend to focus on quantitative factors while others may include qualitative information in a POR as well (Hershberger, 1999). Academics have also proposed different methods in preparing a POR. A POR, either in a format of an informal conversation, or as a well-composed document included in the design contract, is the departure point in every design project (Hershberger, 1999). Yet existing BIM modeling standards lack the necessary fields to represent PORs. Therefore, architectural programming information is not included in BIM databases resulting in no conversation about the architectural programming requirements by other stakeholders such as energy analyzers and cost estimators through the BIM database. Lost also is the opportunity to check the design automatically against the POR.

In a conventional practice, members of the design team spend a noticeable amount of time to summarize the programming requirements and incorporate them into the design. Designers generate solutions by taking into account all the facts and limitations, including the programming requirements (Palmer, 1981). Because programming requirements are not integrated into the design environment (i.e. a BIM database) finding and extracting required information from the program is an expensive and time-consuming act (Erhan, 2003).

When the design is sufficiently developed, effort is made to collect the information from the model and check it back against the POR. This time consuming process slows down the design and leads to results with a high likelihood of human error (Erhan, 2003). The value of the programming information is recognized by some researchers. However, the requirements rarely become an integrated part of the design model and in most cases are eliminated (Clayton, Johnson, Song, & Al-Qawasmi, 1998). This problem has been recognized by the industry and efforts have begun to search for a solution.

1.2. OVERVIEW OF METHOD AND SCOPE

The goal of this study was to improve the current state of the connection between architectural programming and the AECO industry through the use of BIM technology. After analysis of several widely accepted methods for creating a POR, I produced various data models to express each method. I then synthesized these various data models into a single unified model (UFPOR). A database prototype was implemented to test the capabilities of UFPOR in modeling a POR excerpt from the industry. UFPOR was further adjusted to match the current standards of BIM technology (i.e. IFC). The result was UFPOR-IFC, a limited version of UFPOR that the current state of IFC can support. UFPOR-IFC is a blueprint for partially representing architectural programming content using IFC. I implemented a software prototype to demonstrate the capabilities of UFPOR-IFC in supporting computer applications. UFPOR-IFC was further analyzed by modeling two PORs from the industry. The research produced a generic solution that is not based on a specific approach to POR or a specific workflow in the industry. Modifications and additions are proposed as the outcomes of the research to improve the existing tools and standards in BIM technology.

It is worth noting that improving the theories and methodologies for creating a POR was out of the scope of this research. This research builds on a selected collection of some of the most popular theories on the content of a POR and explores the capabilities of the current state of BIM technology to embed the architectural programming information.

Proposing new software is also not within the scope of this study. The results of this research provide a blueprint for modeling architectural programming information in IFC. BIM computer applications that support architectural programming information can implement a subset of the proposed blueprint according to their needs.

Automating the process of auditing and evaluating the design against the architectural programming criteria is also out of the scope of this research. The research suggests a structure under which the programming information can be included in a BIM database. Many may consider that a prerequisite to automating design evaluation within a BIM process; however any work on design evaluation automation is outside of the scope of this research.

This research does not provide any suggestions on improving the usability of architectural programming information throughout the design process. Adding architectural programming information to a BIM model is the first step in having access to such information, however further research should undergo to explore how the information can be utilized most effectively.

1.3. CHALLENGES

The current state of architectural programming in AECO industry and BIM technology create certain challenges in regard to this study.

1.3.1. DIVERSITY OF POR FORMATS

Architectural companies may use different POR formats for different projects based on the type and the size. A hospital needs a more comprehensive set of requirements than a summer house. For a summer house the whole program may be reduced to the total project area set by the client. A POR for a hospital may have different levels of space packaging, intense adjacency and area requirements, as well as specific layouts for some spaces.

Architectural companies may have custom POR templates for different building types. A universal format for architectural programming should be able to easily accommodate the requirement sets for the projects ranging from a summer house to a hospital.

Different projects with different size and functions are hard to save in an identical structure. That creates a challenge for the task of creating a UFPOR.

1.3.2. DIVERSITY OF THE REQUIREMENTS

Architectural programming requirements can vary in scale, assessment methods, and complexity. They can range from purely quantitative to purely qualitative. The requirements can be imposed on the whole project or a small building element. Different domains within the AECO industry can impose requirements to ensure certain qualities within their respective domains.

The ever-evolving list of architectural programming requirement types makes proposing a schema or a data format that can accommodate all those requirement types a challenging task.

1.3.3. ADAPTATION BY THE INDUSTRY

Architectural companies typically have their uniquely adapted version of POR. Many companies have already developed tools such as Excel templates or in some cases MS Access databases to record their architectural programs. Using these templates enables them to recycle the requirements in different projects.

A UFPOR will not survive without the support and contribution from the major architectural design companies. Checking the UFPOR against the POR formats that are used by some of the major architectural firms will add to the credibility of the UFPOR.

While it is outside the scope of this research, peer-to-peer convertors can be a valuable option. Peer-to-peer convertors create interoperability between two computer applications by allowing one application to read the content created by the other application. Through peer-to-peer convertors, the companies can import their digital PORs (if they exist) to the UFPOR format, enabling reuse of the requirements and PORs.

1.4. RESEARCH METHOD

This section clarifies how the means of classic research methodology such as questions and hypothesis are used to define the problems the study tries to resolve and how it will resolve them.

1.4.1. RESEARCH QUESTIONS

The study attempts to provide answers for three main questions.

- Can a data model be devised that supports the requirements defined in major POR formats?
    - What are the common fields between all the POR formats?
    - What are the specific fields for each POR format?
    - Can a single data model support multiple POR formats?
- How can UFPOR exchange data with BIM modeling standards?
    - Should new data fields be introduced to the BIM data standards?
- Can a POR be stored in BIM modeling standards and be accessible for retrieval at later stages of the building life cycle?

1.4.2. HYPOTHESIS

This research effort has been organized around proving a few key hypotheses:

- A collection of base classes can be developed that can support the preponderance of information items in different PORs.
- Existing BIM standards can be enhanced to host the data stored in UFPOR throughout the lifecycle of a project.

Evidence to support these hypotheses has been gathered largely through logical argumentations and construction and testing of software models.

1.4.3. METHODOLOGY

Logical argumentation attempts to make sense of some aspect of the cosmos in a systematically rational manner (Groat & Wang, 2002). Academics and practitioners within the AECO industry have defined the scope and content of architectural programming. They use logical argumentation to demonstrate why architectural programming is a necessary activity and what its elements are. Defining the scope of architectural programming does not involve any equations, computer programs or mathematical formula; therefore, it fits more toward the Cultural/Discursive end of the spectrum of logical argumentation. Figure 1 shows the logical argumentation spectrum.

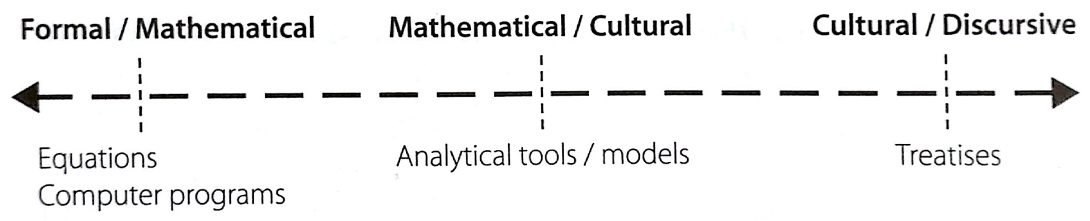


Figure 1, Logical Argumentation Spectrum. Reprinted from (Groat & Wang, 2002).

The standards for product modeling and Building Information Modeling (BIM) are also based on logical argumentation. They start from the right side of the spectrum (Cultural/Discursive) to define their scope and move toward the left side (Formal/Mathematical) as they propose machine-readable formats to store the information. Logical argumentation can set the basis for other kinds of research, such as correlational and statistical studies, model-based research, and experimental research.

Logical argumentation is the backbone of this study. The goal is to enhance the existing trends in architectural programming through proposing a model as a superset of several well-accepted models (UFPOR). The first step is to use logical reasoning to find similar data fields with different names in different data models. The following step is to enhance the existing machine-readable BIM standards by proposing additional fields or modifications so that they can host architectural programming information. From that standpoint this research is a secondary logical argumentation to the primary logical argumentation that has defined the BIM standard.

The discursive logical argumentation is developed to the Formal/Mathematical level by developing software prototype that demonstrates how the system can be implemented. The prototypes demonstrate how specific aspects of architectural programming can be translated to the proposed data model (UFPOR) and defined and accessed through a computer application.

I also used several test cases to investigate the capabilities of the proposed data models in modeling specific cases from the industry. The prototypes were defined and developed with the purpose of serving the selected case studies.

1.5. THE ORGANIZATION OF THE DISSERTATION

Chapter 2 will review the literature focus on BIM, data modeling, and predesign methods. Chapter 3 is a discussion of commercial software tools for predesign and an analysis of their strengths and weaknesses. These two chapters lead to a point of departure for my research. Chapter 4 is an analysis of leading methods for developing a POR and how each may be expressed as a data model. Chapter 5 explains how these various ontologies for the POR and data models may be unified into a single data model. Chapter 6 examines the capabilities of UFPOR in supporting PORs from the industry by developing a software prototype based on a POR excerpt. Chapter 7 describes how the model developed in section five can be adapted into IFC. Chapter 8 describes the implementation of the data model from chapter 7 as a software tool for supporting predesign. Chapter 9 describes the results of testing the software tool by putting data from POR's developed by industry into the software tool. Chapter 10 presents the conclusions from the test, the implications of the research, the limitations of the research, and suggests future work.

2. LITERATURE REVIEW

2.1. ARCHITECTURAL PROGRAMMING METHODS

Over the past 50 years, academics and practitioners have documented several processes for creating a POR. Amongst them are *Problem Seeking* (Peña, 1977), *Methods of Architectural Programming* (Sanoff, 1977)*, Information Management for Design* (Duerk, 1993), *Programming for Design : From Theory to Practice* (Cherry, 1999) and *Architectural Programming and Predesign Manager* (Hershberger, 1999). Each of these methods has strengths and weaknesses. Design firms usually adopt one of these formats and modify them over time to fit their practice.

PORs have always been an essential part of the architectural design processes. There are books and methodologies on how to collect the data that should be included in PORs (Preiser, 1975). There are also different opinions on the relationship between architectural programming and design. Different processes have been devised in which the POR can be collected, analyzed, and presented to the designer. The final outcome usually includes the data collection phase, the methods and processes of analyzing data, and the large-scale goals and objectives that the client has envisioned for the project. The detailed requirements that the designer should observe throughout the design are usually the last section of the programming document.

Historically, the formats of the programming documents are very different. They are also very different in the number of requirements that they would impose on a project. In some projects the POR is only a couple of pages and in some it is more than a hundred. The architectural offices usually adopt a version and tailor it to fit their practice. This format depends on the building type, size of the project, and the clients' requirements (Cherry, 1999).

A brief review of the existing POR models reveals the existing POR formats have different structures. Figure 2, Figure 3, and Figure 4 show the structure of three different POR formats.

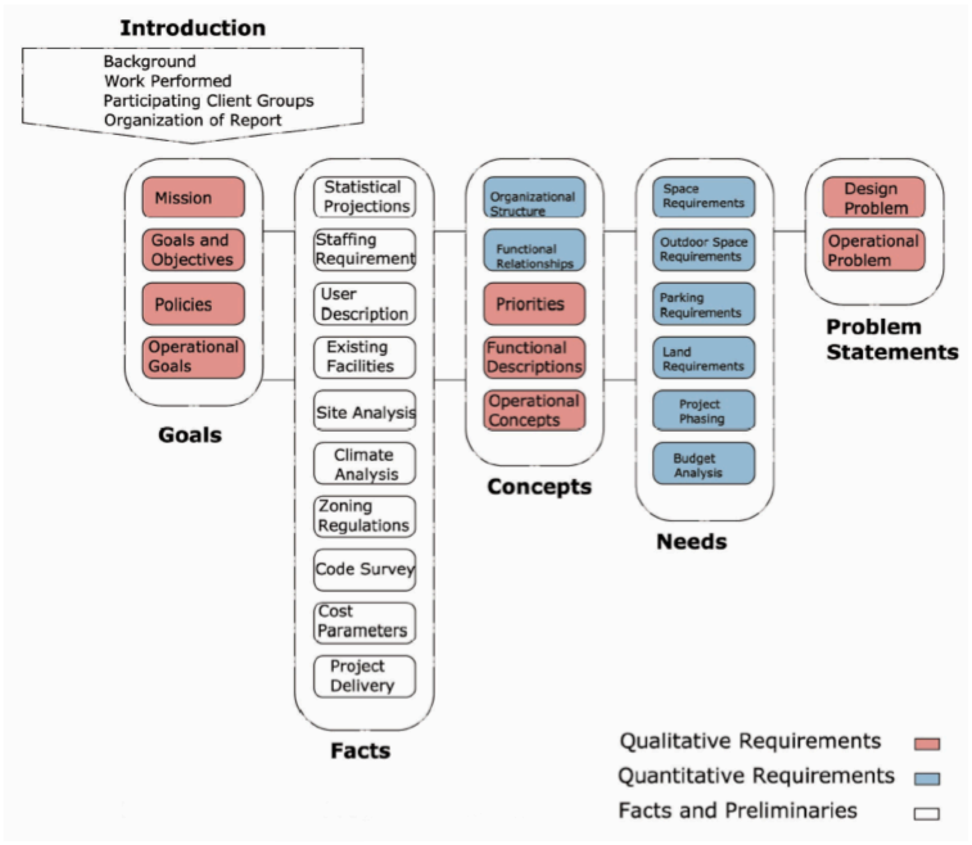


Figure 2, Peña's POR Diagram

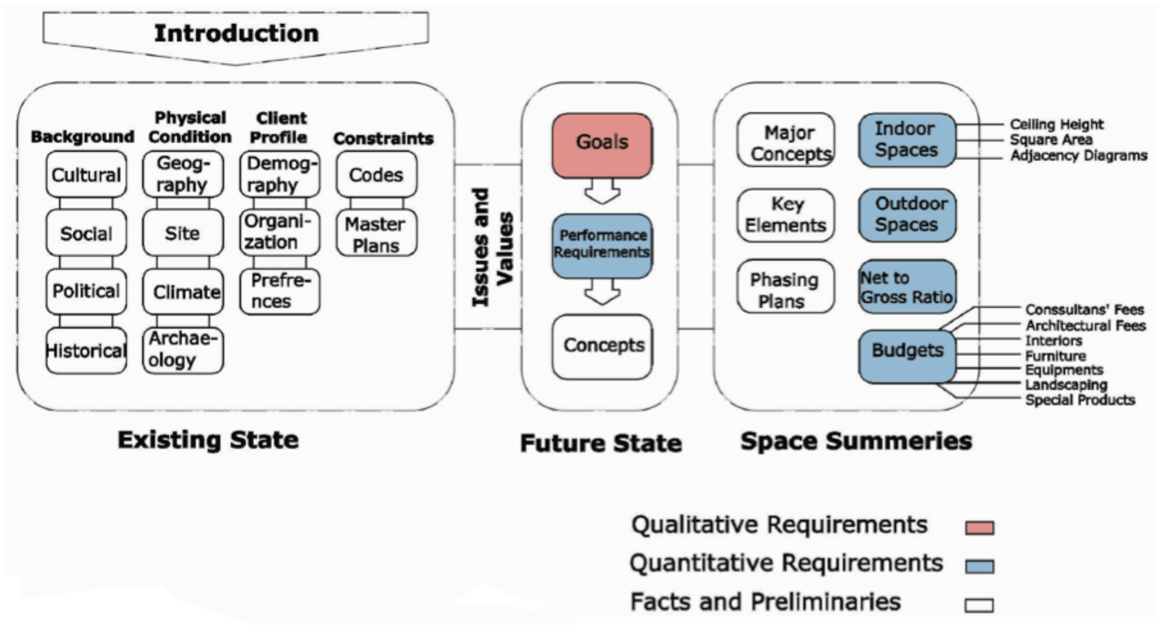


Figure 3, Duerk's POR Diagram

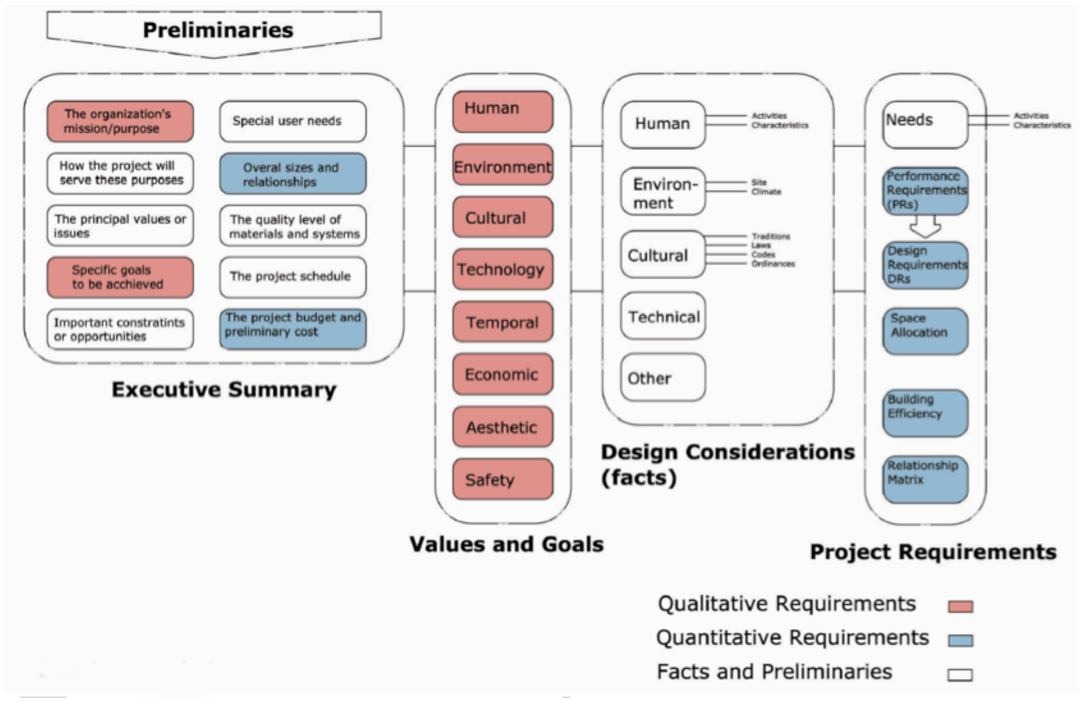


Figure 4, Hershberger's POR Diagram

All three models follow a general pattern of deriving detailed requirements from the gathered data, but there are some differences between all these models. To unify all these models in one model requires a precise analysis and close examination of all the information categories. Many of these categories, although referred to with different names, are pointing to the same concepts.

Chapter 4 is an in depth analyses of three well-known theories on architectural programming and how they can be modeled with the same schema.

2.2. STANDARDS FOR BIM DATABASES

Sharing information is a key concept in BIM technology. BIM creates a database for each project that is shared by all the design stakeholders (Smith, 2009). All the participants deposit their information into this database and are informed on all the changes imposed by the other players. Yet each participant may use different tools and internal data models to accomplish a task. For example, the energy consultant will not use the same tools for energy analysis that the MEP engineer would use to design the plumbing system. Nevertheless, some of the information is used by multiple members of the team.

Interoperability is the key concept in connecting all these design and analytical tools (Succar, 2009). Interoperability is gained when all the players use the same terminology throughout the process (Roddis, Matamoros, & Graham, 2006). For example, when the architect calls the moving part of an opening system in a wall a "Door", it may have a different meaning to the cost estimator who would call all the moving and stable parts of the opening system a "Door." Without having a dictionary that defines what a door is, the architect and cost estimator cannot have a productive conversation.

To support interoperability, several modeling standards have been developed in the AECO industry. Industry Foundation Classes (IFC), is one of the official standards (ISO 16739:2013) in the AECO industry.

Standards usually define processes to incorporate the efforts in improving and enhancing their capabilities. By following such processes, individuals and authorities can add new capabilities to these standards. The flexibility of a data model is not determined just by the owners; the data model itself should have a flexible structure as well (Björk & Laakso, 2010).

Björk (Björk & Laakso, 2010) classifies AECO modeling standards into three categories: de facto, official, and industry. This classification is based on the development process and acceptance in the industry. De facto standards gain their authority by dominating the market and attracting the end users. Official standards are approved by a standard institute (i.e. ISO standards). Industry standards are the ones that are backed by industry consortiums. The scope of this study is limited to the official standard of the AECO industry that has been adopted by ISO and ANSI (i.e. IFC).

IFC is a data exchange standard developed by buildingSMART for BIM. The object based file format provided by IFC is used by software vendors to create compatibility between computer applications in AECO industry. The IFC model specification is open and available (Liebich, 2010). It is registered by ISO as ISO/PAS 16739 and is currently in the process of becoming the official International Standard ISO

16739. IFC represents both industry consortium category and official standard. While identifying the de facto standard is more difficult and subjective, many consider IFC as the de facto data exchange standard within the AECO industry as well (ISO/IEC, 2013).

In section six and seven of this chapter, I have reviewed IFC as the official data exchange standard of the AECO industry in details.

Although BIM is a relatively new concept in the AECO industry, both architectural programming and design computation have been around for a long time and the relationship between them has been studied before. Researchers have tried to add PORs to design computation systems since the emergence of computation in architecture. For example in 1970s, OXSYS, a CAD system to support hospital design was developed by ARC (Applied Research of Cambridge) in Cambridge, UK. (Richens, 1978). OXSYS was based on a predefined set of building components that included semantic information besides the geometric information. OXSYX was capable of recording requirements such as adjacencies and had capabilities in representing a hierarchical structure for building spaces and space zones (Eastman, 1999).

2.3. CONVERTING DOMAIN KNOWLEDGE INTO SOFTWARE

Proposing software and data exchange standards for an established industry or its sub-domains requires domain knowledge conversion. This section reviews the principle that should be followed and the pitfalls that should be avoided for a successful domain knowledge conversion.

2.3.1. SEMANTIC INTEROPERABILITY AND DATA INTEGRATION

One of the main challenges in this study was to create convergence between the different POR models that were analyzed. During analyzing the models and comparing them with each other, I noticed similar concepts that are referred to by different names. In information technology, this problem is a common obstacle in dealing with data from multiple sources. Semantic interoperability is a branch of information technology that studies the problem and proposes ways to overcome it.

In Information Technology, data integration is the process of combining different data sources in order to answer a query posed by a user (Gruber, 2009). In contrast with the warehouse method that requires a copy of data stored in local databases, data integration method is based on interaction with the distributed data sources in real time. (Pontieri, Ursino, & Zumpano, 2003). While this method will provide the users with fresh and updated data, it may be slower to access the data comparing to the warehouse method.

The idea of data integration was first introduced in the 1990s. The emergence of the Internet showcased the significance of this technology. The common example to demonstrate the usage of data integration involves a user trying to find movies directed by "Woody Allen" that are in theatres close to a certain location. In order to answer this query, the system needs to query a couple of different data sources on. It needs to query a movie database to find all the movies directed by Woody Allen, and another query should be posted to another data source to find where those movies are on screen. The results from these two queries should be combined to provide the user with the answer to their query. A data warehouse with a local copy of the queried databases will not be useful in this case since having access to the most updated data is crucial.

In order to implement a data integration system, access to the semantic of the data is crucial (Lenzerini, 2002). Without having semantic knowledge about the data source, accurate data is impossible to acquire. On top of that, there are new data sources emerging on the Internet every day and if they want to last, they need to efficiently communicate with the data integration systems that are already in use. The premise of semantic interoperability is to guarantee that the meaning of the data (their semantic) is provided to the third party users. (Ouksel & Sheth, 1999). This process involves an integrated transmission of the meaning with the data. In order to accomplish this task, each data scheme should include a package that explains all the data fields and their purpose. In other words, it has some data about the data.

The idea of data integration has a long thread in the AECO industry. By nature, the AECO industry has numerous stakeholders. Each branch of the AECO industry has an independent domain, which is formed around the unique expertise that the respective domain can bring into the industry. In spite of the independence of all these sub-domains, they all contribute information to the lifecycle of a facility. They consume information that is provided by other domains and produce information that will be consumed by another domain. This is an iterative process that historically has been expensive (Cuff, 1991). Below is an example to illustrate the importance of the issue.

Imagine a scenario where an architect is hired by a school district to design a new school. While the architectural firm is in charge of providing the design for the school, the architect has to work with other consultants, for example a structural engineer, to properly include structural and mechanical elements in the design. The structural engineer needs to have access to the design information in order to add the structural elements to the design. On the other hand, the architect may modify the design in response to the changes imposed by the structural engineer. Traditionally the designer would share the design drawings as CAD files which include the geometry of the design and some tags that explain the spaces and other elements. The engineers should explore the drawings and spend some time to understand them. They hand pick the pieces that matter to them and add the structural elements to the drawings and send it back to the architect. The architect then will update her drawings by manually adding the structural elements to the design.

This imaginary scenario is not always as smooth as described above. The architect may disagree with some of the changes made by the structural engineer, which may turn this process to an iterative cycle. During this process a lot of time and energy is wasted due to the absence of an efficient data sharing system.

By applying the principles of data integration and semantic interoperability the AECO industry can be much more efficient. If there was a shared terminology the structural engineer could query the data source that the architect used to store the design information and directly access the right information in real time.

2.3.2. ONTOLOGY

Ontology is a key concept in semantic interoperability. The idea behind ontology is that within a community (domain) the same concepts and objects should be referred to with the same name. Thomas Gruber defines ontology as "Formal, explicit specification of a shared conceptualization" (Gruber, 2009). An ontology is a dictionary of all the concepts within a domain along with the relationship between the concepts.

Without having an ontology within a domain, data integration may become an impossible task. (Arvidsson & Flycht-Eriksson, 2008). Ontologies are very important parts of a data integration system. A new data scheme within a domain with an established ontology should follow the existing definitions and relationships defined by the ontology. If the ontology is missing some of the required fields, it is important to investigate the existing applications and if possible use the terminology that is widely accepted among the users.

2.3.3. AECO ONTOLOGIES

The efforts on creating a comprehensive data standard for the AECO industry date back to the 1970s (Björk & Laakso, 2010). IGES (Initial Graphics Exchange Specification) was developed in the 1970s based on the early stages of CAD technology and it was first released in 1980 by the American National Standards Institute (ANSI) (Smith & Wellington, 1986) . Another standard was DWG (and the text-based version, DXF) developed for the modeling software AutoCAD. DXF was developed for the other CAD vendors for importing and exporting CAD data (Open Design Alliance, 2013).

These standards, although they facilitated data exchange between different software vendors, had little to do with the ontology of the AECO industry. The CAD industry was not concerned with the components that all together form a building. It was mainly concerned with the shapes and geometries. The lack of a standard ontology for concepts for the building industry became critical when people began exploring the use of simulation software and interoperability with CAD. Although the advantages of semantically rich 3D models for the AECO industry had been well-documented, such models never became predominant until software vendors in the AECO industry switched from the conventional CAD practice to Building Information Modeling (BIM) in 1990s (Eastman & Siabiris, 1995). Unlike CAD that is helpful in documenting the geometry, BIM can be used in documenting objects and their properties in the context of the AECO industry. The widespread usage of BIM has created a new context for different parties involved in the AECO industry to share data. Different software vendors created an internal data schema that was interoperable among different applications in the package proposed by the vendor; soon after the lack of a common ontology became evident in the industry (Howard & Björk, 2008). While the implementation of BIM did not happen until 1990s, the idea gained momentum in 1980s and the efforts on creating the ontology had already started. ISO STEP standardization project started in the 1980s (Pratt, 2001). STEP stands for Standard for Exchange of Product Data. There was a team within STEP implementation group who were in charge of the building construction industry.

In 1988, Wim Gielingh, chairman of AECO committee of ISO/STEP, proposed a data standard under the name of GARM (General AECO Reference Model). The goal of GARM was to facilitate data-exchange amongst the computer applications in the AECO industry (Gielingh, 1988). GARM is an extension of STEP, the Standard for Exchange of Product Model data. Gielingh adapted STEP for the AECO industry by adding six extra models: Building System Model, Ship Structure Model, Plant Design Model, Distribution System Model, and General AECO Reference Model.

In the 1990s due to the momentum in the AECO industry to create and develop a common data model, twelve US companies initiated a cooperation to make their software applications compatible. These companies eventually formed the International Alliance for Interoperability (IAI) and developed a data standard under the name of IFC (Industry Foundation Classes) (Björk & Laakso, 2010; Howard & Björk, 2008). IFC inherited the efforts done through STEP and adopted most of the terminology and methodologies including the EXPRESS language. IFC is an open source effort that benefits from the contribution of several contributors in the industry and academia (Bazjanvac, 2002).

The majority of BIM design tools exchange data via the IFC format through export/import features. The IFC format can be potentially used as a wrapper for different databases in the AECO industry although some believe there are still several parts missing from the IFC ontology, which makes it less effective (Coates et al., 2010).

2.3.4. ONTOLOGY IMPLICATION IN THIS STUDY

Interoperability is one of the key premises and differentiators of this study. In order to provide seamless interoperability with other players in the AECO industry, there are two main challenges.

First, there is a lack of the ontology in architectural programming. Although architectural programming represents one of the classic tasks in the AECO industry, there has not been enough effort in creating a comprehensive ontology for this field.

Most of the effort has been focused on the content of a POR and the methods in data gathering. There are different methods in preparing a POR and usually every architectural firm has its own definition of POR. One of the main reasons for the absence of such standard is the absence of a super data model similar to IFC that the POR data model can communicate with. Now after emergence of the IFC data model it looks like a good time to create the POR ontology. Providing a uniform model for architectural programming will be one of the main outcomes of this study.

Second, there is a lack of the respective fields in the IFC data model. Having a POR ontology independent from the IFC data model will not solve the problem completely. In order to create communication options for the data integration systems, the relationship between the fields in the POR ontology and the IFC data fields must be defined.

Previous studies have shown that IFC lacks some of the fields that are required in the early stages of design (Lee, Chin, & Kim, 2003). POR is the departure point in any design effort, but much of the data provided through a POR may not be accommodated in the IFC model.

Although the current version of the AECO ontology covers a large portion of the terminology used in the industry, there are many parts that are not completely covered yet. Architectural programming is one of them.

This research includes development of an ontology based on several well-known and accepted POR formats offered by acknowledged researchers and practitioners. The UFPOR derives from the developed ontology. Developing an ontology is a critical early step toward achieving value for the industry once it is approved and adopted by an official consortium and ultimately adopted by the industry itself.

2.3.5. DIFFERENT DATA TYPES IN A POR

A POR imposes qualitative and quantitative requirements over a project (Palmer, 1981). Recording the quantitative requirements is straightforward since they can be reflected through numeric data fields, but it is more complicated for the qualitative requirements. The qualitative requirements can be recorded in the UFPOR as text type requirements or they may be converted to quantities. Quantifying the qualities has a long thread in architectural and urban design. This procedure is commonly used among city planners and urban designers (Ewing, Handy, Brownson, Clemente, & Winston, 2006).

Qualitative factors play a very important role in all the well-known POR formats. Fields such as mission, goals, objectives, policies, and concepts are some examples of the fields that can be potentially occupied by highly qualitative statements in any POR (Preiser, 1993).

Although the qualities can be represented using numbers, in some instances the level of abstraction that is involved in quantification may be unacceptable to the architectural programmer. In these cases the architectural programmer may decide to impose the requirements as qualities. This strategy, although it may involve more computationally expensive operations, is totally acceptable due to the complexities involved in the architectural design processes (Lawson, 1997). A programmer may consider “coziness” a quality that cannot be described mathematically, yet he or she may want to impose it as a requirement on certain spaces.

Some of the requirements describe a relationship between the components of a building, which although not purely qualitative, cannot be defined through simple data fields either. These requirements can be modeled through simple formulas or complicated algorithms and measured with the help of software (Kroenke & Auer, 2007). For example in urban design, enclosure as a qualitative attribute for a street is commonly measured as a function of height to width ratio (Ewing & Handy, 2009).

2.4. ARCHITECTURAL PROGRAMMING AND BIM

Having access to architectural programming information is critical to the operations of several parties within the AECO community (Succar, 2009). During the design process, architects and planners constantly evaluate the design against the programming criteria. After the construction, facility managers need access to the architectural programming information for a full understanding of the facility (Hershberger, 1999). A better integration of programming information into BIM improves interoperability amongst the parties within the AECO industry. While there is a consensus within the industry on what BIM is and what it should do, there are multiple implementations of BIM. Some vendors have branded implementations of BIM that rely on the vendor’s data models, file formats, and graphical user interfaces. Autodesk Revit and Graphisoft ArchiCAD are two examples of BIM implementations (Duell, Hathorn, & Hathorn, 2014; MacKenzie, 2015). On the other hand, BIM open standards provide a neutral solution for data exchange amongst such applications. IFC has become the standard way to share data among BIM applications (Vanlande, Nicolle, & Cruz, 2008). A non-profit committee creates IFC definitions through an open process. The definitions are openly published and freely distributed. IFC is regarded by the International Standards Organization (ISO) as the standard for data sharing in the construction and facility management industry (Liebich, 2010).

The AECO industry is diverse and the intended uses of BIM are pervasive therefore, any BIM implementation is continually evolving. Proprietary commercial products are being updated regularly with new releases. IFC includes a flexible way to define and utilize data standards to enable incremental growth (Liebich & Wix, 1999). The most common classes and their attributes are part of the core IFC, but data models and custom property sets can be defined to expand the core model when needed.

There have been several attempts to make architectural programming information accessible within the BIM context. While every attempt to solve this problem is different in terms of their approach, they can be divided in two main categories: bridging between specific proprietary formats, and integrating into open standard formats.

2.4.1. PROPRIETORY FORMATS AND SOLUTIONS

Some of the attempts to incorporate architectural programming information into the design process rely on creating a bridge between a specific proprietary BIM application and a specific proprietary tool for architectural programming. For example, Trelligence Affinity provides a plugin for Revit architecture that makes the Trelligence format for architectural programming available in Revit Architecture environment (About Trelligence, 2015). HKS Architects, one of the largest healthcare designers in North America, uses a custom Revit Architecture plugin that connects their Revit models to their format for architectural programming (Barekati, 2012).

2.4.2. OPEN STANDARDS

IFC promotes creation of sub-standards within IFC by adding custom property sets that are geared toward the needs of a specific subset in AECO industry (Liebich, 2009). This aspect of IFC has created the opportunity to define standards for AECO sub-domains based on IFC. In section 2.8, three of such standards that involve architectural programming information are discussed.

2.5. OBJECT-ORIENTED DESIGN/PROGRAMMING

Object-oriented design is an approach in software development where a system is conceptualized as a collection of interacting objects. An object is a group of related methods and variables that represent a unique entity (Bruegge & Dutoit, 2004). Object-oriented design is the approach selected by the de-facto and official standards in the AECO industry. Representing POR structures through object-oriented systems is an important step in making them interoperable with the existing BIM standards. To achieve a better understanding of object-oriented design, the following subsections present some of the fundamental concepts in object-oriented Design.

2.5.1. CLASSES AND INSTANCES

Classes and instances are the main building blocks of any object-oriented system. They are best explained through real world examples. Imagine a scenario where one needs to create a software system that represents all the cars in a parking lot. All those cars can move and carry people (capabilities). They also all have body colors and build years (attributes). The concept of vehicle is defined based on all the common attributes and capabilities. The vehicle, as a concept, creates a blueprint for all the cars in the parking garage to follow. By following such a blueprint all the cars share common attributes and capabilities. In object-oriented Programming, the concept of an automobile is an example of a class. In contrast to a class, which collects the descriptors as variable attributes, an instance provides the values to the class attributes. In this example, Vehicle is a class and all the cars in that parking garage are instances of the type vehicle.

2.5.2. ATTRIBUTES AND OPERATIONS

Each class has some attributes (variables) and provides some operations (methods). The class attributes store the information that the class holds directly or depends on to perform its operations. Class operations represent the capabilities that the class offers (Page-Jones, 2000). A simple form of class operation is providing access to the class attributes to read or change the values. In the vehicle example from the previous section, "weight" and "color" can be examples for class attributes and "get color" and "accelerate" can be examples for class operations.

2.5.3. INHERITANCE

Inheritance promotes reusability by allowing new classes to extend an existing class by adding new attributes while retaining all the attributes and capabilities of the parent class. In the parking lot example, while all the vehicles in the parking garage share capabilities and attributes, they can be further categorized into smaller groups where they share further attributes and capabilities. For example, one can have a "Bus" category, a “Motorcycle” category and a “Sedan” category. One can use the concept of inheritance and create three new classes (Bus, Motorcycle, and Sedan) where they all inherit from the Vehicle class. That means they inherit and share the capabilities and attributes on the Vehicle level but they add their unique capabilities and attributes as well.

Various classes may be organized into an inheritance hierarchy that shows base classes with the most common methods and attributes and specialized classes that add attributes and methods.

2.5.4. INFORMATION HIDING

Object-oriented systems provide mechanisms to manage the visibility to the internal information and implementation of classes and objects. This characteristic is sometimes referred to as encapsulation and prevents the system from undesirable changes caused by unauthorized parties. In the parking lot example, in modeling a car an important attribute is acceleration, and use of the model of the car may require change to the acceleration. This can be accomplished by exposing a capability (a class method) called “accelerate”. While the consumer of the class can use this capability in developing a simulation of driving a car, the internal implementation of the method is hidden. The user of the class may also query about the physical attributes of the car such as weight, but those attributes can be kept read-only to prevent the driver from manipulating them.

2.5.5. POLYMORPHISM

Polymorphism allows one object to take different roles. Polymorphism can be accomplished through several mechanisms. For example an instance of “Sedan” can also be used as an instance of “Vehicle” if needed. It can also expose several other external interfaces besides Sedan, and Vehicle such as “Private Car” or “Electric Car”. This capability allows the object-oriented systems to stay versatile and flexible (Stroustrup, 2014).

2.6. UML DIAGRAMS

The Object-Oriented models for a POR are presented through simplified UML class diagrams. UML is a widely accepted and ISO endorsed method for analyzing and documenting processes so that they can be implemented as software (Bruegge & Dutoit, 2004). Use of a simplified version of the UML diagrams is intended to increase readability.

2.7. WHAT IS UML

UML (unified modeling language) is a set of standard graphical languages to represent an object-oriented system. Through UML diagrams an object-oriented system can be drafted graphically with precise representation of system architecture, classes, relationships and other aspects (D'souza & Wills, 1998).

UML defines nine types of diagrams: class (package), object, use case, sequence, collaboration, statechart, activity, component, and deployment. Each of these diagrams represents a unique aspect of software architecture during its lifecycle. Below I describe some of the diagrams that are used in understanding the behavior of a software system in relation to the final users and the industrial domain that it belongs to.

- *Class*. Class diagrams are the backbone of almost every object-oriented development process, including UML. They describe the static structure of a system.
- *Package diagram*. Classes will be grouped into several packages that implement a well-defined set of functionality. A package diagram will show which class belongs to what package and how these packages would communicate.
- *Object*. While class diagrams and package diagrams show the static condition of a system, object diagrams represent the condition of the system in a given situation based on an instance of a class. They are usually used to test the functionality of a system under certain circumstances. They include instances as well as classes.
- *Use Case*. Use case diagrams are the sequence of actions that the user of a system is expected to take in order to accomplish a task.

2.7.1. UML DIAGRAMS IN THIS DISSERTATION

In this dissertation, UML diagrams are used to illustrate the data models of the PORs, and also proposing UFPOR. They are also used as an analytical tool to clarify the logical argumentation that the study pursues. All the UML diagrams in this dissertation are class diagrams. In some cases the list of class variables are followed by three dots to show that more variables can be added (Figure 29). The class diagrams do not show any of the class methods to increase the readability of the models. Each diagrams consists of two or more classes and their relationships. Arrows and lines that connect the classes together show the relationships between the classes. Each arrow is annotated with some text that descries the relationship. The beginning and the end of each connecting line may be annotated with a number range (e.g. 0..1), the range shows how many instances of the class can be included in the relationship.

2.8. IFC FILE FORMAT

Industry Foundation Classes (IFC) was initially developed by International Alliance for Interoperability (IAI) to facilitate data exchange between different parties involved in the AECO industry (Hietanen & Final, 2006). IFC is useful in describing building and construction data in a neutral platform format. IFC is an official International Standard registered under ISO 16739:2013 (ISO/IEC, 2013).

2.8.1. INTODUCTION TO STEP AND EXPRESS

STEP (ISO 10303) is the international standard for product modeling. STEP is widely used in CAD applications to exchange 3D objects (Pratt, 2001). It is worth noting that STEP is the biggest standard within ISO, which is an indication of the complexities involved in product modeling. STEP relies on EXPRESS modeling language to define data models. IFC also relies on EXPRESS to define its data models.

EXPRESS is a standard data modeling language (Schenck & Wilson, 1994). Data modeling is a branch of information technology and computer science that defines the processes and techniques used in investigating a system and creating the data models representing that system (Hirschheim, Klein, & Lyytinen, 1995). Comparing to physical implementation of a product through a computer system, EXPRESS provides a more abstract definition. EXPRESS is conceived to support different implementations such as object-oriented systems and relational databases (Eastman, 1999). Below is a list of some of the key concepts in EXPRESS

- ENTITY, each entity describes an object including its attributes and behavior. Entities can have inheritance relationships with each other. In the example above, the Male entity is a subtype of Person entity.
- SCHEMAS, A group of interrelated entities create a schema. A schema is capable of modeling a full system through multiple entities. The entities of a schema are related to each other through inheritance, composition, association, or other types of relationships. The example above shows family as a system that can be independently described through its entities such as, father, and mother.
- BASIC TYPES, some of the major basic types in EXPRESS are INTEGER, REAL, STRING, BOOLEAN, and BINARY. All the entities are directly or indirectly (through their relationships with other entities) composed of these basic types.

EXPRESS is the language used in defining IFC schemas and entities. IFC documentation provides two EXPRESS equivalent for each entity, Inheritance Graph and EXPRESS Specification. In Express Specification only direct attributes for each entity are shown. Inherited attributes can be accessed through the provided subtype and supertype entity names. In Inheritance Graph the supertypes of each entity along with their attributes are shown directly. Code 1 shows the EXPRESS Specification and Inheritance Graph for IfcObjectDefinition.

EXPRESS Specification

```
ENTITY IfcObjectDefinition
 ABSTRACT SUPERTYPE OF(ONEOF(IfcContext, IfcObject, IfcTypeObject))
 SUBTYPE OF IfcRoot;
 INVERSE
   HasAssignments        : SET OF IfcRelAssigns FOR RelatedObjects;
   Nests                 : SET [0:1] OF IfcRelNests FOR RelatedObjects;
   IsNestedBy            : SET OF IfcRelNests FOR RelatingObject;
   HasContext            : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
   IsDecomposedBy        : SET OF IfcRelAggregates FOR RelatingObject;
   Decomposes            : SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
   HasAssociations       : SET OF IfcRelAssociates FOR RelatedObjects;
END_ENTITY;
```

Inheritance Graph

```
ENTITY IfcObjectDefinition
 ENTITY IfcRoot
   GlobalId              : IfcGloballyUniqueId;
   OwnerHistory          : OPTIONAL IfcOwnerHistory;
   Name                  : OPTIONAL IfcLabel;
   Description           : OPTIONAL IfcText;
 ENTITY IfcObjectDefinition
 INVERSE
   HasAssignments        : SET OF IfcRelAssigns FOR RelatedObjects;
   Nests                 : SET [0:1] OF IfcRelNests FOR RelatedObjects;
   IsNestedBy            : SET OF IfcRelNests FOR RelatingObject;
   HasContext            : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
   IsDecomposedBy        : SET OF IfcRelAggregates FOR RelatingObject;
   Decomposes            : SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
   HasAssociations       : SET OF IfcRelAssociates FOR RelatedObjects;
END_ENTITY;
```

Code 1, EXPRESS Specification and Inheritance (BuildingSmart, 2016n)

2.8.2. STEP PHYSICAL FORMAT

The entities and schemas defined in STEP provide the templates to describe a system. For example, to describe a family, an instance of the Mother entity and an instance of the Father entity are needed. If the family has any children, a “Child” entity should be created by extending the “Person” entity and then instantiated for every child.

STEP physical format (STEP-File) is the part of EXPRESS standard that standardizes the format in which a system can be described through the instances of entities (Peak, Lubell, Srinivasan, & Waterbury, 2004). STEP physical format is a text-based format with .stp extension that resembles mark-up languages such as XML and HTML. It is both machine and human readable (Lalmas, 2009). IFC inherits the STEP physical file as one of its standard formats. The other standard formats are .ifcXML and .ifcZIP (ISO/IEC, 2007). While both of these formats are also accepted by ISO and equally powerful as .ifc files, their usage is not as common as .ifc files. In this study only .ifc format was used to describe IFC models.

Every .ifc file is composed of a header and a body section. The header contains information about the file itself while the body documents all the instances of entities that together form the system. Each item in the body describes one instance.

```
STEPId=EntityName(AttributeValue1, AttributeValue2, …);
```

Code 2, .IFC Entity

The STEP Id is only valid in one file exchange; therefore the same entities may have different STEP Ids over multiple file exchanges. The entity name is determined by the entity specifications. The attribute values are determined by the instance and should fit the type and other criteria that are determined by the entity specifications. The attribute values are associated with attributes by their position in the list. For optional attributes, if the instance does not have any value, an asterisk represents empty value. Code 3 shows a family described in an .ifc file.

```
HEADER;
FILE_DESCRIPTION(description, implementation level);
FILE_NAME(
‘name’,
‘’time,
('The User'), ('The Company'),
'The name and version of the IFC preprocessor',
'The name of the originating software system',
'The name of the authorizing person');
FILE_SCHEMA(('IFC2X2_FINAL'));
ENDSEC;
DATA;
#1=FEMALE('Jill',*,*);
#2=MALE('Jake',*,*);
#3=FEMALE('Jannet,#1,#2);
#4=MALE('John',#1,#2);
ENDSEC;
END-ISO-10303-21;
```

Code 3, IFC File Sample

2.9. IFC SCHEMA LAYERS

There are 38 schemas that together form IFC. These schemas are divided into 4 categories. All the 38 schemas are interrelated with their entities re-appearing in other schemas. Also the higher-level schemas may build upon the lower-level schemas by extending their entities.

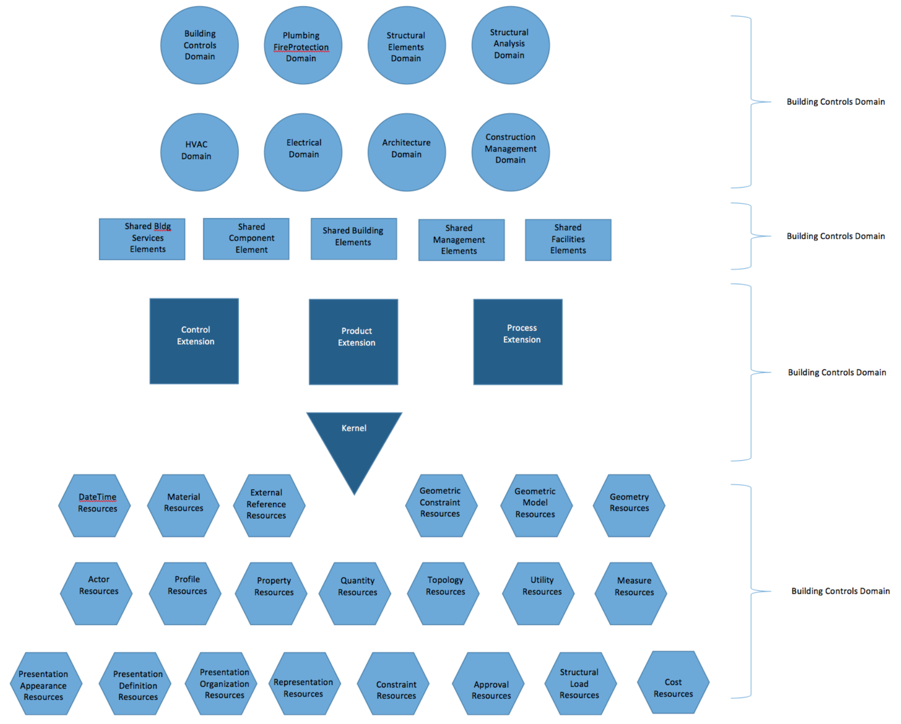


Figure 5, IFC 4 Data Schema

2.9.1. CORE SCHEMA

Core Schema provides a collection of base entities that are needed to model a system in the AECO industry. All the entities in Core Schema are either direct or indirect extensions of IfcRoot, or are collections that contain IfcRoot subclasses. IfcRoot only captures the essential attributes such as GUID (Globally Unique Identification), Name, and Description. Code 4 is the EXPRESS specification for IfcRoot.

```
ENTITY IfcRoot
 ABSTRACT SUPERTYPE OF(ONEOF(IfcObjectDefinition, IfcPropertyDefinition, IfcRelationship));
  GlobalId                 : IfcGloballyUniqueId;
  OwnerHistory             : OPTIONAL IfcOwnerHistory;
  Name                     : OPTIONAL IfcLabel;
  Description              : OPTIONAL IfcText;
 UNIQUE
  UR1                      : GlobalId;
END_ENTITY;
```

Code 4, IfcRoot (BuildingSmart, 2016ac)

IfcRoot has three direct children: IfcObjectDefinition, IfcPropertyDefinition, and IfcRelationship. The direct children of IfcRoot define a foundation for other entities and do not capture any AECO oriented information. These entities are abstract, which means they cannot be instantiated directly and need to be extended first.

Four schemas form the Core schema together. Those four schemas are IfcKernel, IfcControlExtendion, IfcProcessExtension, and IfcProductExtension.

2.9.1.1. IfcKernel

As the name suggests, IfcKernel is a generic layer that contains all the fundamental abstract entities needed to create a system. The entities in this schema are generic to the entire product modeling disciplines and are not specialized for the AECO industry. IfcRoot and its direct subclasses are some of the most important entities in IfcKernel. These entities are described below.

2.9.1.2. OBJECT DEFINITIONS

IfcObjectDefinition is the base entity for defining objects in IFC. It is an abstract entity that captures the most generic attributes of every object regardless of what the object represents. Code 5 is the EXPRESS specification for IfcObjectDefinition.

```
ENTITY IfcObjectDefinition
 ABSTRACT SUPERTYPE OF(ONEOF(IfcContext, IfcObject, IfcTypeObject))
 SUBTYPE OF IfcRoot;
 INVERSE
   HasAssignments       : SET OF IfcRelAssigns FOR RelatedObjects;
   Nests                : SET [0:1] OF IfcRelNests FOR RelatedObjects;
   IsNestedBy           : SET OF IfcRelNests FOR RelatingObject;
   HasContext           : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
   IsDecomposedBy       : SET OF IfcRelAggregates FOR RelatingObject;
   Decomposes           : SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
   HasAssociations      : SET OF IfcRelAssociates FOR RelatedObjects;
END_ENTITY;
```

Code 5, IfcObjectDefinition (BuildingSmart, 2016n)

This entity does not define any direct attributes and only declares INVERSE relationships. INVERSE attributes are not populated through the entity directly, but rather indicate that the entity can be referenced from other entities. For example an instance of IfcRelAssigns can reference an instance of IfcObjectDefinition through its STEP ID. Taking a closer look at IfcRelAssigns in Code 6 can help to understand inverse relationships:

```
ENTITY IfcRelAssigns
ABSTRACT SUPERTYPE OF (ONE OF
  (IfcRelAssignsToActor
  ,IfcRelAssignsToControl
  ,IfcRelAssignsToGroup
  ,IfcRelAssignsToProcess
  ,IfcRelAssignsToProduct
  ,IfcRelAssignsToResource))
SUBTYPE OF (IfcRelationship);
  RelatedObjects : SET [1:?] OF IfcObjectDefinition;       RelatedObjectsType :
OPTIONAL IfcObjectTypeEnum;
WHERE
  WR1 : IfcCorrectObjectAssignment(RelatedObjectsType, RelatedObjects);
END_ENTITY;
```

Code 6, IfcRelAssigns (BuildingSmart, 2016w)

IfcRelAssigns has a direct relationship with IfcObjectDefinition. This is the same relationship that is shown in IfcObjectRelationship entity as "INVERSE" relationship.

IfcObject and IfcTypeObject are two main base classes that are used in IFC to define building elements such as walls, buildings, windows, and doors. They both have similar entity hierarchies. By using IfcTypeObject, one can capture the common attributes of similar objects and avoid duplication. For example, imagine a house with four bedrooms that each has the same door type. To document those four doors through IFC, the .ifc file will have four IfcDoor instances that capture the unique attributes about each door (such as their location). There is one instance of IfcDoorType that captures all the attributes that the doors have in common. A separate entity called IfcRelDefinesByType is used to relate the objects and their types together (Code 7).

```
ENTITY IfcRelDefinesByType
  SUBTYPE OF (IfcRelDefines);
    RelatedObjects    : SET [1:?] OF IfcObject;
    RelatingType      : IfcTypeObject;
END_ENTITY;
```

Code 7, IfcRelDefinesByType (BuildingSmart, 2016z)

IfcContext defines the context in which the building elements reside. One of the subclasses of IfcContext is IfcProject. IfcProject is the root concrete element in an .ifc file. The spatial structure, building elements, and their types are all parts of IfcProject. Figure 8 shows how IfcProject connects all the elements together. Chapter 7 explains IfcProject in more detail and describes how the spatial structure can be assigned to the project.

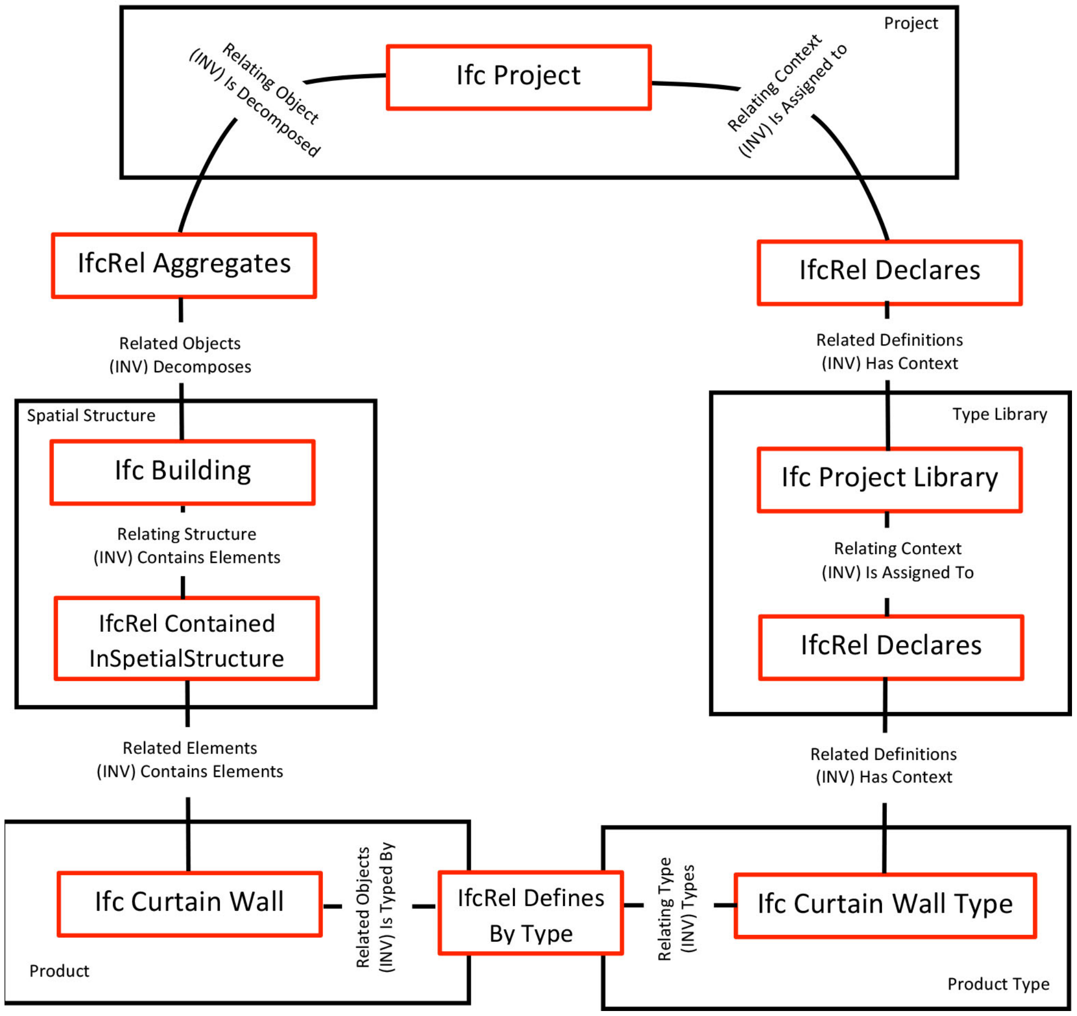


Figure 6, IfcProject. Reprinted from (BuildingSmart, 2016u).

2.9.2. RESOURCE SCHEMAS

Resource Schemas is a collection of utility schemas that are used by other schemas as their attributes. None of these entities are subclasses of IfcRoot since they do not independently represent a building element.

IfcGeometryResource is one of the schemas that belong to this group. The subclasses of this schema are used to describe the geometry of building elements.

IfcPolyline, IfcCurve, and IfcLine are some of the entities in this schema. More schemas from this group are introduced in the next chapter for their usage in UFPOR.

2.9.3. SHARED SCHEMAS

Shared schema represent more specialized entities that are not covered by the Core Schemas. Some of the entities that are used to create a POR in IFC are in Core and Resource schema. The detailed attributes that are needed to describe specific building elements such as doors and windows may create relationships to entities from the shared or domain schema such as IfcBeam and IfcDoor. Shared Schema falls into 5 different categories.

2.9.3.1. SHARED BUILDING SERVICES ELEMENTS

With schemas such as IfcHvacDomain, IfcPlumbingFireProtectionDomain, and IfcElectricalDomain, this category focuses on the building system elements. Architectural programming requirements concerning areas such as thermal comfort, lightning levels, and earthquake resistance may use this domain's entities.

2.9.3.2. SHARED COMPONENT ELEMENTS

This category focuses on detail building components such as various accessories and fasteners. PORs usually deal with requirements on a larger scale than the one covered by this category, but entities from this schema can be used in detailed requirements.

2.9.3.3. SHARED BUILDING ELEMENTS

With IfcBuildingElement being the base class for all the entities in this category, building elements such as wall, beam, column, slab, roof, stair, ramp, window, door and covering are some of the main entities in this category. PORs can impose direct or indirect requirements on walls, windows, and doors. The direct subclasses of IfcBuildingElement are IfcBeam, IfcBuildingElementProxy, IfcChimney, IfcColumn, IfcCovering, IfcCurtainWall, IfcDoor, IfcFooting, IfcMember, IfcPile, IfcPlate, IfcRailing, IfcRamp, IfcRampFlight, IfcRoof, IfcShadingDevice, IfcSlab, IfcStair, IfcStairFlight, IfcWall, IfcWindow. Code 8 shows IfcBuildingElement inheritance graph.

```
ENTITY IfcBuildingElement
 ENTITY IfcRoot
  GlobalId                 : IfcGloballyUniqueId;
  OwnerHistory             : OPTIONAL IfcOwnerHistory;
  Name                     : OPTIONAL IfcLabel;
  Description              : OPTIONAL IfcText;
 ENTITY IfcObjectDefinition
 INVERSE
  HasAssignments           : SET OF IfcRelAssigns FOR RelatedObjects;
  Nests                    : SET [0:1] OF IfcRelNests FOR RelatedObjects;
  IsNestedBy               : SET OF IfcRelNests FOR RelatingObject;
  HasContext               : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
  IsDecomposedBy           : SET OF IfcRelAggregates FOR RelatingObject;
  Decomposes               : SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
  HasAssociations          : SET OF IfcRelAssociates FOR RelatedObjects;
 ENTITY IfcObject
  ObjectType               : OPTIONAL IfcLabel;
 INVERSE
  IsDeclaredBy             : SET [0:1] OF IfcRelDefinesByObject FOR RelatedObjects;
  Declares                 : SET OF IfcRelDefinesByObject FOR RelatingObject;
  IsTypedBy                : SET [0:1] OF IfcRelDefinesByType FOR RelatedObjects;
  IsDefinedBy              : SET OF IfcRelDefinesByProperties FOR RelatedObjects;
 ENTITY IfcProduct
  ObjectPlacement          : OPTIONAL IfcObjectPlacement;
  Representation           : OPTIONAL IfcProductRepresentation;
 INVERSE
  ReferencedBy             : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
 ENTITY IfcElement
  Tag                      : OPTIONAL IfcIdentifier;
 INVERSE
  FillsVoids               : SET [0:1] OF IfcRelFillsElement FOR RelatedBuildingElement;
  ConnectedTo              : SET OF IfcRelConnectsElements FOR RelatingElement;
  IsInterferedByElements   : SET OF IfcRelInterferesElements FOR RelatedElement;
  InterferesElements       : SET OF IfcRelInterferesElements FOR RelatingElement;
  HasProjections           : SET OF IfcRelProjectsElement FOR RelatingElement;
  ReferencedInStructures   : SET OF IfcRelReferencedInSpatialStructure FOR RelatedElements;
  HasOpenings              : SET OF IfcRelVoidsElement FOR RelatingElement;
  IsConnectionRealization  : SET OF IfcRelConnectsWithRealizingElements FOR RealizingElements;
  ProvidesBoundaries       : SET OF IfcRelSpaceBoundary FOR RelatedBuildingElement;
  ConnectedFrom            : SET OF IfcRelConnectsElements FOR RelatedElement;
  ContainedInStructure     : SET [0:1] OF IfcRelContainedInSpatialStructure FOR RelatedElements;
 ENTITY IfcBuildingElement
 INVERSE
  HasCoverings             : SET OF IfcRelCoversBldgElements FOR RelatingBuildingElement;
END_ENTITY;
```

Code 8, IfcBuildingElement Inheritance Graph (BuildingSmart, 2016f)

2.9.3.4. SHARED MANAGEMENT ELEMENTS

This category holds elements that are used in construction management processes and generally do not contain architectural information. Cost schedules, purchase orders, change orders and work orders are some of the entities covered in this category.

2.9.3.5. SHARED FACILITIES ELEMENTS

While IFC considers this category a prerequisite to support the facility management domain, some of the base classes are of interest for architectural programming information. Furniture is one of the main entities covered in this category and furniture requirements are one of the common requirements in architectural programing information.

2.9.4. DOMAIN SCHEMAS

Domain Schemas contains eight schemas: Building Controls, Plumbing Fire Protection, Structural Elements, Structural Analysis, HVAC, Electrical, Architecture, Construction Management. These categories cover detail information about each domain that is not covered in the lower levels. These categories can be regarded as annexes to the main IFC schema. Entities from these schemas can be used for more detailed requirements. For example, Architecture Domain includes several property sets and type enumerations focused on doors and windows. They can be useful if a POR has requirements beyond the opening dimensions. For example, the requirements regarding a door operation type (single swing, double swing, sliding, folding, etc.) can be covered through applying constraints on the property sets covered in Architecture Domain.

2.10. IFC AND PROPERTY SETS

IFC is a large collection of entities that can represent a very large array of information in the AECO industry. Usually computer applications in the AECO industry support a predefined subset of IFC. On top of that, the simple form of IFC entities can be further enriched with custom property sets. Property sets are one of the fundamental concepts in IFC and are part of the Kernel schema. IfcPropertyDefinition entity is the base entity for properties (Hietanen & Final, 2006).

```
ENTITY IfcPropertyDefinition
 ABSTRACT SUPERTYPE OF(ONEOF(IfcPropertySetDefinition, IfcPropertyTemplateDefinition))
 SUBTYPE OF IfcRoot;
 INVERSE
  HasContext              : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
  HasAssociations         : SET OF IfcRelAssociates FOR RelatedObjects;
END_ENTITY;
```

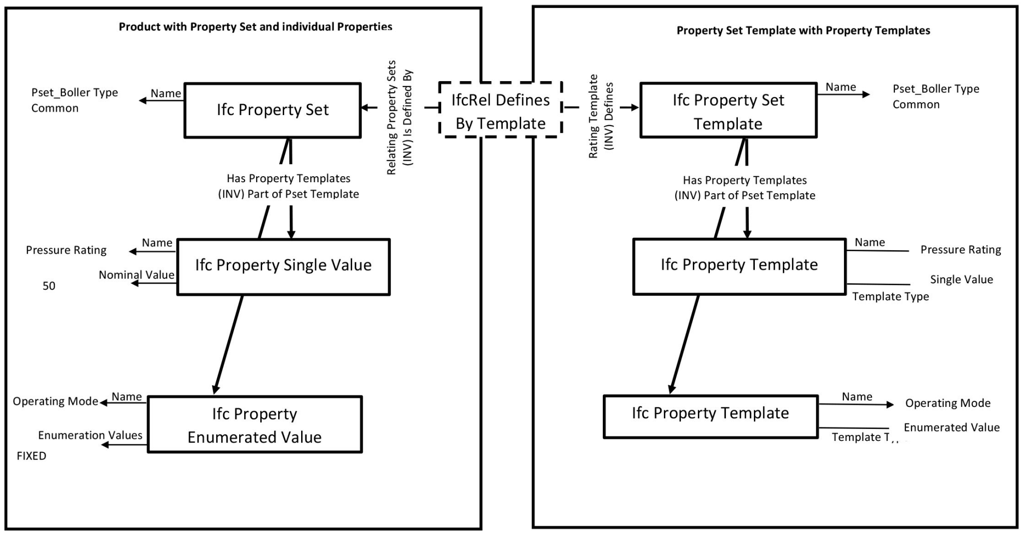
Code 9, IfcPropertyDefinition (buildingSMART, 2015a)

Defining a property consists of two steps: first, defining the template and second, defining the property. Property templates are used to further define the details on a property. Code 10 shows IfcSimplePropery Template entity as an example for IfcPropertyTemplateDefinition.

```
ENTITY IfcPropertyTemplate
  SUBTYPE OF (IfcPropertyTemplateDefinition);
    PropertyType      : IfcPropertyTemplateTypeEnum;
    PrimaryDataType   : OPTIONAL IfcIdentifier;
    SecondaryDataType   : OPTIONAL IfcIdentifier;
    PrimaryUnit       : OPTIONAL IfcUnit;
    SecondaryUnit     : OPTIONAL IfcUnit;
    AccessState       : OPTIONAL IfcStateEnum;
    HasPropertyTemplates   : OPTIONAL SET [1:?] OF IfcPropertyTemplate;
  INVERSE
    PartOfPropertyTemplate   : SET OF IfcPropertyTemplate FOR HasPropertyTemplates;
    PartOfPsetTemplate   : SET OF IfcPropertySetTemplate FOR HasPropertyTemplates;
  WHERE
    CorrectDecomposition   : ((PropertyType = IfcPropertyTemplateTypeEnum.COMPLEX)
                             AND EXISTS(HasPropertyTemplates)) XOR ((PropertyType <>
                             IfcPropertyTemplateTypeEnum.COMPLEX) AND
                             NOT(EXISTS(HasPropertyTemplates))) ;
END_ENTITY;
```

Code 10, IfcPropertyTemplate (buildingSMART, 2015c)

IfcProperty entity has several subclasses such as IfcPropertyBoundedValue, IfcPropertyEnumeratedValue, IfcPropertyListValue, IfcPropertyReferenceValue, IfcPropertySingleValue, IfcPropertyTableValue, This wide array of choices for attaching properties to IFC entities makes IFC a flexible data model that's easy to extend. Figure 9 shows how a property and a property template should be used together.



Product with Property Set and individual Properties
Pset_Boller Type Common
Name
Ifc Property Set
Relating Property Sets (INV) Is Defined By
IfcRel Defines By Template
Has Property Templates (INV) Part of Pset Template
Pressure Rating
Name
Nominal Value
50
Ifc Property Single Value
Operating Mode
Name
Enumeration Values
FIXED
Ifc Property Enumerated Value
Property Set Template with Property Templates
Rating Template (INV) Defines
Ifc Property Set Template
Name
Pset_Boller Type Common
Has Property Templates (INV) Part of Pset Template
Ifc Property Template
Name
Pressure Rating
Single Value
Template Type
Ifc Property Template
Name
Operating Mode
Enumerated Value
Template T..

Figure 7, Property Template and Property Relationship. Reprinted from (buildingSMART, 2015d).

2.10.1. DYNAMICALLY EXTENDIBLE AND STATICALLY DEFINED PROPERTIES

IFC entities can have two different types of property sets: dynamically extendable property sets and statically defined property sets (buildingSMART, 2015b). Dynamically extendable property sets are defined as "a set of properties for which the IFC model only provides a kind of 'meta model', to be further declared by agreement". This means no entity definition of the properties exists within the IFC model. The declaration is done by assigning a significant string value to the Name attribute of the entity as defined in the entity IfcPropertySet and at each subtype of IfcProperty, referenced by the property set. Dynamically defined property sets may have an underlying template provided by IfcPropertySetTemplate.

Statically defined property sets are defined as “A property set entity that exists within the IFC specification. The semantic meaning of each statically defined property set is declared by its entity type and the meaning of the properties is defined by the name and data type of the explicit attribute representing it.” (buildingSMART, 2015b).

2.10.2. DYNAMICALLY EXTENDIBLE PROPERTIES AND MVDs

Dynamically extendable property sets (Custom property sets) are in the heart of defining Model View Definitions (MVD). MVDs are one of the most important concepts in IFC. IFC specification defines a large array of schemas. Every schema is defined to support the needs of a specific subset of the AECO industry. Computer applications that rely on IFC for data exchange are also designed for the usage of a specific subset of the industry. An MVD is a subset of IFC enriched with dynamically extendable property sets that is most suitable for a subset of AECO industry (Hietanen & Final, 2006; Liebich & Wix, 1999; Weise, Katranuschkov, & Scherer, 2003). BuildingSMART.org provides a standard way of creating MVDs that is documented in the Information Delivery Manual format, IDM (also ISO29481). MVDs are documented in a format called MVDXML. BuldingSmart has published an application called "ifcDoc" that can be used to create MVDs in MVDXML format (buildingSmart, 2015b). Alternatively a text (or xml) editing application can be used to create or edit an MVD format as long as all the rules defined by MVDXML are followed.

2.11. IFC AND ARCHITECTURAL PROGRAMMING INFORMATION

There have been several initiatives to define IFC model view definitions that are capable of capturing information that is often in a POR.

2.11.1. IFC AR-5

IFC AR-5 is one of the first attempts to expand IFC capabilities to host Architectural Programing information. AR-5 was actively under development between 2002 and 2004 with the results being incorporated into IFC 2*2 (*Information requirements specification – [AR-5] early design,* 2004). The goal of AR-5 project was to extend the capabilities of the IFC schema to include early design decisions and requirements. The intent of AR-5 project was to make early design decisions and requirements available throughout the design process. The idea is to add an architectural requirement to an IFC file and later evaluate whether the requirement was met or not. The research addressed four primary areas: Analysis of Alternatives, Blocking and Stacking, Bubble Diagram, Review and Revision.

The final report on AR-5 project starts by describing Architectural Programing process and the types of information included in an Architectural Programing document. The proposed structure for Architectural Programing in AR-5 is based on the concept of activities and their relationship. Activities are defined as the facilitated actions that the users of a building perform in that building. For a school, examples of activities include learning and playing. The requirements are the result of understanding what the users need to perform the activities successfully. The list of spaces in a building is derived from the activities. AR-5 also recommends the use of space relationship matrices to describe space relationships (Figure 10).

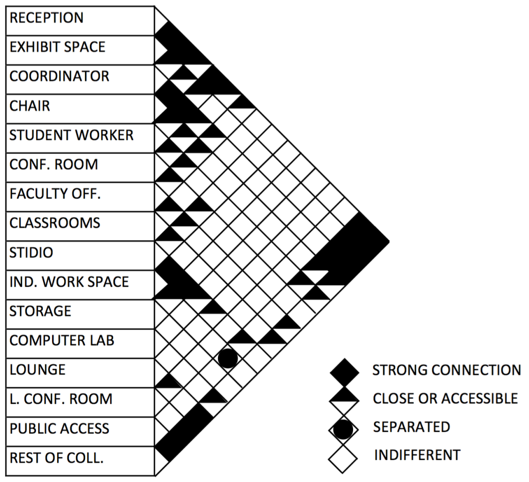


Figure 8, AR-5 Adjacency Diagram. Reprinted from (*Information* requirements specification – [AR-5] early design, 2004).

AR-5 defines two space types: building and functional. Building spaces are the ones that define the anatomy of the building such as rooms, circulation spaces, cores, and structural depth. Functional spaces correlate with the activities and contain identifiable actions on the part of the users. Examples of functional spaces are departments of a hospital or the departure and arrival sections of an airport. Figure 11 shows AR-5 building and functional spaces side by side.

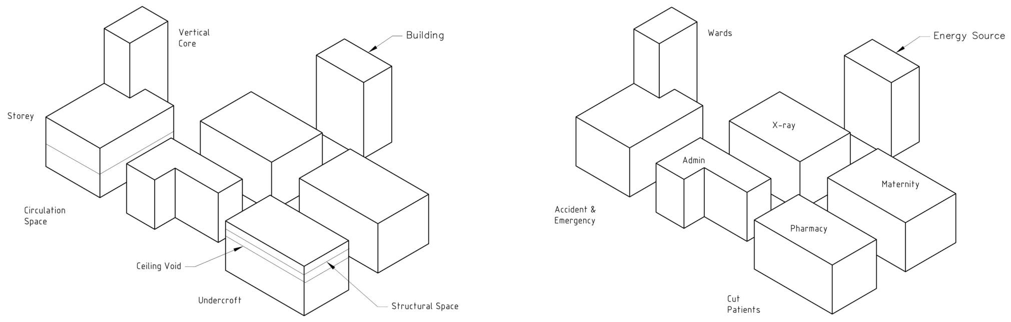


Figure 9, AR-5 Building and Functional Space. Reprinted from (*Information requirements specification – [AR-5] early design,* 2004).

Currently in IFC there are independent class hierarchies for describing the spatial structure independent from functions and activities. IfcSpatialElement, a sub-class of IfcProduct is the base class to describe all the spatial elements such as buildings, floors, and rooms. To describe activities and functions that take place within the spatial elements, IFC offers IfcSpatialZone and IfcZone. Both classes along with their sub-classes create class hierarchies specialized for describing functions and activities.

AR-5 takes a broad approach and does not get involved in the details such as the property sets and their variables. It proposes several concepts to be supported by IFC. While in the latest versions of IFC all the concepts introduced by AR-5 are supported at some level, it is not clear whether the level of coverage in IFC is adequate to support all the programing concepts effectively or not.

2.11.2. SPATIAL COMPLIANCE INFORMATION EXCHANGE (SCie)

Other investigations have explored how to represent the information of POR in a digital form. SCie specifically addresses the need of access to architectural programming information throughout the lifecycle of a facility in public projects (East & Nisbet, 2009). Public projects are funded by tax revenue and public obligations and the process of initiating a new facility starts with a POR. In many cases the request for a new facility is approved or denied based on the program. The POR is also used to estimate the cost of design and construction. By keeping the architectural programming information throughout the lifecycle of a facility, organizations can estimate their functional capacities.

SCie focuses on the delivery of the architectural programming related information on a facility as part of the handoff from construction to maintenance phase. SCie is defined as a subset of COBie standard. COBie stands for Construction-Operation Building Information Exchange (East & Carrasquillo-Mangual, 2012).

COBie focuses on providing a more efficient solution for the significant amount of time spent on handling information in large construction projects. The backbone of the solution is a standard format for documenting such information, which as the inventors describe, can be "ASCII for buildings" (Malleson, Mordue, & Hamil, 2012). What is special about COBie is that while it is a standard most effective when paired with software, it has a tabular human readable format that can be used independently. COBie can be viewed as a collection of Excel worksheets or similar tabular formats such as Google Sheets.

COBie is an IFC MVD (East, 2016). In COBie all the geometric information is eliminated. Besides geometry, the tabular format of COBie also eliminates a large portion of complicated relationship between entities that IFC can represent. There are several applications that extract COBie files from an IFC file. The goal is to provide necessary information about the facility to the parties in charge of management and maintenance. Architectural programming is very essential in such information as it provides a clear picture of the facility's functional capacities.

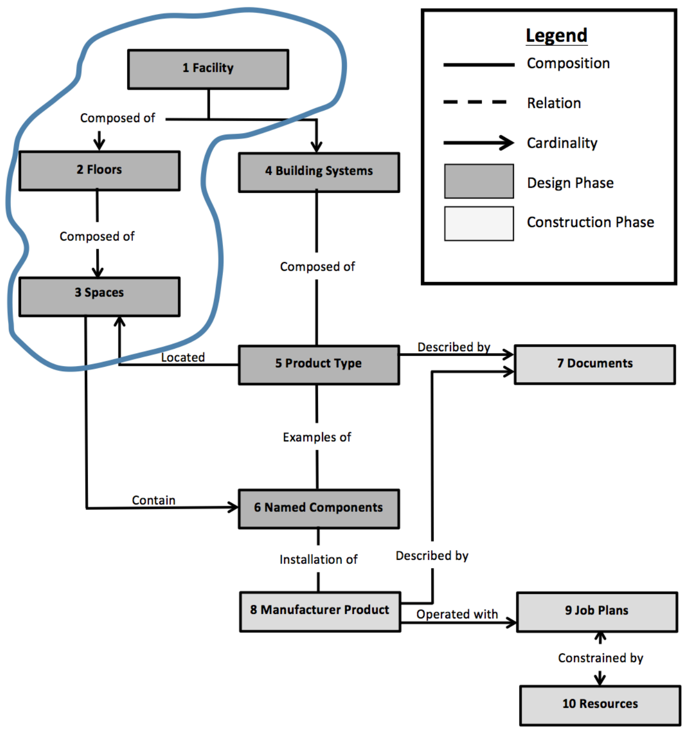


Figure 10, SCie and COBie. Reprinted from (East & Nisbet, 2009).

SCie is intended to support tracking the changes over time. The creators of SCie identify the untracked changes as one of the reasons that PORs lose their value as the project evolves. By using SCie, facility managers and other parties can update the information that is already documented in SCie in every phase of design and construction.

SCie requires the inclusions of four different tables from the COBie standard: Facility worksheet, Floor worksheet, Space worksheet and Contact worksheet. These four tables together represent an important subset of architectural programming information. Facility table provides general information about the facility and does not provide any detail on functional capacities as far as spaces and their areas. Floor and Space table provide similar information on two scales. Both tables provide details on dimensions and areas such as gross area, interior and exterior plannable area, rentable area, etc. Figure 13 – 16 show some of the attributes from the four SCie tables.

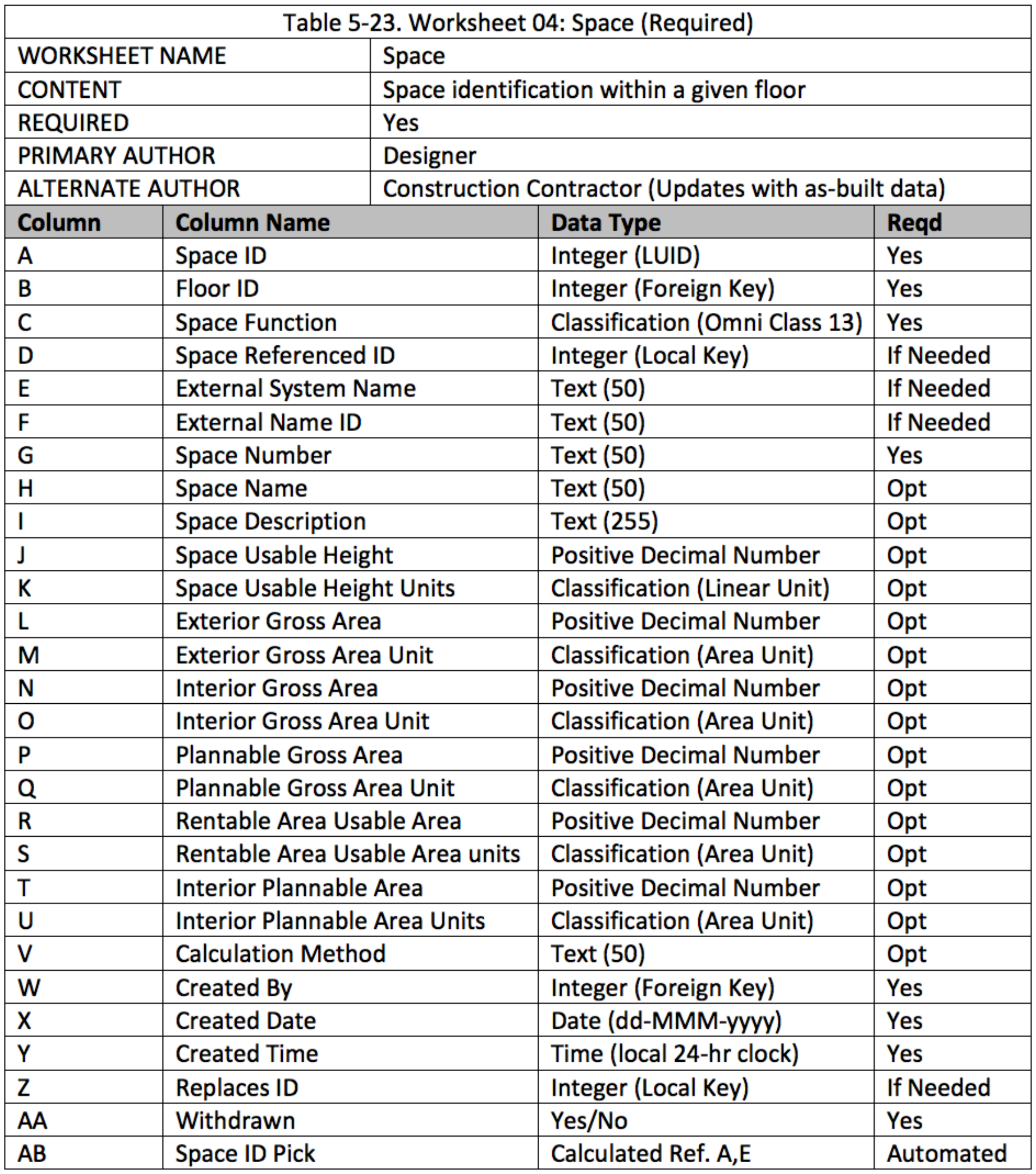


Table 5-23. Worksheet 04: Space (Required)			
WORKSHEET NAME	Space		
CONTENT	Space identification within a given floor		
REQUIRED	Yes		
PRIMARY AUTHOR	Designer		
ALTERNATE AUTHOR	Construction Contractor (Updates with as-built data)		
Column	**Column Name**	**Data Type**	**Reqd**
A	Space ID	Integer (LUID)	Yes
B	Floor ID	Integer (Foreign Key)	Yes
C	Space Function	Classification (Omni Class 13)	Yes
D	Space Referenced ID	Integer (Local Key)	If Needed
E	External System Name	Text (50)	If Needed
F	External Name ID	Text (50)	If Needed
G	Space Number	Text (50)	Yes
H	Space Name	Text (50)	Opt
I	Space Description	Text (255)	Opt
J	Space Usable Height	Positive Decimal Number	Opt
K	Space Usable Height Units	Classification (Linear Unit)	Opt
L	Exterior Gross Area	Positive Decimal Number	Opt
M	Exterior Gross Area Unit	Classification (Area Unit)	Opt
N	Interior Gross Area	Positive Decimal Number	Opt
O	Interior Gross Area Unit	Classification (Area Unit)	Opt
P	Plannable Gross Area	Positive Decimal Number	Opt
Q	Plannable Gross Area Unit	Classification (Area Unit)	Opt
R	Rentable Area Usable Area	Positive Decimal Number	Opt
S	Rentable Area Usable Area units	Classification (Area Unit)	Opt
T	Interior Plannable Area	Positive Decimal Number	Opt
U	Interior Plannable Area Units	Classification (Area Unit)	Opt
V	Calculation Method	Text (50)	Opt
W	Created By	Integer (Foreign Key)	Yes
X	Created Date	Date (dd-MMM-yyyy)	Yes
Y	Created Time	Time (local 24-hr clock)	Yes
Z	Replaces ID	Integer (Local Key)	If Needed
AA	Withdrawn	Yes/No	Yes
AB	Space ID Pick	Calculated Ref. A,E	Automated

Figure 11, SCie Space Table. Reprinted from (East & Nisbet, 2009).

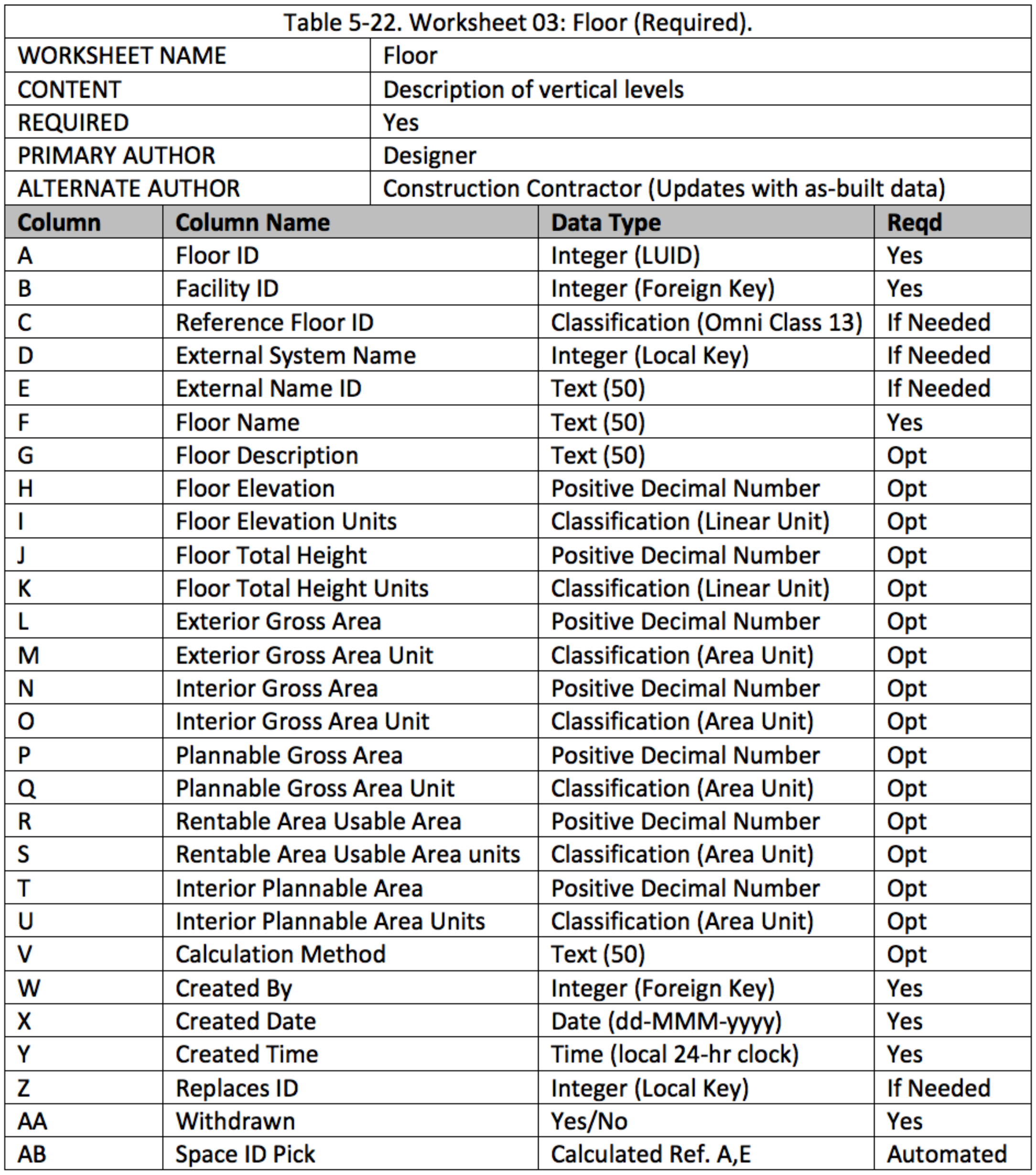


Table 5-22. Worksheet 03: Floor (Required).

WORKSHEET NAME	Floor
CONTENT	Description of vertical levels
REQUIRED	Yes
PRIMARY AUTHOR	Designer
ALTERNATE AUTHOR	Construction Contractor (Updates with as-built data)

Column	**Column Name**	**Data Type**	**Reqd**
A	Floor ID	Integer (LUID)	Yes
B	Facility ID	Integer (Foreign Key)	Yes
C	Reference Floor ID	Classification (Omni Class 13)	If Needed
D	External System Name	Integer (Local Key)	If Needed
E	External Name ID	Text (50)	If Needed
F	Floor Name	Text (50)	Yes
G	Floor Description	Text (50)	Opt
H	Floor Elevation	Positive Decimal Number	Opt
I	Floor Elevation Units	Classification (Linear Unit)	Opt
J	Floor Total Height	Positive Decimal Number	Opt
K	Floor Total Height Units	Classification (Linear Unit)	Opt
L	Exterior Gross Area	Positive Decimal Number	Opt
M	Exterior Gross Area Unit	Classification (Area Unit)	Opt
N	Interior Gross Area	Positive Decimal Number	Opt
O	Interior Gross Area Unit	Classification (Area Unit)	Opt
P	Plannable Gross Area	Positive Decimal Number	Opt
Q	Plannable Gross Area Unit	Classification (Area Unit)	Opt
R	Rentable Area Usable Area	Positive Decimal Number	Opt
S	Rentable Area Usable Area units	Classification (Area Unit)	Opt
T	Interior Plannable Area	Positive Decimal Number	Opt
U	Interior Plannable Area Units	Classification (Area Unit)	Opt
V	Calculation Method	Text (50)	Opt
W	Created By	Integer (Foreign Key)	Yes
X	Created Date	Date (dd-MMM-yyyy)	Yes
Y	Created Time	Time (local 24-hr clock)	Yes
Z	Replaces ID	Integer (Local Key)	If Needed
AA	Withdrawn	Yes/No	Yes
AB	Space ID Pick	Calculated Ref. A,E	Automated

Figure 12, SCie Floor Table. Reprinted from (East & Nisbet, 2009).

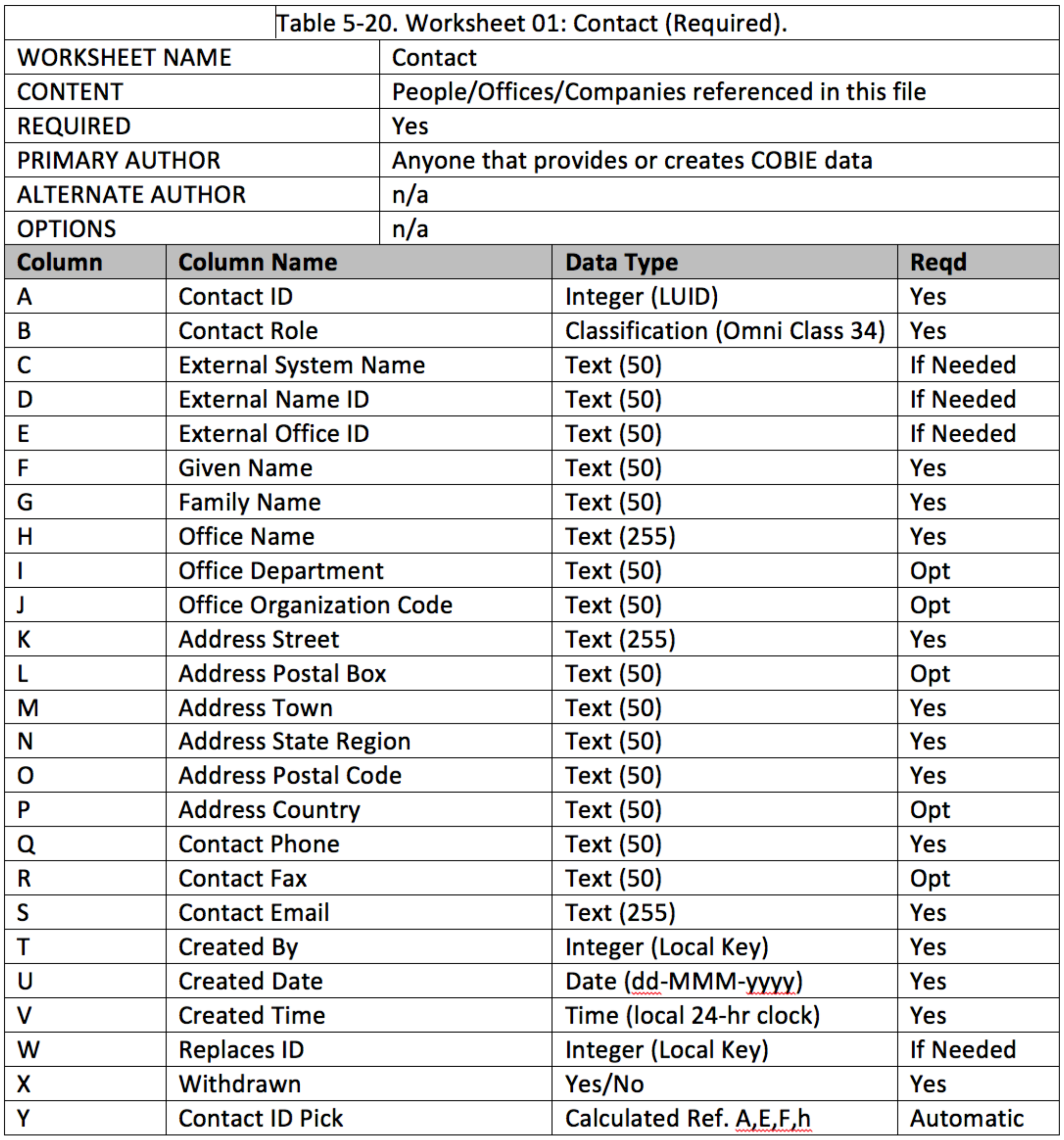


Table 5-20. Worksheet 01: Contact (Required).

WORKSHEET NAME	Contact
CONTENT	People/Offices/Companies referenced in this file
REQUIRED	Yes
PRIMARY AUTHOR	Anyone that provides or creates COBIE data
ALTERNATE AUTHOR	n/a
OPTIONS	n/a

Column	**Column Name**	**Data Type**	**Reqd**
A	Contact ID	Integer (LUID)	Yes
B	Contact Role	Classification (Omni Class 34)	Yes
C	External System Name	Text (50)	If Needed
D	External Name ID	Text (50)	If Needed
E	External Office ID	Text (50)	If Needed
F	Given Name	Text (50)	Yes
G	Family Name	Text (50)	Yes
H	Office Name	Text (255)	Yes
I	Office Department	Text (50)	Opt
J	Office Organization Code	Text (50)	Opt
K	Address Street	Text (255)	Yes
L	Address Postal Box	Text (50)	Opt
M	Address Town	Text (50)	Yes
N	Address State Region	Text (50)	Yes
O	Address Postal Code	Text (50)	Yes
P	Address Country	Text (50)	Opt
Q	Contact Phone	Text (50)	Yes
R	Contact Fax	Text (50)	Opt
S	Contact Email	Text (255)	Yes
T	Created By	Integer (Local Key)	Yes
U	Created Date	Date (dd-MMM-yyyy)	Yes
V	Created Time	Time (local 24-hr clock)	Yes
W	Replaces ID	Integer (Local Key)	If Needed
X	Withdrawn	Yes/No	Yes
Y	Contact ID Pick	Calculated Ref. A,E,F,h	Automatic

Figure 13, SCie Contact Table. Reprinted from (East & Nisbet, 2009).

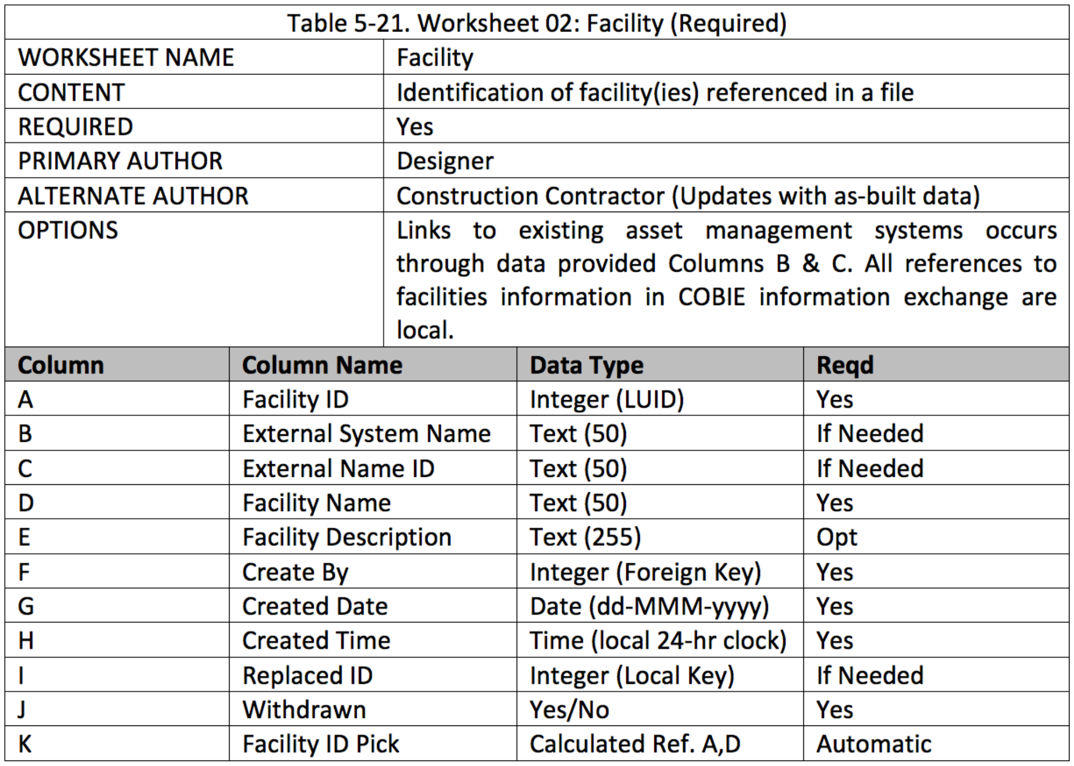


Table 5-21. Worksheet 02: Facility (Required)

WORKSHEET NAME	Facility
CONTENT	Identification of facility(ies) referenced in a file
REQUIRED	Yes
PRIMARY AUTHOR	Designer
ALTERNATE AUTHOR	Construction Contractor (Updates with as-built data)
OPTIONS	Links to existing asset management systems occurs through data provided Columns B & C. All references to facilities information in COBIE information exchange are local.

Column	**Column Name**	**Data Type**	**Reqd**
A	Facility ID	Integer (LUID)	Yes
B	External System Name	Text (50)	If Needed
C	External Name ID	Text (50)	If Needed
D	Facility Name	Text (50)	Yes
E	Facility Description	Text (255)	Opt
F	Create By	Integer (Foreign Key)	Yes
G	Created Date	Date (dd-MMM-yyyy)	Yes
H	Created Time	Time (local 24-hr clock)	Yes
I	Replaced ID	Integer (Local Key)	If Needed
J	Withdrawn	Yes/No	Yes
K	Facility ID Pick	Calculated Ref. A,D	Automatic

Figure 14, SCie Facility Table. Reprinted from (East & Nisbet, 2009).

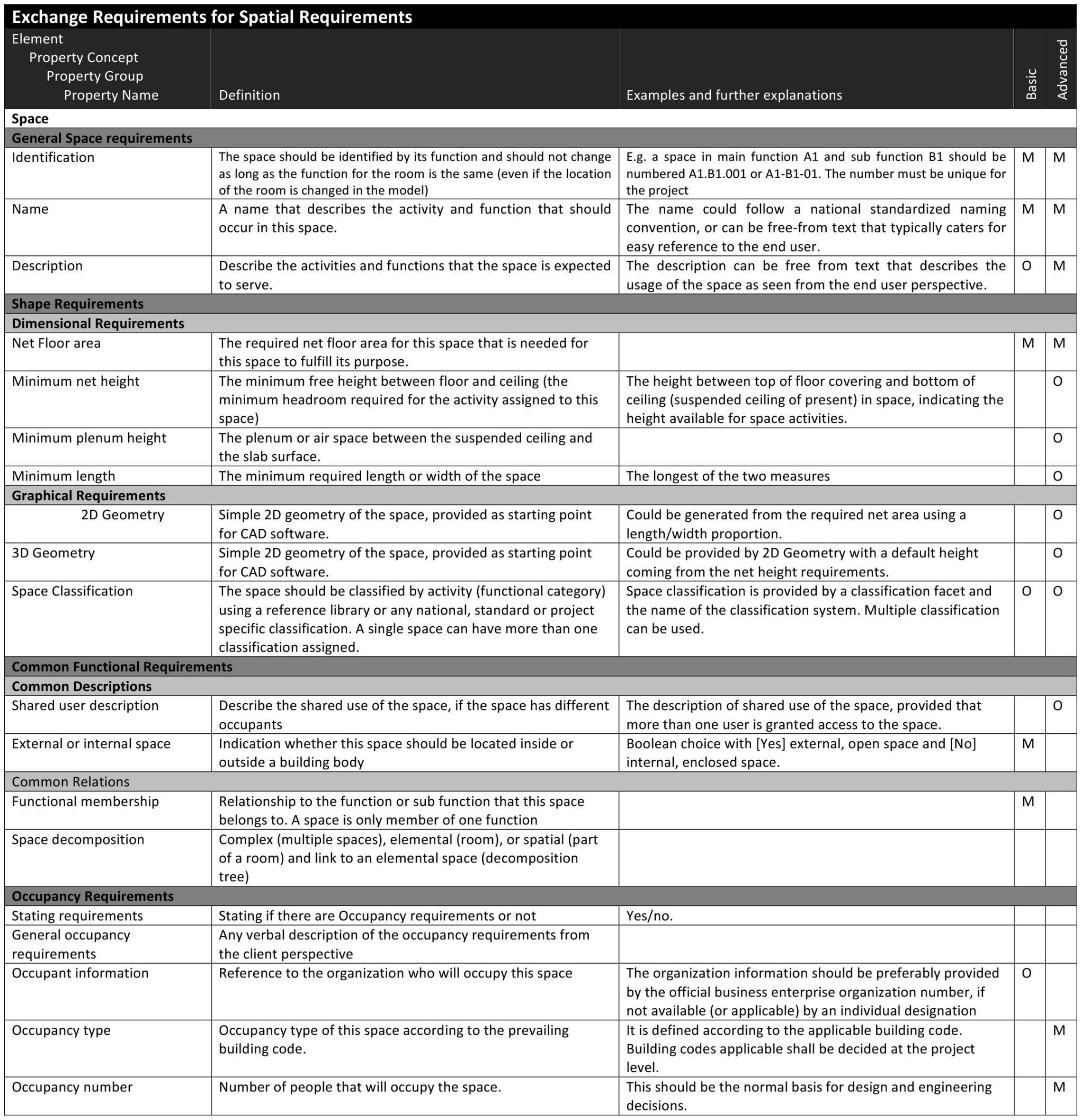
SCie creates an abbreviated version of the POR that is tailored toward facility maintenance and management. The SCie information can be delivered included in a COBie file or as a standalone file.

2.11.3. BUILDING PROGRAMMING INFORMATION EXCHANGE (BPie)

Comparing to SCie, BPie is a more recent effort to capture architectural programming information through a BIM compatible format. While SCie is more focused on providing architectural programming information for the FM industry, BPie offers the following definition: “Building programming is the process of collecting all requirements the building must fulfill" (East, 2012). BPie also regards a POR as a dynamic document that changes through the lifecycle of the project; therefore it should be included in BIM models and be available for modification. BPie gathers the requirements imposed by a POR to support evaluation of the design against them. The difference between the use cases of the two standards reflects on their content. SCie addresses the as-built information regardless of whether they fulfill the requirements or not. SCie relies on architectural programming information since it is a reflection on the functional capacity of the facility, while BPie regards architectural programming as a collection of requirements. BPie might include information regarding the required ventilation airflow of a specific space; such information is not of use in SCie.

The other major difference between the two standards is the data model upon which they are based. SCie is defined as a subset of COBie as it is meant to support the FM industry. COBie is a stripped down version of IFC that essentially provides a less elaborated data model. BPie is defined as a direct MVD within the IFC data model and provides a more flexible data model that can capture a wider array of information.

BPie relies on five different entities from IFC data model to capture architectural programming information. The five entities are: Project, Building, Story, Space, and Zone. Attaching additional properties to these five entities represents the requirements. These properties are grouped together under several property sets.



Exchange Requirements for Spatial Requirements				
Element Property Concept Property Group Property Name	Definition	Examples and further explanations	Basic	Advanced
Space				
General Space requirements				
Identification	The space should be identified by its function and should not change as long as the function for the room is the same (even if the location of the room is changed in the model)	E.g. a space in main function A1 and sub function B1 should be numbered A1.B1.001 or A1-B1-01. The number must be unique for the project	M	M
Name	A name that describes the activity and function that should occur in this space.	The name could follow a national standardized naming convention, or can be free-from text that typically caters for easy reference to the end user.	M	M
Description	Describe the activities and functions that the space is expected to serve.	The description can be free from text that describes the usage of the space as seen from the end user perspective.	O	M
Shape Requirements				
Dimensional Requirements				
Net Floor area	The required net floor area for this space that is needed for this space to fulfill its purpose.		M	M
Minimum net height	The minimum free height between floor and ceiling (the minimum headroom required for the activity assigned to this space)	The height between top of floor covering and bottom of ceiling (suspended ceiling of present) in space, indicating the height available for space activities.		O
Minimum plenum height	The plenum or air space between the suspended ceiling and the slab surface.			O
Minimum length	The minimum required length or width of the space	The longest of the two measures		O
Graphical Requirements				
2D Geometry	Simple 2D geometry of the space, provided as starting point for CAD software.	Could be generated from the required net area using a length/width proportion.		O
3D Geometry	Simple 2D geometry of the space, provided as starting point for CAD software.	Could be provided by 2D Geometry with a default height coming from the net height requirements.		O
Space Classification	The space should be classified by activity (functional category) using a reference library or any national, standard or project specific classification. A single space can have more than one classification assigned.	Space classification is provided by a classification facet and the name of the classification system. Multiple classification can be used.	O	O
Common Functional Requirements				
Common Descriptions				
Shared user description	Describe the shared use of the space, if the space has different occupants	The description of shared use of the space, provided that more than one user is granted access to the space.		O
External or internal space	Indication whether this space should be located inside or outside a building body	Boolean choice with [Yes] external, open space and [No] internal, enclosed space.	M	
Common Relations				
Functional membership	Relationship to the function or sub function that this space belongs to. A space is only member of one function		M	
Space decomposition	Complex (multiple spaces), elemental (room), or spatial (part of a room) and link to an elemental space (decomposition tree)			
Occupancy Requirements				
Stating requirements	Stating if there are Occupancy requirements or not	Yes/no.		
General occupancy requirements	Any verbal description of the occupancy requirements from the client perspective			
Occupant information	Reference to the organization who will occupy this space	The organization information should be preferably provided by the official business enterprise organization number, if not available (or applicable) by an individual designation	O	
Occupancy type	Occupancy type of this space according to the prevailing building code.	It is defined according to the applicable building code. Building codes applicable shall be decided at the project level.		M
Occupancy number	Number of people that will occupy the space.	This should be the normal basis for design and engineering decisions.		M

Figure 15, A Partial List of BPie Space Attributes. Reprinted from (East & Nisbet, 2009).

2.11.4. CURRENT STATE OF ARCHITECTURAL PROGRAMMING AND IFC

The three reviewed projects aim to enhance the capabilities of IFC in capturing architectural programming information in different capacities. Activity is a key word in

AR-5. Differentiating between physical spaces and activities is discussed as the key concept in adding architectural programming requirements to IFC. AR-5 provides general guidelines and does not engage in detailed discussions on how to incorporate architectural programming into IFC.

SCie focuses on making architectural programming information accessible for the facility managers. SCie is defined based on COBie, therefore inheres all the limitation of the COBie standard. While facility managers can benefit from having access to parts of architectural programming information, there are many other domains that benefit from such information.

Comparing to the first two standards BPie has a more inclusive approach in adding architectural programming information to IFC. In comparison with SCie, using BPie the requirements are directly added to an IFC file instead of the more limited COBie format. BPie provides a collection of predefined requirements for five IFC entities (IfcProject, IfcBuilding, IfcSpace, and IfcZone).

3. COMMERCIAL SOFTWARE FOR ARCHITECTURAL PROGRAMMING

Although many of the data modeling standards are used through software packages, some of the data models propose human readable formats that can be used without the support of software. In AECO industry, many of the standards are developed to be used through a software package. DWG, for example, is not considered a human readable data model. There are other standards in the AECO industry that rely less on software. COBie for example provides a human readable standard structure.

Weather the output of the standards are human readable or not, the interaction between the architectural programmers and the data standard is almost always through software. To understand the common practices and limitations of documenting PORs though software, this section provides reviews of some of the computer applications that are designed for this purpose.

3.1. COMMERCIAL TOOLS

Creating a bridge between architectural programming and Computer-Aided Design has been an appealing matter to software vendors. Trelligence Affinity (About Trelligence, 2015), Onuma System (About Onuma, 2015), and dRofus (About dRofus AS, 2015) are three of the major products available in the market to manage POR information throughout the lifecycle of a facility.

All the solutions cited above approach the problem by creating an independent user interface and data models to store the programming data. The user starts the design task in this environment and eventually exports the model to a BIM design tool.

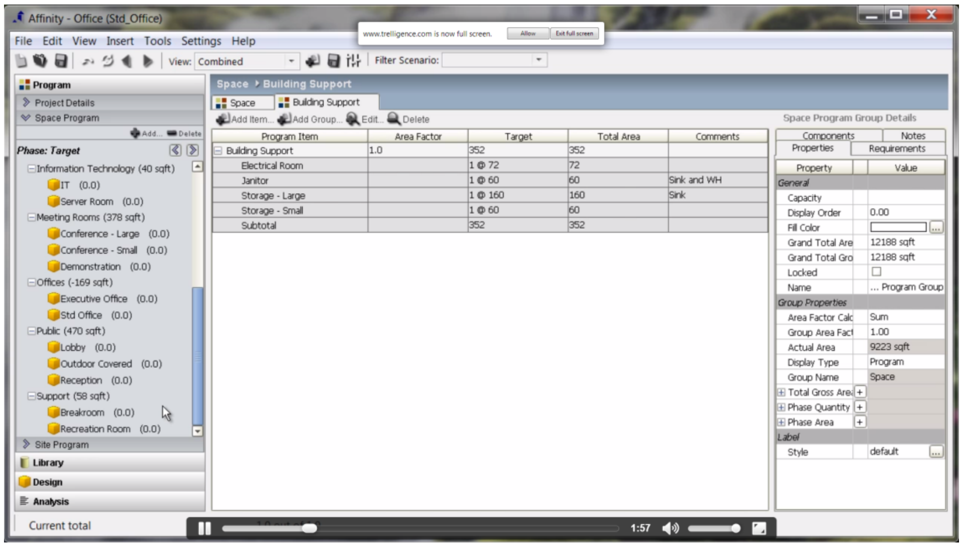
Although this approach can potentially solve the problem, it also brings some limitations to the table. Introducing a brand new gadget to the architects' toolbox of applications requires tweaking of the design process and making room for the new tool. The POR data may also become inextricably tied to the software and thus inaccessible for other purposes.

3.1.1. TRELLIGENCE AFFINITY

Trelligence is a software development company founded in 2003 based in Houston, TX. AECO industry is the main focus of the company. All the members of the management team cited by Trelligence website have a software development background and prior to their current endeavor have not been involved in the AECO industry. Trelligence Affinity is the main and only product of the company (About Trelligence, 2015).

Trelligence Affinity provides solutions for architectural programming, planning and early stages of design. Trelligence divides the functionalities provided by Trelligence Affinity into three categories, Programming, Schematic Design, and Design to Revit.

Affinity Programming allows the users to capture and manage project requirements, components, notes and relationships. The process of creating a POR in Affinity starts with creating a new project and placing some departments in the project. New spaces will be added to the department. Area and adjacency requirements can be added to the spaces (Figure 18). Adjacency requirements can be added to the program by modifying the relationship property of the added spaces.



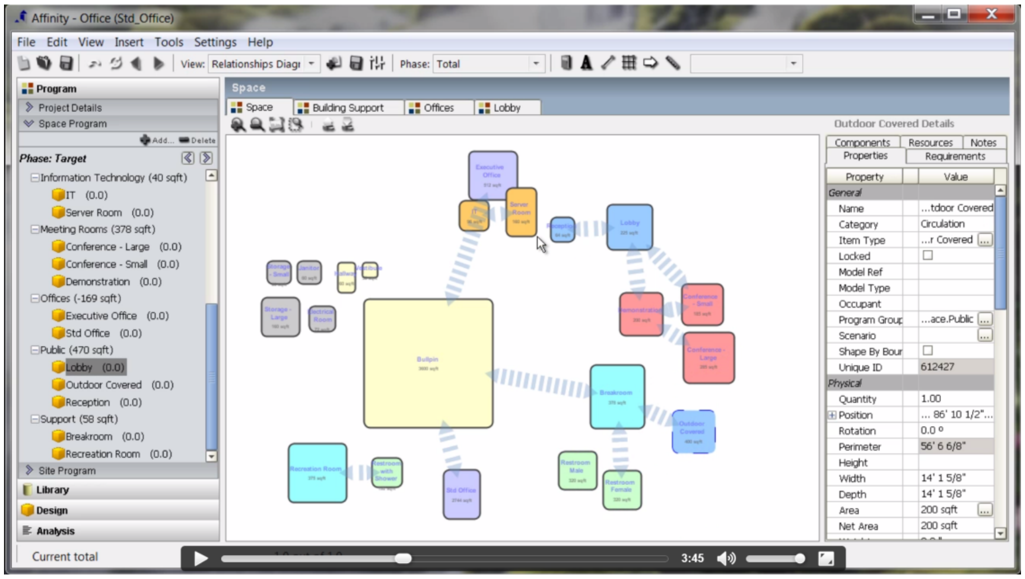
Program Item	Area Factor	Target	Total Area	Comments
Building Support	1.0	352	352	
Electrical Room		1 @ 72	72	
Janitor		1 @ 60	60	Sink and WH
Storage - Large		1 @ 160	160	Sink
Storage - Small		1 @ 60	60	
Subtotal		352	352	

Property	Value
General	
Capacity	
Display Order	0.00
Fill Color	
Grand Total Are	12188 sqft
Grand Total Gro	12188 sqft
Locked	☐
Name	... Program Group
Group Properties	
Area Factor Calc	Sum
Group Area Fact	1.00
Actual Area	9223 sqft
Display Type	Program
Group Name	Space
Total Gross Are	
Phase Quantity	
Phase Area	
Label	
Style	default

Figure 16, Affinity Programming Tool

Affinity Schematic Design allows the user to generate spaces by dragging items from the program into the design user interface. The application compares the design information with the program and informs the user if there is any incompatibility in the space count (Figure 19).

The Schematic Design tool allows the users to add spaces that are not listed in the programs to the layout. Therefore additional spaces such as hallways and corridors that are not required by the POR can be added to the layout. The relationship between the Programming component and the Schematic Design component is a one way relationship and adding a new space in the Schematic Design component will not update the POR. Spaces that are added merely through the schematic design tool should be added manually to the POR as well.

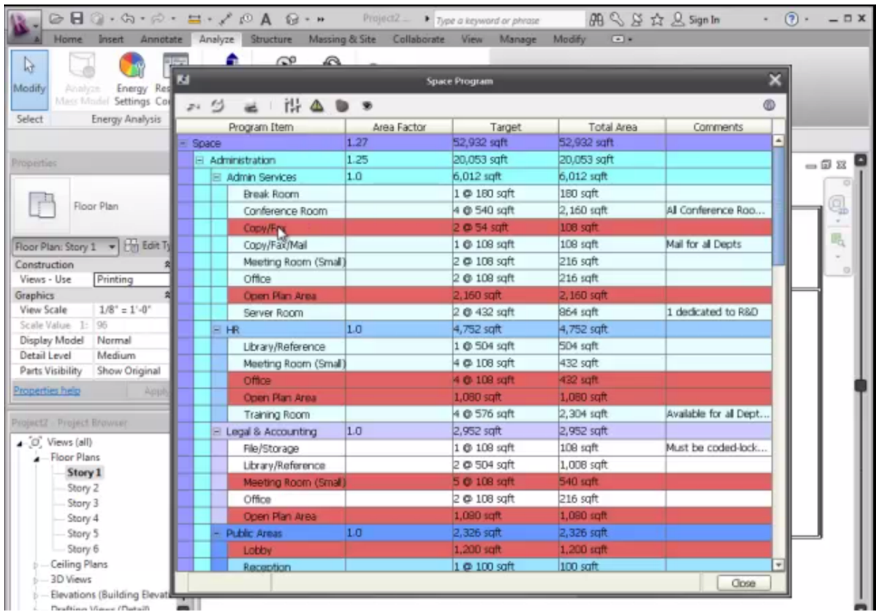


Property	Value
General	
Name	...tdoor Covered
Category	Circulation
Item Type	...r Covered
Locked	☐
Model Ref	
Model Type	
Occupant	
Program Grour	...ace.Public
Scenario	
Shape By Bour	☐
Unique ID	612427
Physical	
Quantity	1.00
Position	... 86' 10 1/2"...
Rotation	0.0 º
Perimeter	56' 6 6/8"
Height	
Width	14' 1 5/8"
Depth	14' 1 5/8"
Area	200 sqft
Net Area	200 sqft

Figure 17, Affinity Schematic Design

The space layout is drawn on a 2D plan view. There are no architectural section/elevation views available to modify the design.

Affinity Design to Revit is an Autodesk Revit Architecture plugin. Through this plugin the user can import a POR along with a space layout generated in Trelligence Affinity Programming and Schematic Design tool into Revit. The plugin also includes a monitoring option, which enables the operator to check requirements compatibility of the design model within the Revit environment (Figure 20).



Program Item	Area Factor	Target	Total Area	Comments
Space	1.27	52,932 sqft	52,932 sqft	
Administration	1.25	20,053 sqft	20,053 sqft	
Admin Services	1.0	6,012 sqft	6,012 sqft	
Break Room		1 @ 180 sqft	180 sqft	
Conference Room		4 @ 540 sqft	2,160 sqft	All Conference Roo...
Copy/Fax		2 @ 54 sqft	108 sqft	
Copy/Fax/Mail		1 @ 108 sqft	108 sqft	Mail for all Depts
Meeting Room (Small)		2 @ 108 sqft	216 sqft	
Office		2 @ 108 sqft	216 sqft	
Open Plan Area		2,160 sqft	2,160 sqft	
Server Room		2 @ 432 sqft	864 sqft	1 dedicated to R&D
HR	1.0	4,752 sqft	4,752 sqft	
Library/Reference		1 @ 504 sqft	504 sqft	
Meeting Room (Small)		4 @ 108 sqft	432 sqft	
Office		4 @ 108 sqft	432 sqft	
Open Plan Area		1,080 sqft	1,080 sqft	
Training Room		4 @ 576 sqft	2,304 sqft	Available for all Dept...
Legal & Accounting	1.0	2,952 sqft	2,952 sqft	
File/Storage		1 @ 108 sqft	108 sqft	Must be coded-lock...
Library/Reference		2 @ 504 sqft	1,008 sqft	
Meeting Room (Small)		5 @ 108 sqft	540 sqft	
Office		2 @ 108 sqft	216 sqft	
Open Plan Area		1,080 sqft	1,080 sqft	
Public Areas	1.0	2,326 sqft	2,326 sqft	
Lobby		1,200 sqft	1,200 sqft	
Reception		1 @ 100 sqft	100 sqft	

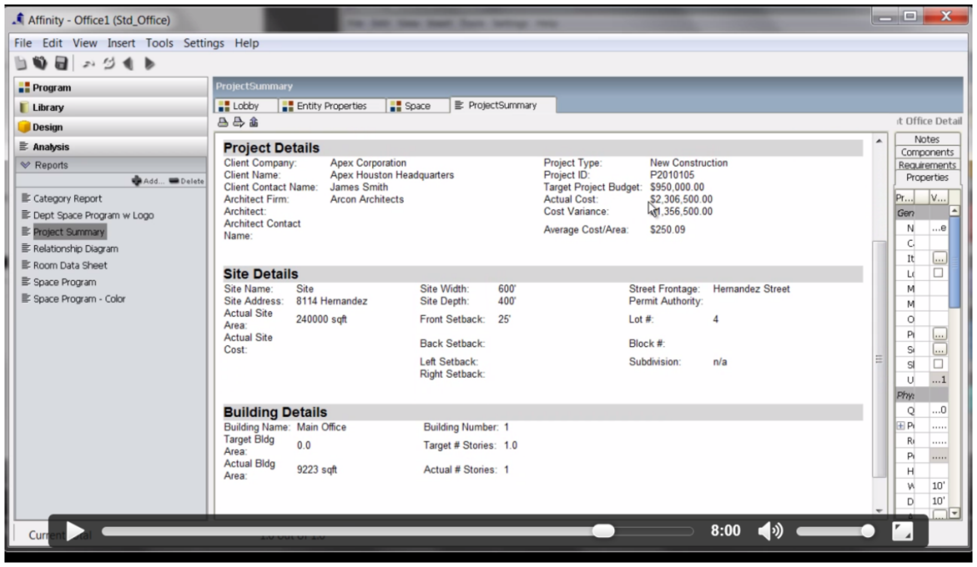
Figure 18, Affinity Design to Revit

Below are some of the main characteristics of the solution provided by Trelligence.

- *Design Environment,* Affinity not only provides the features required to enter POR requirements, it also provides tools and means for planning and design. The embedded design and planning tool allows users to draw basic 2d shapes. The modeling toolset is not comparable to advanced BIM modeling applications such as Autodesk Revit that the designers and planners are accustomed to outside of Affinity. The users need to choose between having direct access to programming information and using less sophisticated design tools, or using more sophisticated design tools and compromise on their access to the programming data.

- *The Scope of Architectural Programming,* The current version of Affinity does not support the entire span of architectural programing information proposed by the textbooks. Therefore the user may have to use other tools in conjunction with Affinity to cover those fields. Chapter 4 describes several textbooks on architectural programming.
- *Creating Space Packages,* The application provides only one level of space packaging. Space packaging is categorizing spaces in groups based on one or multiple common attribute. In Affinity the user can put the spaces into packages, but one cannot group those packages any further. While this might be adequate for a simple project, in more complicated projects (e.g. healthcare facilities) there are multiple levels of packaging imposed through the program.
- *Interaction with BIM Data Models,* While the main advantage of BIM is to manage the project data over the lifecycle of the building, Trelligence provides a one-way export to Revit architecture. All the programming requirements provided in Affinity will be added to the Revit model as room properties but they lose some of their automated verification functionalities. By exporting the Revit file into another BIM format or application all the imported room properties will be eliminated from the generated BIM model.

- *Adjacency Assessment.* While the user can add adjacency requirements to the Affinity edition of program, there is no mechanism to check if the requirements are met or not.
- *Reporting Option.* After entering the POR requirements, Affinity gives the user the option to present the data entries with different formats. The user can choose the proper format for different procedures, such as presenting the POR to the client, design team, and other stakeholders (Figure 21).
- *An Intuitive Relationship Between Design and POR,* Trelligence Affinity provides an intuitive mechanism for placing items from the program into the design. The users can drag the items from the program and place them into the design.



Project Details

Client Company:	Apex Corporation	Project Type:	New Construction
Client Name:	Apex Houston Headquarters	Project ID:	P2010105
Client Contact Name:	James Smith	Target Project Budget:	$950,000.00
Architect Firm:	Arcon Architects	Actual Cost:	$2,306,500.00
Architect:		Cost Variance:	1,356,500.00
Architect Contact Name:		Average Cost/Area:	$250.09

Site Details

Site Name:	Site	Site Width:	600'	Street Frontage:	Hernandez Street
Site Address:	8114 Hernandez	Site Depth:	400'	Permit Authority:	
Actual Site Area:	240000 sqft	Front Setback:	25'	Lot #:	4
Actual Site Cost:		Back Setback:		Block #:	
		Left Setback:		Subdivision:	n/a
		Right Setback:			

Building Details

Building Name:	Main Office	Building Number:	1
Target Bldg Area:	0.0	Target # Stories:	1.0
Actual Bldg Area:	9223 sqft	Actual # Stories:	1

Figure 19, Trelligence Affinity Reporting Options

3.1.2. ONUMA SYSTEMS

Unlike Trelligence, Onuma is an architectural firm with an interest in software development for architects. They claim to use their own products during the design process. Onuma is a Web-based system so it does not require any software installation prior to use (About Onuma, 2015).

Being a Web-based application enables Onuma to access all the information of online services (e.g. Google maps) and incorporate them into the BIM model. On the other hand, it has limited the capabilities of Onuma in providing a more sophisticated user interface. Below you can see the Onuma user interface (Figure 22).

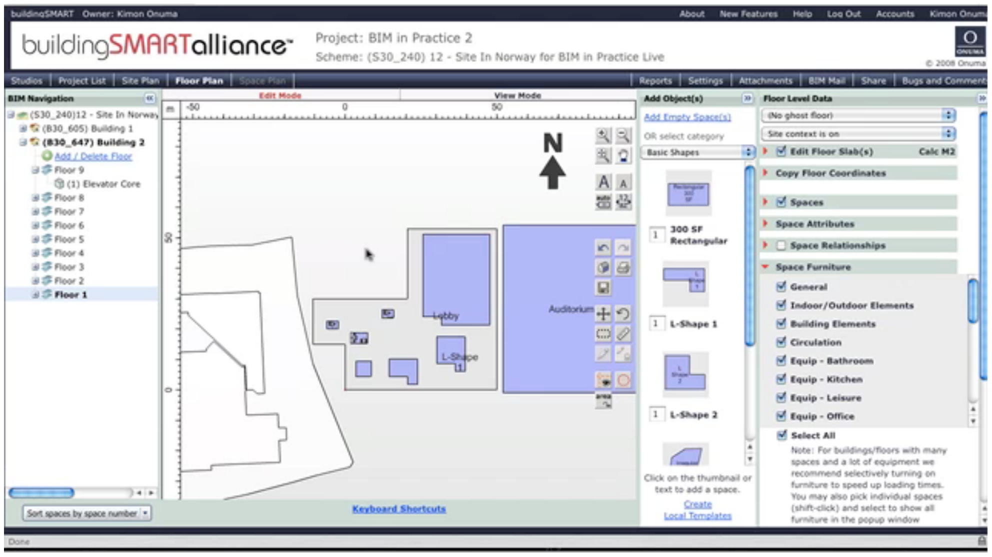


Figure 20, Onuma Systems Main User Interface

The design functionalities provided by Onuma are very similar to the ones of Trelligence. One of the main concepts behind Onuma System is to capitalize on all the opportunities of being a Web-based service. This includes an inbuilt messaging system, interacting with online GIS systems such as Google earth and Google maps, and interacting with social media.

The incorporation of POR into design in Onuma System is very limited. Every model in Onuma can be exported to an Excel file and imported back. In other words, you can have two different views of the same data. The user can initiate a project either in Excel or in Onuma. By starting the project in Excel the user can assign a target and dimensions to each space. The user can also assign a room to a certain floor level. When Onuma System imports the Excel file, it generates respective spaces based on the defined parameters.

The scope of a POR in Onuma System does not go beyond specifying the area and the level. There is no automated mechanism to check the design versus the programming requirements. The requirements are not defined independent from the design attributes, therefore comparison is impossible. The only mechanism offered by Onuma System to enforce any requirements is locking the space size and ratio.

Below are the main characteristics of Onuma System.

- *Web Based Application,* One of the main advantages of web-based applications is data synchronization. All the members of the design team work on the same model and avoid generating multiple copies on the desktop computers (Lapierre & Cote, 2007).

- *3D Components,* Unlike Trelligence Affinity, the components generated in Onuma are 3D BIM components. This would add a great deal of accuracy and practicality to the Onuma System.
- *IFC Support,* Onuma supports IFC format by providing the capability of exporting the Onuma System model as an IFC file to other BIM applications. It is not clear if there is a level of data loss involved during the conversion.
- *Adjacency Control,* There is no automated method provided to assess the adjacencies. After adding adjacency requirements, the adjacencies will be shown in the model through connecting lines between the design components (Figure 23).
- *Separation of Programming and Design Requirements,* There are no parameters imbedded in the Onuma System models to record the programming requirements. Therefore any level of comparison between design and program is out of reach.
- *Flexibility,* Onuma System has to be kept very simple to run on the Web. This has limited the software flexibility. Unlike Trelligence Affinity, custom parameters cannot be added to the POR format.

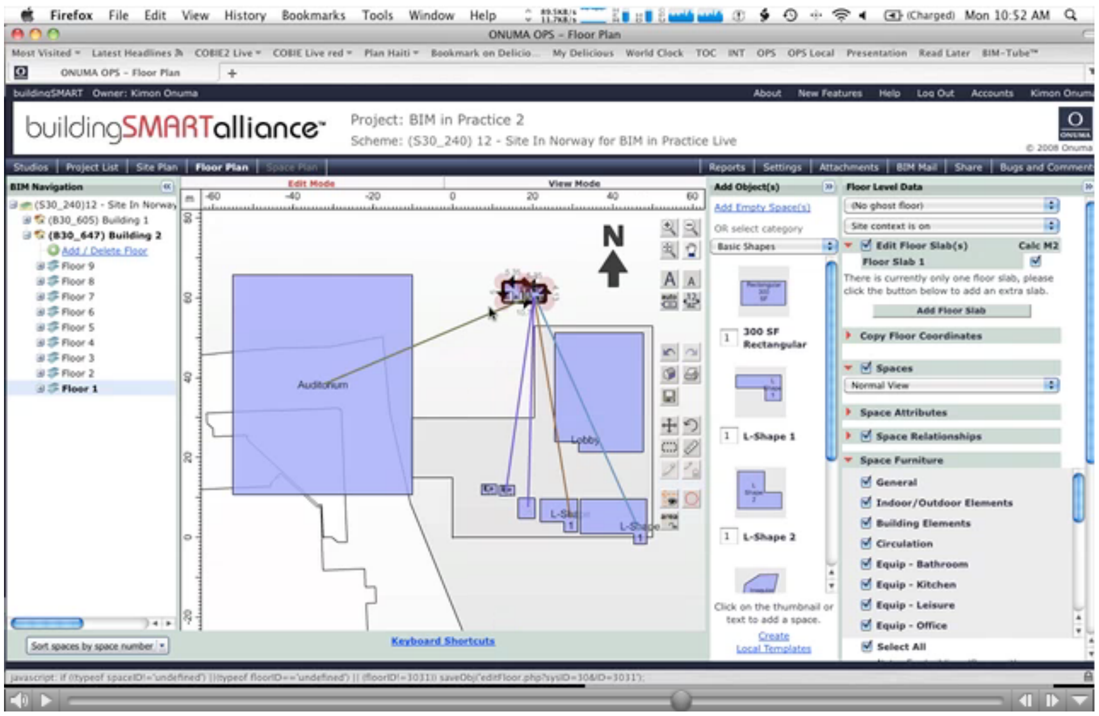
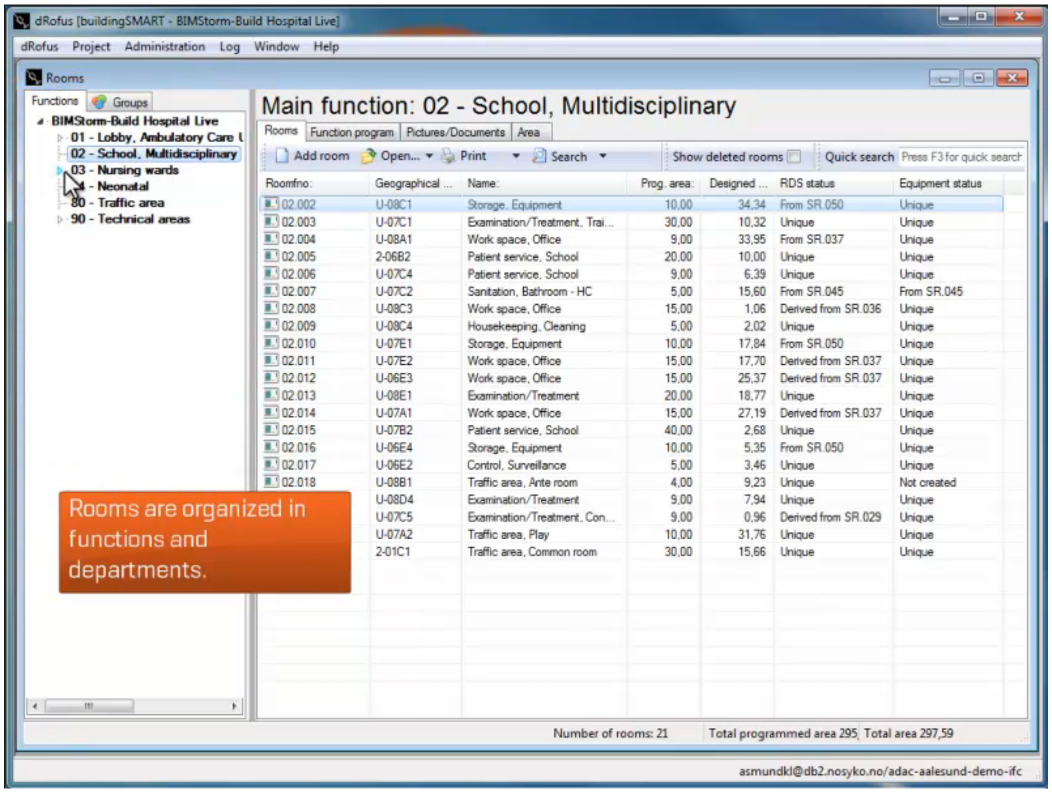


Figure 21, Onuma Adjacency Control

3.1.3. dROFUS

dRofus was originally developed by Nosyko, a leading consulting company for hospital planning in Norway. dRofus was established as a separate company in 2011 and commercialized dRofus computer application as a planning and data management tool for the building industry.

dRofus user interface is very similar to the one of Trelligence Affinity (Figure 24). The rooms are organized in groups and any further packaging of the groups is not supported. One of the major advantages of dRofus over Trelligence is the import feature from IFC files that are compatible with BPie standard. (About dRofus AS, 2015).


dRofus [buildingSMART - BIMStorm-Build Hospital Live]
dRofus Project Administration Log Window Help
Rooms
Functions
Groups
BIMStorm-Build Hospital Live
01 - Lobby, Ambulatory Care
02 - School, Multidisciplinary
03 - Nursing wards
04 - Neonatal
80 - Traffic area
90 - Technical areas
Rooms are organized in functions and departments.
Main function: 02 - School, Multidisciplinary
Rooms
Function program
Pictures/Documents
Area
Add room
Open...
Print
Search
Show deleted rooms
Quick search
Press F3 for quick search
Number of rooms: 21
Total programmed area 295, Total area 297,59
asmundkl@db2.nosyko.no/adac-aalesund-demo-ifc

Roomfno:	Geographical ...	Name:	Prog. area:	Designed ...	RDS status	Equipment status
02.002	U-08C1	Storage, Equipment	10,00	34,34	From SR.050	Unique
02.003	U-07C1	Examination/Treatment, Trai...	30,00	10,32	Unique	Unique
02.004	U-08A1	Work space, Office	9,00	33,95	From SR.037	Unique
02.005	2-06B2	Patient service, School	20,00	10,00	Unique	Unique
02.006	U-07C4	Patient service, School	9,00	6,39	Unique	Unique
02.007	U-07C2	Sanitation, Bathroom - HC	5,00	15,60	From SR.045	From SR.045
02.008	U-08C3	Work space, Office	15,00	1,06	Derived from SR.036	Unique
02.009	U-08C4	Housekeeping, Cleaning	5,00	2,02	Unique	Unique
02.010	U-07E1	Storage, Equipment	10,00	17,84	From SR.050	Unique
02.011	U-07E2	Work space, Office	15,00	17,70	Derived from SR.037	Unique
02.012	U-06E3	Work space, Office	15,00	25,37	Derived from SR.037	Unique
02.013	U-08E1	Examination/Treatment	20,00	18,77	Unique	Unique
02.014	U-07A1	Work space, Office	15,00	27,19	Derived from SR.037	Unique
02.015	U-07B2	Patient service, School	40,00	2,68	Unique	Unique
02.016	U-06E4	Storage, Equipment	10,00	5,35	From SR.050	Unique
02.017	U-06E2	Control, Surveillance	5,00	3,46	Unique	Unique
02.018	U-08B1	Traffic area, Ante room	4,00	9,23	Unique	Not created
	U-08D4	Examination/Treatment	9,00	7,94	Unique	Unique
	U-07C5	Examination/Treatment, Con...	9,00	0,96	Derived from SR.029	Unique
	U-07A2	Traffic area, Play	10,00	31,76	Unique	Unique
	2-01C1	Traffic area, Common room	30,00	15,66	Unique	Unique

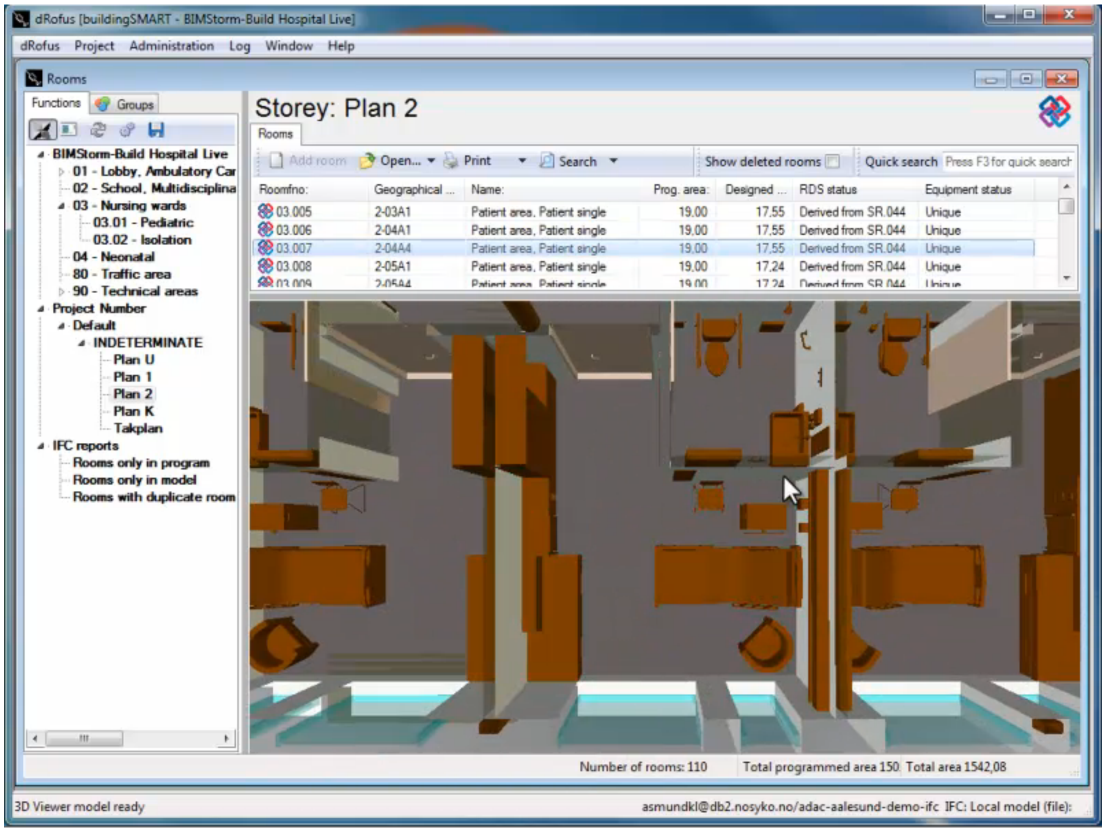
Figure 22, dRofus Home Page

dRofus provides a more comprehensive list of requirements for each room. The requirements are grouped under these titles: Description, Design/building, Windows and doors, HVAC, Air conditioning, El power, ICT alarm and signal, Acoustics, Operation, Fire, Equipment, Pictures/Documents.

Unlike Trelligence and Onuma, dRofus does not provide any design generation features and the user is limited to creating the POR. To use the created POR in a design environment, the POR should be exported to the IFC format.

The design information can be added to the IFC file generated by dRofus. Alternatively, the design application can initialize the IFC file and the POR can be integrated into the same file.

dRofus comes with an IFC viewer that does side by side comparison of the design and the POR (Figure 25).



Roomfno:	Geographical ...	Name:	Prog. area:	Designed ...	RDS status	Equipment status
03.005	2-03A1	Patient area, Patient single	19,00	17,55	Derived from SR.044	Unique
03.006	2-04A1	Patient area, Patient single	19,00	17,55	Derived from SR.044	Unique
03.007	2-04A4	Patient area, Patient single	19,00	17,55	Derived from SR.044	Unique
03.008	2-05A1	Patient area, Patient single	19,00	17,24	Derived from SR.044	Unique
03.009	2-05A4	Patient area, Patient single	19,00	17,24	Derived from SR.044	Unique

Number of rooms: 110 Total programmed area 150 Total area 1542,08

3D Viewer model ready asmundkl@db2.nosyko.no/adac-aalesund-demo-ifc IFC: Local model (file):

Figure 23, dRofus Design vs. Program Comparison

The operator can select any item from the POR and the IFC viewer has the capability of finding the selected item in the model and zooming into the selected item.

The items generated in dRofus can be placed in the Revit model as unplaced rooms or family instances. All the properties defined in dRofus will be added to the Revit file as customized properties. To check the design versus the POR the Revit file should be synchronized with the IFC file that is generated by dRofus. The IFC file later can be opened in dRofus and the design errors can be reviewed.

Below are the main characteristics of dRofus.

- *IFC Support,* dRofus is the only system out of the reviewed systems that has adopted the open BIM practice by adapting BPie model view definition. (i.e. IFC integration) (Jernigan, 2008).
- *Design and POR Connection,* dRofus does not provide any design environment and by supporting IFC, gives the designer a wide range of computer applications to select from for the design phase.
- *Design Assessment,* There are too many steps in checking the POR requirements against the design. This complicated process can decrease the usability of the tool.
- *No Requirement Control in Design Mode,* dRofus does not provide design related features; on the other hand the provided Revit plugin does not perform any requirement control in the Revit environment. This workflow can lower the chance of spotting the design errors in real time by the designer.

- *No Visual Representation of the Design Errors,* The design errors will be marked up in the tables that are provided through the inbuilt IFC viewer, but there is no mark-up of the design errors provided in the 3D model.
- *IFC Format As The Native File,* Using IFC as the native file format is a bold decision since a text-based file is not the most efficient format for storing data. On the other hand it eliminates any potential inconsistency through data conversion.

3.2. LESSONS LEARNED FROM THE REVIEWED APPLICATIONS

All the reviewed applications provide a level of connection between a pre-defined version of POR and the early stages of design. Their limitations with respect to the goal of this research can be clustered into several criticisms.

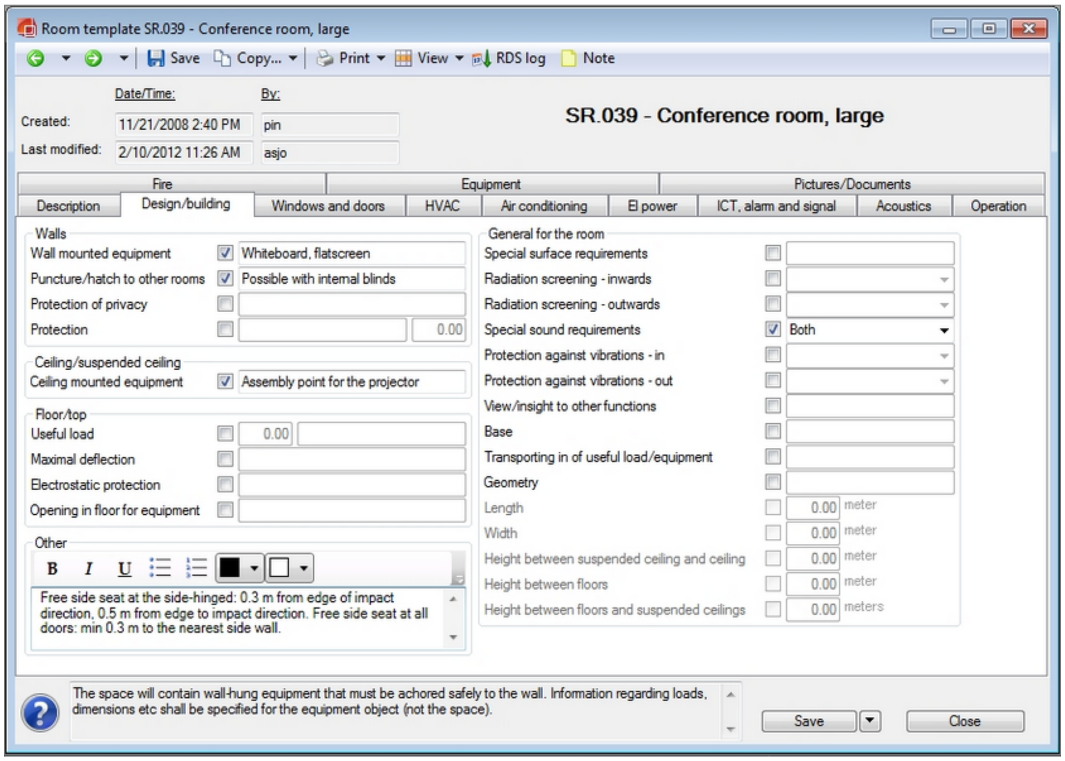
3.2.1. EXPERTS' VIEWS IN ARCHITECTURAL PROGRAMMING

The commercial applications reviewed in this chapter are not well-grounded in published expert advice for creating a POR. The software vendors do not provide any references or studies to support the effectiveness of their proposed model.

Trelligence Affinity provides a limited format for architects to add some requirements. The default template includes fields for area, equipment, and adjacency requirements. The user can also group the spaces into departments, but it will not impose any direct requirements over the design. Adjacency requirements can be added to the spaces as well. Any further requirements can be added to the space instances in the format of textual comments (About Trelligence, 2015).

Onuma System claims to create a connection between the POR and the design. In Onuma System there is no distinction between POR requirements and design decisions. Instead, the application provides two different views for the same data, textual and graphical. The textual view is based on a tabular format similar to what MS Excel provides, while in the graphical view the spaces are represented as two-dimensional shapes. Onuma System considers any modification of the data in the textual (Excel) view a programming decision, and any modification of data in the graphical view a design decision.

dRofus takes a different approach to the POR. It provides a list of requirements that can be added to a space. dRofus lets the user select which fields to fill and which fields to leave blank (About dRofus AS, 2015). Figure 26 shows dRofus user interface in requirement entry mode. The list of requirements differentiates dRofus from the other competition.



Room template SR.039 - Conference room, large

SR.039 - Conference room, large

	Date/Time:	By:
Created:	11/21/2008 2:40 PM	pin
Last modified:	2/10/2012 11:26 AM	asjo

Walls

Wall mounted equipment	✓	Whiteboard, flatscreen
Puncture/hatch to other rooms	✓	Possible with internal blinds
Protection of privacy		
Protection		0.00

Ceiling/suspended ceiling

Ceiling mounted equipment	✓	Assembly point for the projector

Floor/top

Useful load		0.00
Maximal deflection		
Electrostatic protection		
Opening in floor for equipment		

Other

Free side seat at the side-hinged: 0.3 m from edge of impact direction, 0.5 m from edge to impact direction. Free side seat at all doors: min 0.3 m to the nearest side wall.

General for the room

Special surface requirements		
Radiation screening - inwards		
Radiation screening - outwards		
Special sound requirements	✓	Both
Protection against vibrations - in		
Protection against vibrations - out		
View/insight to other functions		
Base		
Transporting in of useful load/equipment		
Geometry		
Length		0.00 meter
Width		0.00 meter
Height between suspended ceiling and ceiling		0.00 meter
Height between floors		0.00 meter
Height between floors and suspended ceilings		0.00 meters

The space will contain wall-hung equipment that must be achored safely to the wall. Information regarding loads, dimensions etc shall be specified for the equipment object (not the space).

Figure 24, dRofus Requirements Entry Interface

3.2.2. CONNECTIONS WITH IFC

Greater integration with the entire life cycle of a building can be achieved by extending the IFC data model by adding the missing fields to make it fully capable of capturing the programming requirements. This is a basic premise of this research,

Amongst the reviewed applications, dRofus is the only one that provides both import and export features for IFC format.

4. DATA MODELING OF THREE EXPERT METHODS FOR PROGRAMS OF REQUIREMENT

AIA defines a program of requirement (POR) as “An organized collection of the specific information about the client’s requirements which the architects need in order to design a particular facility.” (Palmer, 1981). There are different interpretations of this definition in Architectural, Engineering, Construction and Operation (AECO) industry (Hershberger, 1999). Exclusion of architectural programming information makes BIM data models fall short from the BIM core premise, which advocates the inclusion of the whole lifecycle of design information. “Can one comprehensive data structure support all the existing formats for architectural programming?” This is the main focus of this chapter.

4.1. METHODOLOGY

Among all the existing POR structures, the ones offered by William Peña (Peña & Parshall, 2001), Henry Sanoff (Sanoff, 1977) and Donna Duerk (Duerk, 1993) were chosen to represent a scaled collection of diverse POR formats. These scholars were chosen based on the following main two reasons:

- *Recognition in the industry and academia,* A short search on any scholastic database shows that the textbooks used in this research have been cited several times by other scholars also; they are some of the bestselling books on architectural programming.

- *Relevance to the research objectives,* As it was mentioned before, not all the subdomains of architectural programming were of interest to this research. Textbooks on data gathering techniques, while very useful to the programmers, would not provide this research with any useful material. The textbooks selected for this research all devote substantial portions to requirement types and classifications.

Some of the textbooks used in this chapter provide a straightforward model that includes programming data types and how they are classified. Some others define it in fragmented segments throughout the text.

I analysed the texts to extract a complete data model. I used resources from other authors wherever they were referenced to complete the data model. Each data model presented in this chapter represents a POR format of one of the authors, as compiled from the various sources.

4.2. SANOFF'S MODEL

Sanoff describes a program as a statement of intent. He believes a program is a collection of a desired set of events. Objective is a key concept in organizing a program. Based on Sanoff's methodology, the designer tries to achieve some objectives through the design and therefore, the programmer should help to set these objectives. Sanoff believes "Management by objectives" (MBO) helps the designer throughout the design. He describes a cyclical process where every cycle potentially consists of six phases.

- Setting objectives (or goals)
- Prioritizing objectives

- Mapping out a plan to achieve the goals (Activities)
- Setting a timeline to achieve the objectives
- Designing a measurement system
- Executing the plan

Sanoff introduces two more concepts besides “Objectives” to articulate this six-step cycle: “Activity” and “Performance Requirements”. Definitions of each of these three concepts are presented below in more detail and their relationships explained.

4.2.1. OBJECTIVE

Objectives are the backbone of this model. Sanoff looks at objectives in different scales and believes that objectives help an organization to define its “purposes of existence”. Objectives help the organization to:

- Define its “purpose of existence”
- Define areas of responsibility
- Evaluate accomplishments
- Clarify individual pursuits
- Make personal views visible

Sanoff defines a hierarchical model to derive more specific objectives from the high level goals.

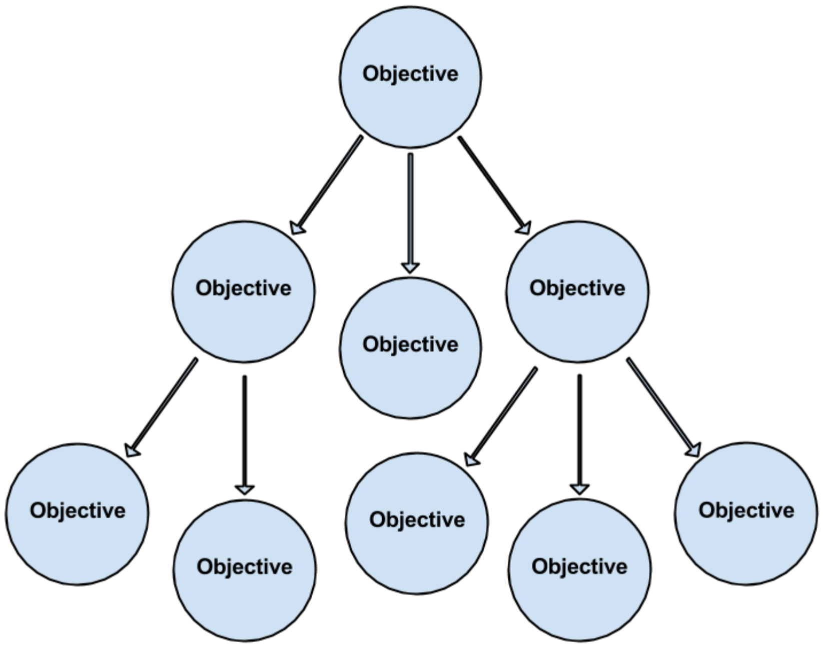


Figure 25, Sanoff's Hierarchy of Objectives

4.2.2. ACTIVITY

Activities describe how people utilize spaces. Objectives can be defined without any direct environmental indication. In the context of an architectural project, one accomplishes the objectives by modifying or creating built environments. Activities are the achievement plans. The concept of "Activities" is introduced to connect the objectives to a built environment.

Sanoff introduces the concept of activities as a path toward spatial requirements. He defines an activity as an occurrence in which there is a spatial connection between people and the environment. He suggests breaking down each activity into sub-activities and also drawing an interaction matrix to discover adjacency and grouping requirements.

Sanoff defines a requirement as a quantifiable statement about a proposed behaviour. A requirement can be based on fact, intuition, or assumption. Sanoff proposes a seven-step procedure regarding activities.

- Identify the activities for which the artifact is being designed.
- Identify secondary or derived activities.
- Determine the requirements for each activity (based on fact, intuitive fact, or assumption).
- Determine dimensional and area standards (spaces).
- Analyse activities and predict alternate tentative optimal arrangement.
- Determine and rank activity relationships (adjacencies).
- Synthesize the analysis.

Activities describe human behaviour whereas spaces describe the physical environment. In that sense, forming the program based on activities is more humane. One activity may reside in multiple spaces whereas one single space may host multiple activities.

4.2.3. PERFORMANCE REQUIREMENTS

Sanoff introduces the concept of performance as a bridge between requirements and the built environment assessment. Performance is a procedure to calculate the attributes of the physical environment to compare with the requirements imposed in the previous steps. Performance answers to the users' needs. A statement of needs is derived from the following questions.

- What usage or function is being considered?
- For whom is the requirement imposed?
- Why is there a need?
- Where will the needs exist?
- When will the needs exist and for how long?

After developing a performance requirement, its performance criteria should be defined. Performance criteria are characteristics that will be used in evaluating whether the requirement is satisfied or not. Along with the criteria, the evaluation techniques should be defined. Some evaluation techniques are based on measuring physical attributes while some require more sophisticated techniques such as computer simulation. In other cases the judgement of the experts may be the only possible evaluation technique.

Sanoff does not further describe the three proposed evaluation techniques. A numerical requirement would impose a numerical range on an attribute from the project. Area requirements are the most obvious examples. Theoretically, one can impose a numerical requirement on any of numerical attributes in a project or assets that can be presented through numbers (e.g. colours), but Sanoff does not provide a list of those attributes. One can rely on the BIM data models to cover that shortage. Any numerical property in a BIM model can be targeted by requirements and limitations can be imposed.

At the other end of the spectrum are the requirements that need a panel of experts for evaluation. These requirements can be subjective in nature. The most important condition that they should meet is clarity and readability. The architectural programmer can communicate directly with the panel of experts to further explain the imposed requirement.

Simulation requirements are neither direct numerical comparisons nor based on subjective judgement. They use numeric attributes of a design or a built project and through computation evaluate capabilities and characteristics of the system. Simulation can be used to predict thermal behaviour of a building, walking travelling time between two rooms, and waiting time in a clinic at a specific time and date.

Simulation algorithms should be developed for each requirement specifically. Each algorithm will rely on a set of attributes that should be supported by the BIM data models in order to evaluate a BIM model.

4.2.4. SUMMARY OF SANOFF MODEL

The first step in converting Sanoff's ideas to UML diagrams is to identify and name the concepts.

- A program consists of three interrelated systems. ***Objectives***, ***Activities***, and ***Requirement.***
- Objectives can be broken into ***Sub-Objectives*** and a ***priority*** value can be assigned to them.
- Each objective should start with "***To***" and should have a single ***result*** to accomplish.
- To accomplish the result, a ***timeline*** should be added to an objective.

- ***Activities*** connect objectives to the built environment by connecting a physical process to an objective.
- Activities must be situated in ***spaces***
- Activities can be broken down into ***sub-activities.***
- Activities describe how ***people*** would utilize spaces.
- Several activities may reside in one physical space or a single activity may occupy a group of spaces.
- A requirement can be based on ***fact***, ***intuition***, or ***assumption***.
- Requirements derive from the users' needs.
- ***Needs*** are derived from the following questions.
    - What use or ***function*** is being considered?
    - For ***whom*** is the requirement posed?
    - ***Why*** is there a need?
    - ***Where*** will the needs exist?
    - ***When*** will the needs exist and for how long?
- Evaluation techniques can rely on ***numerical*** techniques, ***simulation*** techniques, or a ***panel of experts.***

From these bullet points I have identified five major components (classes) in Sanoff's model: Objectives, Activities, Needs, Requirements, and Performance Criteria. Figure 28 shows how these five components are connected to each other.

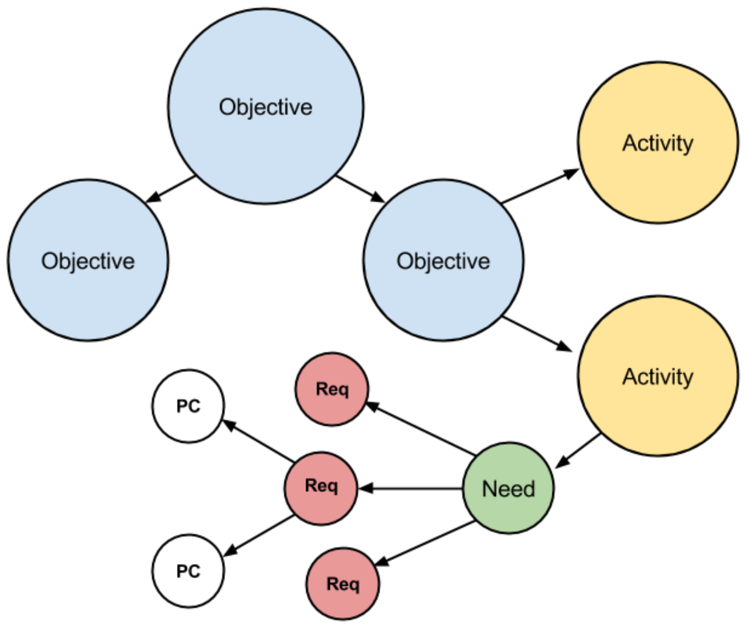


Figure 26, POR Components in Sanoff Model

Figure 29 is a UML diagram that shows the five main classes along with their attributes and relationships. Every instance of the Objective class has references to other instances of the same type, defining them as its children and also a reference to a parent instance of the same class. This pattern is known as Composite pattern in data modeling and software engineering (Vlissides, Helm, Johnson, & Gamma, 1995) Composite pattern allows instances of an object to form a tree shape diagram. This pattern is well suited for our model where more detailed statements are extracted from the more generic ones.

Performance Criteria class has three sub-classes based on the evaluation method. The Simulation and Numerical Criteria classes both can have references to respective properties in BIM data models whereas Panel of Expert may refer to an object instead of a specific property. Numerical Criteria will check the quantity of the respective field by performing basic algebraic formulas. Simulation instances will treat the values as raw data to feed them to appropriate algorithms and evaluate the performance of the BIM model based on the outcomes.

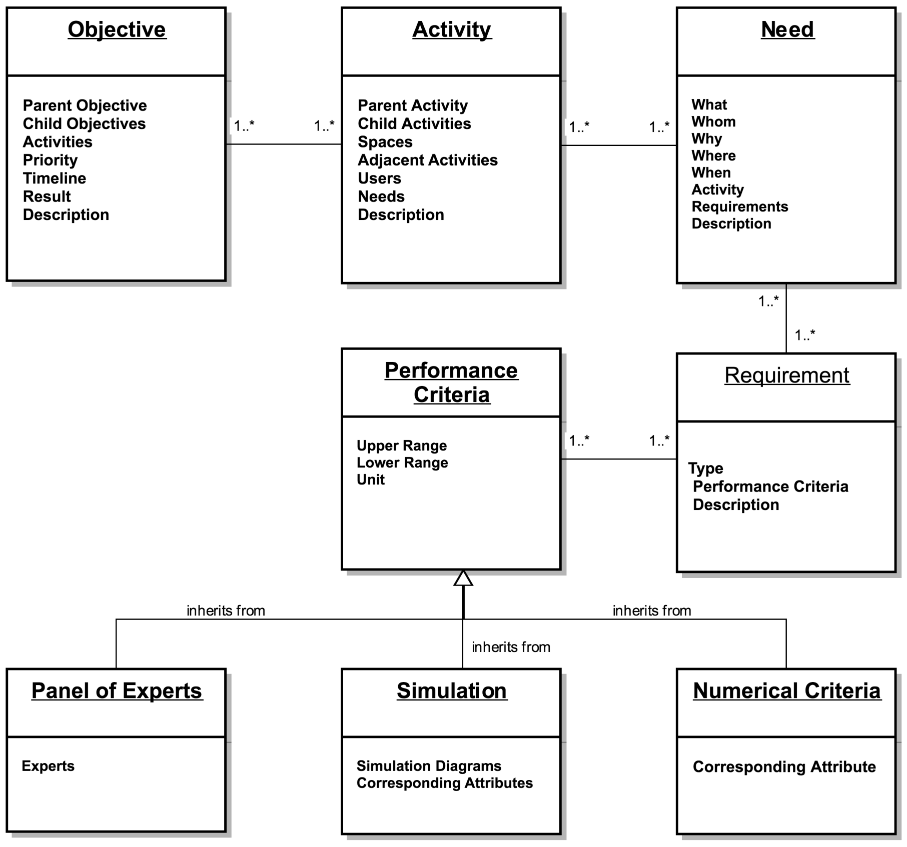


Figure 27, Sanoff's POR Model, UML Class Diagram

This diagram is not an elaborated UML diagram hence cannot be used as a source for implementation. I compared and combine this diagram with similar diagrams generated by analyzing other POR models.

4.3. PEÑA'S MODEL

Peña considers programming the architect's first and often most important task. Peña's model is simple and straightforward, yet yields tangible results. That is probably why this model is one of the most widely used models among practitioners (Cherry, 1999). Peña defines several key concepts:

- **Goals**: What does the client want to achieve and **why**?
- **Facts**: What do I know? What is given?
- **Concepts**: **How** does the client want to achieve the goals?
- **Needs**: How much **money** and **space**? What level of **quality**?
- **Problem**: What are the significant **conditions** affecting the design of the building? What are the general **directions** the designer should take?

Peña defines four main areas, or as he calls them, design determinants, to structure the types of information needed to define a comprehensive architectural program: **Function**, **Form, Economy,** and **Time.**

The design determinants are applied to each of the five steps leaving the programmer with a five by four table.

Peña believes only practical goals that can be achieved through concepts should be part of the program. Facts should be related to goals; otherwise, they bring no value to the program. He breaks down each of the determinants into smaller parts.

- **Function** implies "what's going to happen in this building." It concerns **activities**, **relationship** of **spaces**, and **people**-their **number** and **characteristics**. Key words are **people, activities,** and **relationships.**
- **Form** relates to the **site**, the **physical environment** (psychological, too) and the **quality of space** and **construction.** Form is what will be **seen** and **felt**. It is "what is there **now**" and "what will be there." Key words are: **Site**, **Environment** and **Quality**.
- **Economy** concerns the **initial budget** and quality of construction, but also may include consideration of operating and life cycle costs. Key words are: Initial **Budget**, **Operating Cost, and Life Cycle Costs.**
- **Time** has three classifications-**past, present and future**-which deals with the influences of history, the inevitability of changes from the present and projections into the future. Key words are: **Past, Present, and Future.**

Peña's model is straightforward and can be represented through a four by five table. The model is more complicated when it comes to converting to an object-oriented model. While it is tempting to take the rows and the column of the table as the base classes, they do not represent distinct behaviors, which is critical in separating classes from each other. One can use similar classes to represent a goal and a fact since they are both verbal statements; the difference is they represent two different time spans, one is about the existing state and the other one is about the desired state. One can use the same data structure to represent both of them and set a flag as “desired” or “existing.” Concepts and goals both represent qualitative statements about the desired state. Concepts are generated when goals are passed through the filter of facts and can imply spatial solutions. Concepts can be more specific than goals and can be expressed through textual or graphical statements whereas goals are usually textual statements. Concepts and goals can be described through similar data structures. A closer look at Peña’s model shows that the rows and column are more appropriate to be considered as the properties of the classes than being the classes themselves.

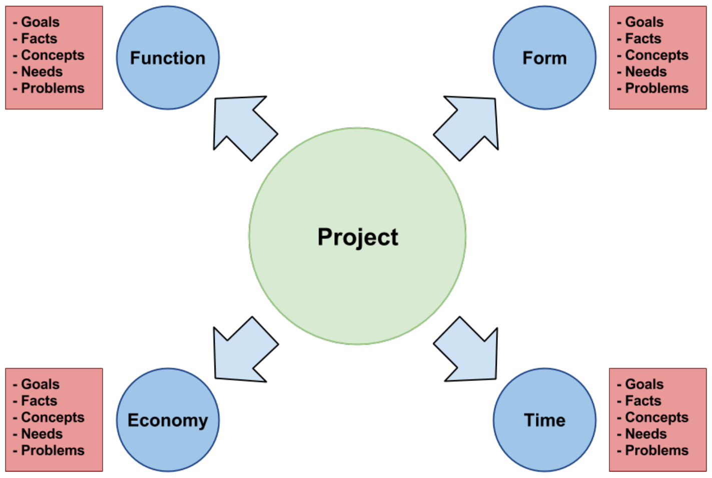


Figure 28, Peña's Scopes and Determinants

To identify the base classes, I took a closer look at the information index and the categories that Peña defines for each determinant. Peña breaks down each determinant into three parts:

- Function:
    - People
    - Activities
    - Relationships
- Form
    - Site
    - Environment
    - Quality
- Economy
    - Initial Budget
    - Operating Cost
    - Life Cycle Costs
- Time
    - Past
    - Present
    - Future

Looking at all the twelve sub-categories, I could distinguish three distinct classes: People, Activities, and Environment. Relationship, as the name describes, is to explain how classes are connected and cannot be taken as an independent class. Site is a special instance of environment and Quality can be represented through the attributes. All the sub-categories of Economy determinant are different aspects of “Cost” which can be applied as an attribute to all the base classes. Time can also be applied to the base classes as an attribute.

4.3.1. ENVIRONMENT

Environment class represents any form of a built or natural environment that the project is related to, or contains regardless of form and scale. This would include the project site, neighboring buildings, existing facilities on site, roads, proposed departments, rooms and etc. The main attributes to represent an environment are: Name, Location, Number Geometry, Adjacent environment, Cost, Growth, Parent environment, Children environment.

4.3.2. PEOPLE

People class represents all the people that would have an influence on the project. That includes Users, Clients, Society, and etc. The main attributes to represent a group of people are: Number, Related People, Growth, Physical Characteristics, Intellectual Characteristics, and Emotional Characteristics.

4.3.3. ACTIVITY

People are connected to the environments through activities. Activities represent how and why people would interact with the environment. The main attributes for activities are: Privacy level, Users, Environment, Growth, Potential activities to mix with, Segregation (other activities.)

4.3.4. CONNECTING THE BASE MODELS TO PEÑA'S TABLE

The table's row can be abstracted into three base classes. Each of the three base classes should represent these five columns of Goals, Facts, Concepts, Needs and Problems. A closer look shows that the difference between the five columns is not in the different attributes that they represent but rather in the scope of those attributes. While Facts represent the current state, Goals, Concepts and Needs represent the desired state. Instances of the Environment class can be used to represent the existing structures on the site as well as a required room proposed by the architectural programmer. In the former case, the Geometry attribute represents Facts while in the latter it represents the Needs.

The attributes of the base classes should be capable of representing all the five scopes defined by Peña. To reach that goal I defined a special class that I call Peña Data. All of our attributes will be Peña Data instances.

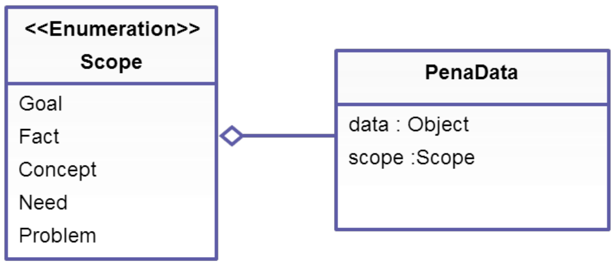


Figure 29, Peña Data Class

As you can see, this class has two fields, an Object holding the value of the attribute and an enumerated field, of the Scope type, representing the scope of the attribute. When architectural programmers assign a value to any of the attributes in the three base classes, besides the value, they should also assign the scope of the attribute as well. Each attribute can hold a list of paired values and scopes; for example, an existing room can be represented by an instance of the Environment class. The area attribute can have two values, the first one representing the current area (scope: Fact) and the second one representing the required area (scope: Need). Figure 32 is the UML diagram representing Peña's model.

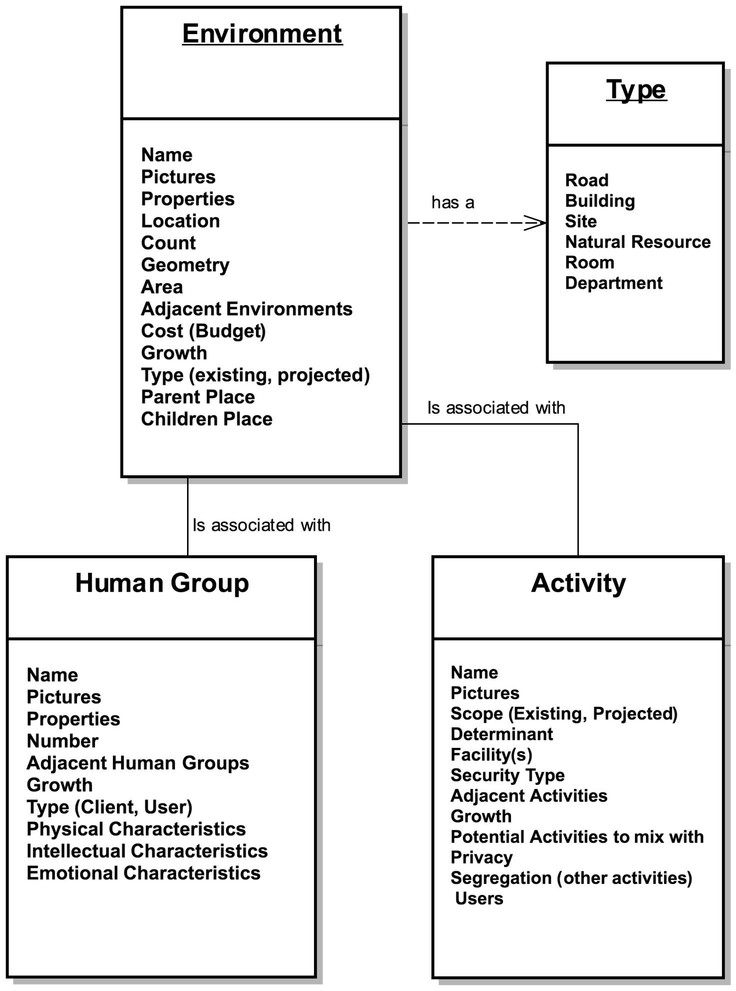


Figure 30, Peña's POR Model, UML Class Diagram

4.4. DUERK'S MODEL

Duerk's model is the most recent model out of the three models. She sets the main goal of her book "Clear Structure for the Process." When it comes to the programming process she defines it as "A rigorous set of procedures." Given the nature of this study, such a goal made it easier to extract the base classes and their relationships.

Duerk defines architectural programming as “A systematic method of inquiry that delineates the context within which the designing must be done. It also defines the requirements that a successful project must meet. (Duerk, 1993, p. 8)” She considers programming the first part of the design process and holds the POR responsible for fulfilling the future inhabitants’ dreams, hopes, wishes and desires. Duerk divides programming into two main phases:

- Analysis of the existing state.
- Projection of what the future state should be.

When it comes to the relationship of architectural programming and design, she considers programming the first part of design and calls it “the basis of good design.”

4.4.1. ANALYSIS OF THE EXISTING STATE

Existing state is the context of the project and includes all the information about site analysis, user profiles, codes, constraints and climate.

Duerk uses a concept of design issues as the main category for exploring the existing state. She believes by using this method one can avoid collecting more information than one knows how to apply or as she calls it “merely interesting facts.”

4.4.2. PEOJECTION OF THE FUTURE STATE

After discovering the existing state, the architectural programmer can make propositions about the future state. Duerk advocated a four-level hierarchical model for making decisions about the future state. The four levels are mission statement, project goals, performance requirements and conceptual relations.

4.4.3. ISSUES

Duerk defines an issue as “Any matter, concern, question, topic, proposition, or situation that demands a design response in order for the project to be successful. (Duerk, 1993, p. 24)” She believes there are generic issues (e.g. circulation, safety, territoriality, privacy, image, and flexibility) that apply to every architectural project; however the priorities may be different. Duerk considers Form, Function, Economy and Time in Peña’s model design issues. Some of the issues can be broken into separate parts. For example, circulation, as one of the generic issues, can be divided into pedestrian circulation and vehicle circulation. While both need to be addressed in the design, they require separate answers. This is the list of Duerk’s generic issues and their sub-issues.

- Audibility
  - Behaviour Settings
- Circulation
  - Information
  - Material
  - Parking
  - Pedestrians
  - Vehicles
- Comfort
  - Physical
  - Psychological
- Convenience
- Durability
- Economy
  - Elegant means
  - Phasing
  - Quality
- Energy Efficiency
- Environmental
  - Impact
- Flexibility

    - Adaptability
    - Choice/Variety
    - Expansion/Contraction
    - Multi-use
- Image
    - Identity
    - Message
    - Ordering/proportion
    - Status/hierarchy
    - Symbolism
- Integration
    - Group participation
    - Social
- Legibility
    - Layering
    - Orientation
    - Plan recognition
    - Sequence
- Maintenance
- Mood/Ambience
    - Attitude
    - Emotional response
    - Spirit of place
- Olfactory
- Personalization
    - Group
    - Individual
- Privacy
    - Group
    - Individual
- Resource
    - Management
- Safety
    - Accidents
    - Hazards
- Security
    - Assault
    - Robbery
    - Unauthorized access/entry
    - Vandalism
- Territory
    - Group
    - Individual

- Visibility

4.4.4. FACTS

Duerk defines facts as the objective information about the site, climate, context, and other verifiable measurements. Some of the facts may raise an issue and require a design answer. Besides the facts that are related to specific issues, there are some facts that are related to the generic issues and thus should be collected in any project.

Other facts may gain significance from the project specific issues. Those facts are collected based on the specific project issues. Figure 33 shows how project specific facts and issues are related.

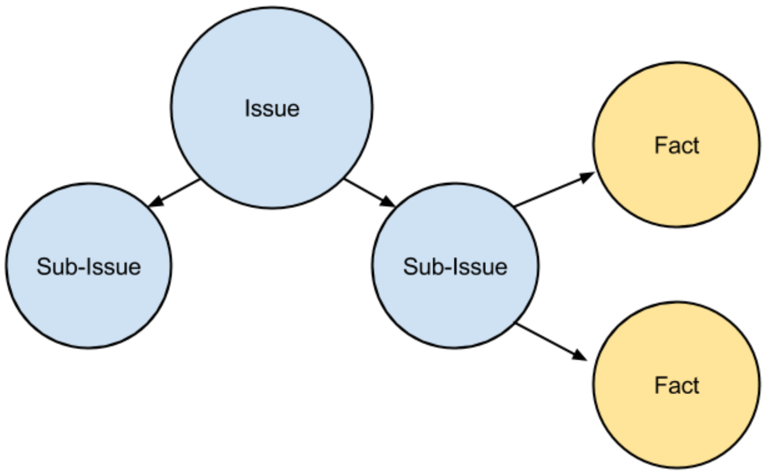


Figure 31, Connection Between Issues and Facts

Duerk proposes a list of generic facts that should be collected for every project:

- Site
    - Climate
        - Degree days
        - Precipitation
        - Solar exposure

        - Wind speed and direction
    - Codes
        - Building
        - Zoning
    - Site condition
        - City services/transit
        - Geology
        - Hydrology
        - Noise
        - Odours
        - Site features
        - Soil bearing capacity
        - Topography
        - Utilities
        - Views to and from site
    - Traffic levels
        - Bicycles
        - Pedestrians
        - Vehicles
- Person/User
    - Activity analysis
    - Age group
    - Anthropometrics
    - Disability
    - Environmental history
    - Number of people/groupings
        - Organizational structures
    - Perceptual abilities
    - Personality
    - Roles
    - Rules
    - Values
- Context
    - Cultural
    - Demographic
    - Economical
    - Ethical
    - Ethnic
    - Historical
    - Political
    - Social

Duerk divides generic facts into three main categories: Site, Person/User, and Context.

4.4.5. SOLUTIONS

Any potential design alternative in response to an issue is a solution. In Duerk's hierarchical solution model the mission statement is the root of all the solutions and at the top level the solutions can take physical forms. Solutions can target different aspects of design. Some solutions can affect the used materials while others would affect lighting or form. Bigger issues may lead to solutions that affect more than one design area and as they become more specific they may focus only on one area. Duerk mentions the following main categories for solutions' design areas.

- Composition Strategies
- Equipment
- Form
  - Colour
  - Dimension
  - Direction

- Lighting
- Material
  - Building
  - Interior
  - Landscape
  - Texture
  - Transparency
- Orientation
- Space
  - Definition
  - Enclosed
  - Open

4.4.6. VALUES AND GOALS

In Duerk's model values are where the people involved in a project including clients, users, and authorities can have an impact on the project. Different people have different values and require different solutions. To reflect the values more systematically, Duerk proposes a method in which issues turn into goals by going through the filter of values. Based on this model the project goals are directly or indirectly connected to an issue/value pair. The indirect route goes through parent goals. Duerk defines a goal as "A concise statement of the designer's promise to the client about the quality of the design in relationship to a particular issue. (Duerk, 1993, p. 27)"

4.4.7. HIERARCHY OF GOALS

To manage the goals in a project, Duerk proposes a hierarchical model where each goal has a parent goal and can be broken down into smaller and more detailed goals (children goals.) The hierarchy starts with a "Mission Statement". Mission statement is "A statement that concisely expresses the reason a client undertakes a project in the first place. (Duerk, 1993, p. 36) "The full hierarchy of goals is shown in Figure 34. "Goals" are the direct children of the mission statement in the hierarchy and each goal is associated with a design issue. Duerk also advocates associating each goal with a priority. Vague and open to interpretation words should be avoided in defining goals. Goals can also contradict or overlap each other. For example, privacy and security can be seen together as two associated goals whereas they may contradict visibility, which may be one of the goals of the project. Here is a pattern that Duerk (Duerk, 1993, p. 46) proposes for defining goals:

"The project SHOULD (verb of intention or "being" such as "promote," "project," "encourage" goes here) (an adjective or descriptive phrase goes here to define the QUALITY desired) (a noun goes here to focus on the area of concern).

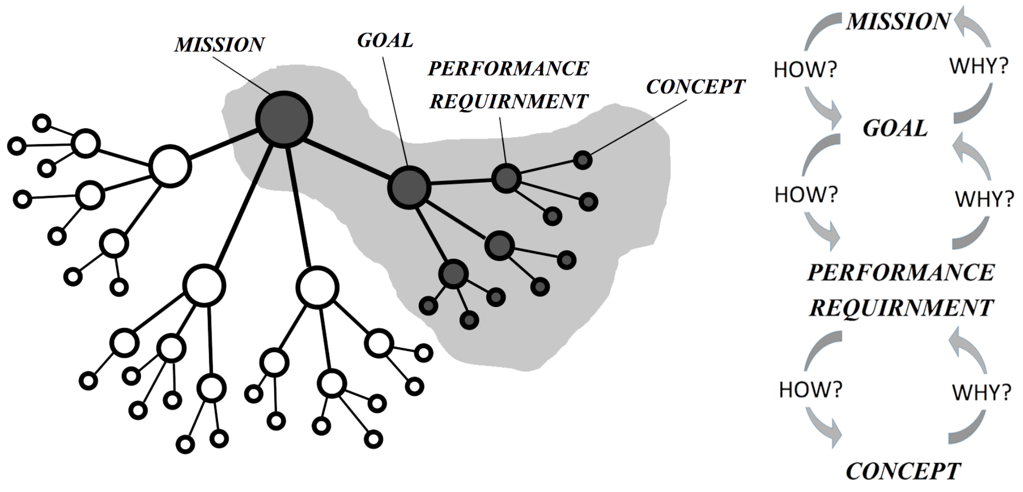


Figure 32, Duerk's Hierarchy of Goals. Reprinted from (DUERK, 1993, P. 9).

Besides the hierarchy, Duerk defines different types for goals as well. She defines four main types of goals:

- Process and resource goals
- Educational goals
- Personal goals
- Project specific goals

4.4.8. PERFORMANCE REQUIREMENT

Duerk defines a Performance Requirement (PR) as “A statement about the measurable level of function that a designed object must provide for a goal to be met.” (Duerk, 1993, p. 48)Performance Requirements are associated with the goals and guarantee that the qualities defined by the goals will be met in the project. Each goal will result in several PRs and each PR should be associated with one sub-issue. Figure 34 shows the relationship between Mission, Goal, Performance Requirement and Concept. Duerk declares three characteristics for a PR; it should be: specific, measurable and operational.

In describing “Measurable” Duerk defines three types of measures: binary, scalar, and judgment. Binary measures can only take two states: satisfied, or unsatisfied. Binary measures are used for the Performance Requirements that impose a clear level of performance to the project. Scalar measures define an acceptable range of answers using a physical measurement (inch, cubic feet per minute, dollars, BTUs, humidity, decibels, foot candles, etc.) Performance Requirements regarding costs usually use scalar measures to define an acceptable monetary range. Judgment is an evaluation based upon priorities and values. Duerk believes that where there is no reliable scalar or binary measure, judgment should take over. Judgmental measurements are based upon personal values and can be delegated to experts.

4.4.9. CONCEPTS

Concepts are the last piece of Duerk’s model. Unlike all the previous parts, “concepts are graphical sets of relationships among several of the elements under an architect's control such as form" (Duerk, 1993, p. 60). Concepts can be big enough to encompass the entire project, or they can illustrate the ideal solution for a small part of the project.

Concepts can address the project in different scales. If a concept is developed against the whole project, it's called mega-concept, theme, or parti. Concepts can also address goals and performance requirements.

Duerk mentions these characteristics for concepts

- A concept answers a question
- It can be diagrammed simply
- It should have a brief caption
- It should be the most generic answer to the problem

4.4.10. SUMMARY OF DUERK'S MODEL

Although Duerk's model presents a more clear structure for a POR, it is rather challenging to propose an object-oriented system to capture all the data. Some of the bold topics such as "Facts" and "Issues" could be the base classes, but one can also consider "User", "Context" and "Site" as base classes and treat Facts as their attributes or a secondary class that its instances can be part of the base classes. Issues can also be regarded as a base class or as required attribute for the "Goals".

All the entities of the solution hierarchy, which includes Mission, Goal, Performance Requirement, and Concept, can be represented through the same class where each instance has a part-of-a relationship with higher scale instances. For example several instances of the solution class representing "goals" can have part-of-a relationships with an instance that represents the “mission”. By following this pattern one can create a tree like structure representing all the solutions.

In analyzing the existing state, Duerk introduced Site, Person/User, and Context as the major categories. One can consider Site and Context the instances of the same class but in different scales. While site refers to the part of surroundings that is acquired by the owners and dedicated to the project, context has a wider range and includes the society and the urban or rural texture that the project interacts with. Similar to the solutions hierarchy, one can represent Context and Site by using the same class where Site has a part-of-a relationship with the Context.

Person/User is the third part of Duerk’s triangle of the existing state. One can dedicate a class to represent Users, but Duerk also talks about the Client as a source for facts when she introduces the concept of Values. One can represent User and Client through instances of the same class as they represent similar concepts. On a slightly different approach, a superclass representing the common attributes of User and Client can be devised. The superclass is extended by classes representing User and Client where they can add specific attributes to represent User and Client.

Based on the conversation above, I devised a UML diagram capable of representing Duerk’s model with the following classes.

4.4.10.1. USER

This class represents the needs, attributes, values and grouping of the users. In that sense, this class represents not only facts, but also values. Each instance of this class can contain sub-categories of users within itself. For example in designing a library,

library visitors can be described through an instance of the “User” class, but the class can also contain smaller groups that are defined based on their age such as “Children” and “Adults” where their specific characteristics will be described.

4.4.10.2. CLIENT

Client class represents similar facts in comparison with the User class, but they each have exclusive attributes. Client class has exclusive attributes that represent the administrative and organizational role of the individuals who are in charge of the project. In some cases the client and the user can be the same.

4.4.10.3. SITE

This class represents information about the piece of land that is dedicated to the project. The information represented by this class can be divided into three categories, climate, code, and site conditions.

4.4.10.4. CONTEXT

This class represents similar information to the Site class, but on a larger scale. Context also represents social and demographical information about the human context where the project is located. Other information represented by this class includes political and economical facts about the context of the project.

4.4.10.5. SOLUTION

All the elements representing the future state can be represented through one class with a hierarchical order where instances can form part-of-a relationship with each other. Solutions are the result of passing the issues through the filter of values; therefore each solution is related to one or more values. Each solution also responds to one or several issues; therefore “Issue” is a key attribute of the solution class. Each solution can be broken down into smaller solutions, which have part-of-a relationship with the original solution. Some solutions can be measurable (performance requirements). Figure 35 is the simplified UML model representing Duerk’s model

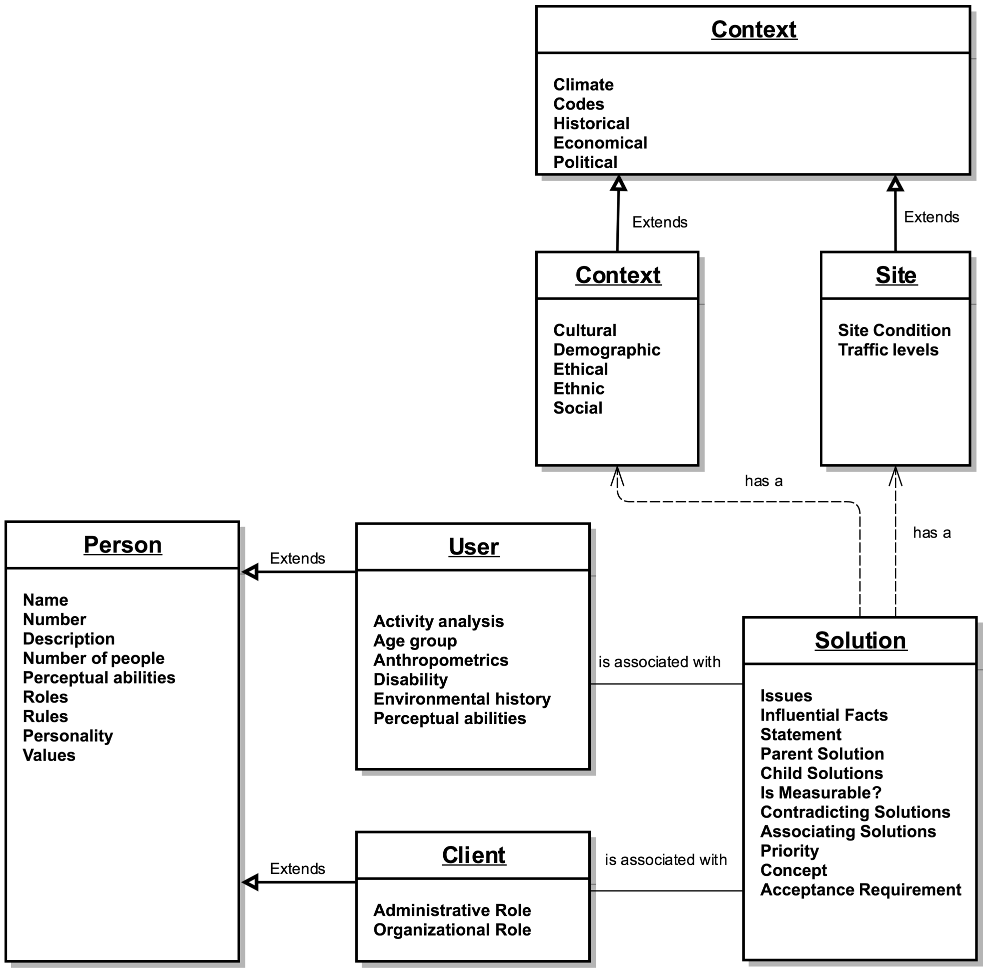


Figure 33, Duerk's POR Model, UML Class Diagram

5. A DATA MODEL FOR A UNIFORM FORMAT FOR PROGRAM OF REQUIREMENTS

UFPOR encompasses the three models that were analyzed in the previous chapter. I started with the similarities shared by all the three models to form a solid foundation, then incorporated the unique characteristics of each model through adding flexible and generic parts to the model.

One can discern a lot of similarities in Peña's, Sanoff's and Duerk's models. They all provide a path from present to future as an inherent process of design. They propose techniques to study the clients through their values, goals, and financial budget to guide them toward their mission.

They all also incorporate similar topics. "Goal" and its synonyms are ubiquitous in all models. All models decompose the big goals into attainable and measurable requirements. They emphasize "People" and try to incorporate descriptions of people. Users, clients, society and owners are all human beings with goals, values and desires that can contradict each other. All three models propose different ways to discover different groups of people who are influential in a project. They include "Environments," whether existing or proposed for future, whether as small as a room or as big as a city. They all advocate studying the existing environments such as the city in which one is proposing a new project, the neighborhood, project site, and existing structures in the site to understand what one should add to this collection. They also incorporate "Activities" as the interaction between people and environments. One cannot decide what our buildings should be and what they should not be if one does not know how people would use the place and interact with its components. All the models also incorporate a hierarchy of goals, objectives and requirements. These are different ways to describe what the project or a specific part of a project “Need”s to succeed. An area requirement is what a room needs to operate successfully and a “Mission Statement” is what the whole project needs to succeed.

5.1. UFPOR BASE CLASSES

Five primary classes were identified as parts of all the three models that were studied. They may be referred to with different names, but all three major models in the previous chapter share the same concepts.

5.1.1. ENVIRONMENT

A new environment is usually the final goal of an architectural project. Environments, in their most generic sense, are the main topic in a POR. I define environment as an entity that can host and contain people or other environments. By this definition a city is an environment because it can host buildings (other environments) and the people who live in that city. A bathroom or a walk-in closet is also an environment because it can temporarily host people. This definition is scale independent. It is also not limited to the environments that are built by human beings.

5.1.2. PEOPLE

People are the reason that we care for places. People can take different roles in a project. Users, clients, society, and authorities are all groups of people with values, requests and powers. While they have different roles, they share a lot of attributes. Similar to places, people can be broken into categories as well. “Users” of a school can be divided into students, teachers, and the administrative staff.

5.1.3. ACTIVITY

Activities define the connection between the people and the places. Activities are the reason why people make places and they are the main topic in a POR. One studies people to know what they want to do (activities) and then defines places where those activities can happen. Similar to the places and people, activities can be broken down into sub-activities. For example “Preparing food” as an activity can be broken into “Storing Ingredients,” “Washing Dishes,” “Cooking” and some more activities. As you can see activities are usually associated with a verb.

5.1.4. NEED

One of the main purposes behind a POR is to ensure the clients that the final project will match their expectations and fulfill their needs. “Need” is a special class in UFPOR. Needs create a web where all parts of a POR get connected to each other. One starts with people and by discovering their needs proceeds to activities, then by discovering activities’ needs one discovers Environments. Detailed and quantified space requirements are attached to environments as their needs. From the data-modeling standpoint, a project mission (the largest need of a project) and a required area assigned to a room are both needs, even though they are at the two ends of a tree structure. While large needs are hard to measure, it is very easy to measure the success of smaller and quantified needs. One can analyze and evaluate large goals that are hard to measure to produce smaller and more measurable goals. PORs break down large and qualitative goals into smaller quantitative goals that are easy to measure, and then one can evaluate the success of answering the large needs based on the success of their children needs.

5.1.5. ASSESSMENT

Needs may project their fulfillment assessment to their sub-needs either directly or by proposing Activities or Environments and defining their needs. All the needs together will form a tree-like structure of connected needs. Fulfillment of each need should be either directly or indirectly assessed. Direct assessment happens through execution of the "Assessment" attached to the need, and indirect assessment happens through evaluating the fulfillment of the child needs. The assessment class can contain the following attributes.

- *Type,* The assessment can be based on Nominal, Ordinal, Interval or Ratio.
- *Attribute under Assessment,* The attribute that is being measured
- *Acceptance Criteria,* Based on the assessment type, acceptance criteria should be defined. It can be a numerical range for Interval and Ratio types. Nominal and Ordinal assessments are based on human judgment and an acceptance criteria should be defined; for example it can be approval from both the client and users.
- *Assessor,* For Nominal and Ordinal assessments involved "People" should be mentioned. For Interval and Ratio assessments, an "Assessor"

implementation that defines the mathematical equations used in the assessment should be referenced.

- *Result,* The assessment will produce several types of results based on the assessment type and the result will be assigned to this attribute.

5.2. UFPOR KEY CONCEPTS

To further define the base classes and identify their sub-classes and attributes, there are some key concepts to keep in mind. Knowing these key concepts helps in defining the subclasses and their attributes.

5.2.1. TIME

Looking at each of the base classes of UFPOR through the filter of time helps to identify all the sub-classes that need to be extended for every base class. “User” is a sub-class of “People” in the future, while “Client/Owner” is more related to the present time. When it comes to “Environment”, it should be studied in past, present, and future. While the existing structures represent the past, neighbors and the site belong to the present time. The proposed spaces and departments, which are the result of architectural programming phase, represent the future.

5.2.2. COMPOSITION

Composition is a design pattern in object-oriented design where instances of a certain type can hold a reference to a list of objects of the same type and form a tree structure representing the whole system (Vlissides et al., 1995). By using the composition pattern one can theoretically model the physical aspects of a building by using only one class (Environment.) Rooms are defined with part-of-a relationship with the departments and departments hold a part-of-a relationship with the building. The same pattern can be used to create activities and sub activities as well as creating a hierarchical model for needs.

Figure 36 shows the relationship between the three main base classes and the main concepts of UFPOR. Studying people's needs results in discovering activities and analyzing activities and understanding what each activity needs helps to understand the environment and also how to assess the environment. One can also study the base classes and their relationships in different scales and times to understand the existing situation and propose changes for the future.

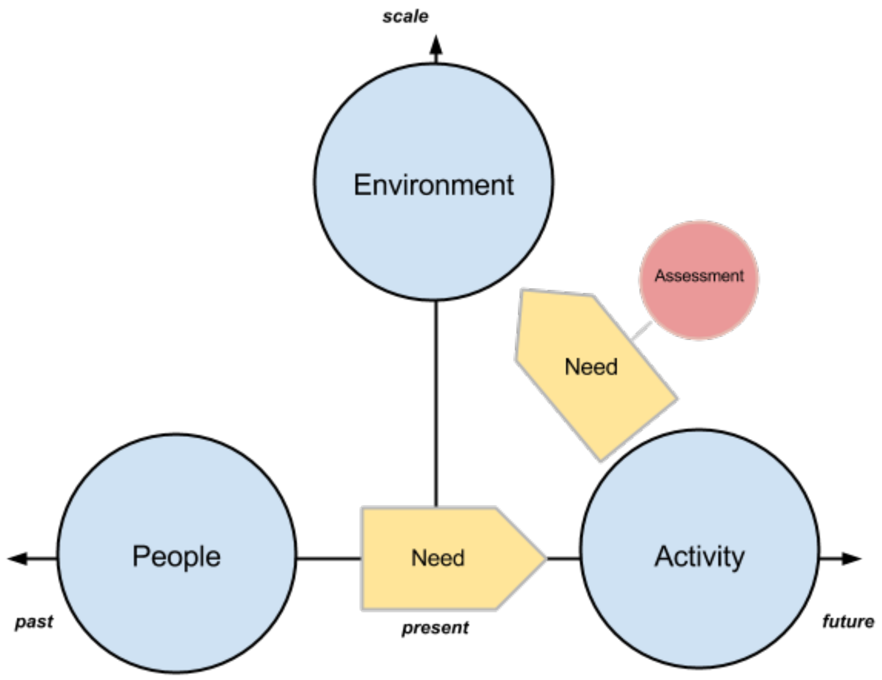


Figure 34, UFPOR Main Classes and Their Relationships

5.3. UFPOR UML MODEL

Figure 37 is s UML diagram representing the main base classes and their relationships. To simplify the model, class methods are not added to the diagram. The attributes are added as examples to demonstrate the range of data that can be captured by each class. This data model can be used as a blueprint to create comprehensive data models supporting a wide range of POR structures. Additional attributes can be added to each class based on the POR structure. The simple lines show Association Relationship between classes. The number or the range of numbers shows how many instances of a class can be associated with the other class. A class with association relationship with itself uses the composition pattern (also called Reflexive Association). The empty diamond arrow represents Aggregation Relationship. Aggregation Relationship indicates one class is part of another class with the condition that the aggregated class can outlive the container class. In this diagram the Geometry is part of the Environment. The filled diamond arrow shows Composition Aggregation relationship the difference between Composition Aggregation and Aggregation is that in Composition Aggregation the aggregated class cannot outlive the container.

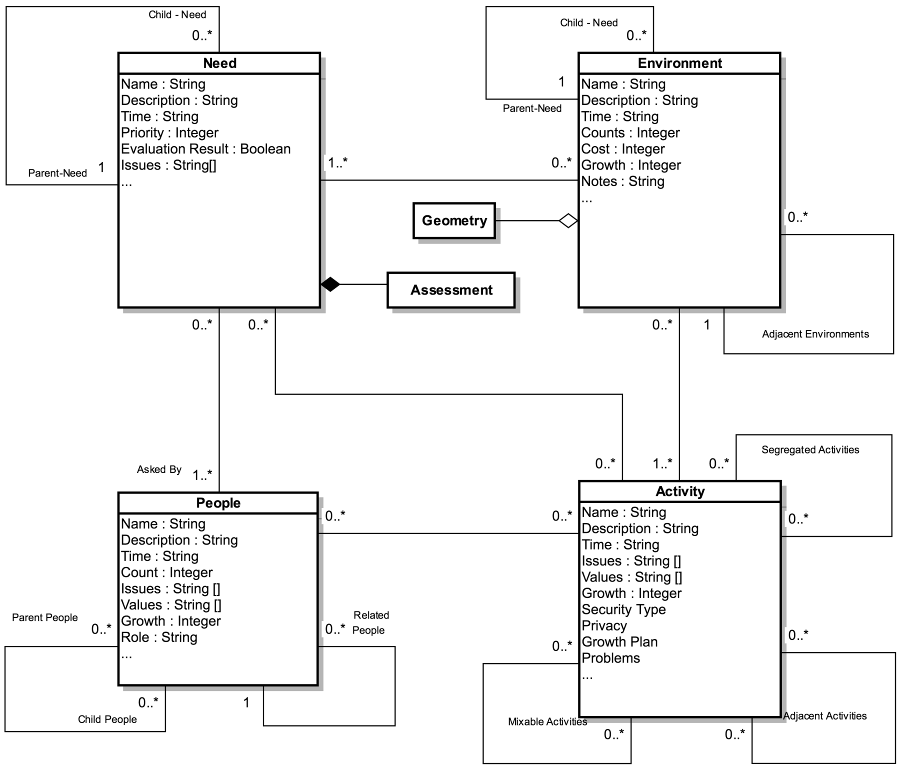


Figure 35, UFPOR UML Model

5.4. MODELLING THE THREE EXPERTS MODELS WITH UFPOR

5.4.1. SANOFF'S MODEL

Sanoff proposes a procedural process where studying people leads to discovering objectives, and studying objectives leads to discovering activities and studying activities leads to a tree structure of what the users need in order to perform their activities. Sanoff advocates tree structures in organizing objectives, which can be modeled through the UFPOR Need class. UFPOR also supports a tree structure for organizing activities. By studying activities, a hierarchical model for needs, performance requirements, and performance criteria is developed.

The concept of Time in UFPOR can be used to devise a two-state UFPOR model to represent Sanoff's model. The first model represents the present time where the users, their objectives, and their activities are studied (Figure 38). The users can be modeled through UFPOR People class. Goals and objectives from Sanoff's model can be modeled through instances of UFPOR Need class. UFPOR supports forming a tree structure by associating instances of UFPOR Need class through part-of-a relationships. The instances of UFPOR Need class representing objectives and goals can form a tree structure to match the Sanoff's model. Activities from Sanoff's model can be modeled in a tree structure by using UFPOR Activity class. Figure 38 shows how the concepts from Sanoff's model fall into categories, which can be modeled by UFPOR classes.

Figure 39 represents the future state where analyzing the discovered activities from the first model produces performance criteria, performance requirements and needs. A new tree structure using instances of UFPOR Need class can be used to model and connect performance criteria, performance requirements and needs together. A tree structure of the proposed spaces can be created using instances of UFPOR Environment class. Each space can be associated to a set of performance criteria. UFPOR People and UFPOR Activity were only used in the present state model and are grayed out in Figure 39.

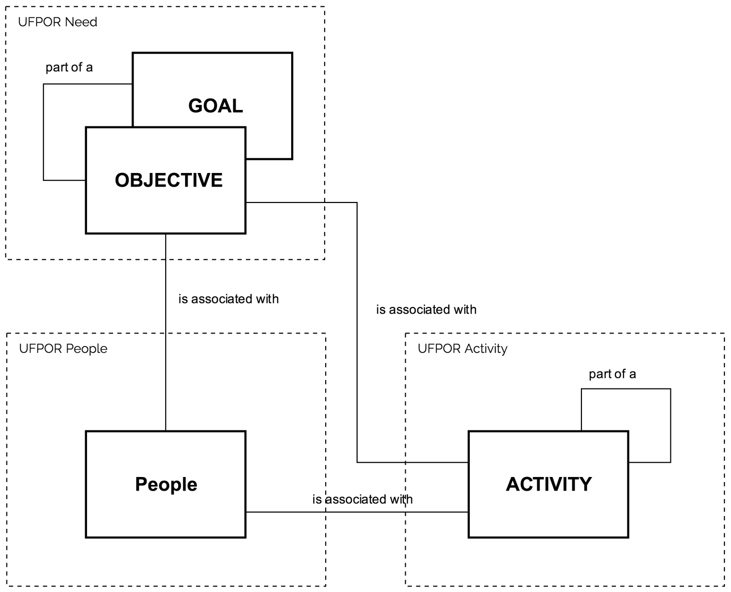


Figure 36, Sanoff's Model, Objectives and Activities in UFPOR

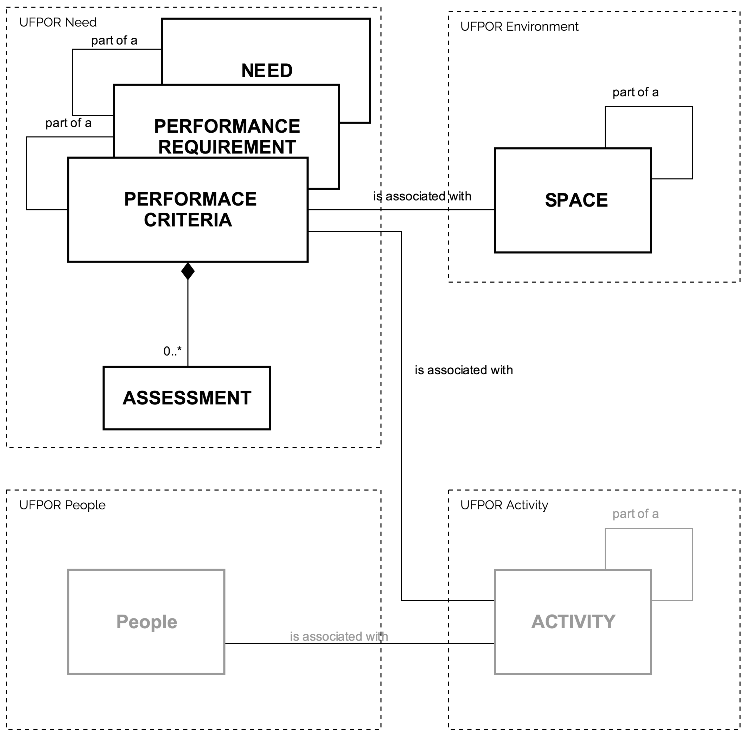


Figure 37, Sanoff's Model, Need Hierarchy and the Future State

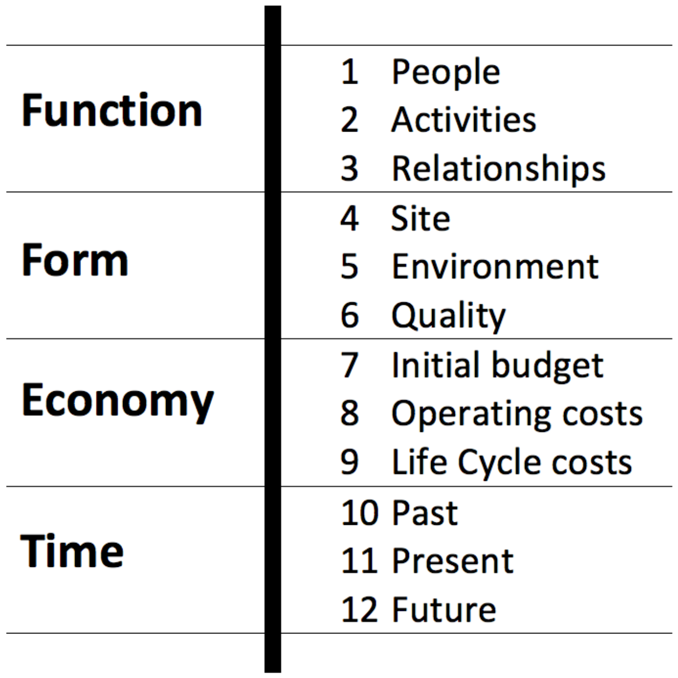
5.4.2. PEÑA’S MODEL

Peña defines a five-step process to define a POR, the five steps are: establishing goals, collecting facts, uncovering concepts, determining needs and stating the problem (Peña, 1977 , p. 24). In each step four areas of consideration including function, form, economy, and time should be explored (Figure 40). The combination of the five steps and four considerations creates a five by four table that summarizes Peña’s model.

The concept of creating different time scopes is instrumental in translating Peña’s model to an object-oriented system. It allows the usage of the same classes to represent several concepts. In UFPOR the concept of Time serves the same purpose in a more flexible way. Instead of changing the scope of a particular class, one can move the whole model from past to future for data gathering and analysis in different time spans.

If UFPOR is capable of representing Peña’s five by four table, then it can represent Peña’s model. UFPOR base classes cover the areas of considerations. UFPOR Environment class covers Form consideration, and Function consideration gets covered by UFPOR People and Activity class. Economy consideration does not correlate to a specific class in UFPOR; instead cost-related attributes are added to all UFPOR classes. Time consideration promotes paying attention to different time spans (past, present, and future); Time is also an inherent concept in UFPOR which allows creation of parallel models covering different time spans.

From the five steps that form Peña’s model columns, Goals, Concepts, and Needs form a tree structure through the usage of UFPOR Need class. The other two steps are Fact and Problem. The former represent the present time while the latter represents future. Through the concept of Time in UFPOR, a two-phase model (one representing present and one representing future) can provide the required tools for representing Peña's model.



Function	1 People 2 Activities 3 Relationships
Form	4 Site 5 Environment 6 Quality
Economy	7 Initial budget 8 Operating costs 9 Life Cycle costs
Time	10 Past 11 Present 12 Future

Figure 38, Areas of Consideration. Reprinted from (Peña, 1977, p. 30).

Adjacencies and groupings are also key concepts in Peña's model. In different time scopes and for different entities, the adjacent and separated entities should be documented (or proposed if it is related to the future state). The concept of composition in UFPOR provides a flexible way to group the entities together and create part-of-a relationships between them.

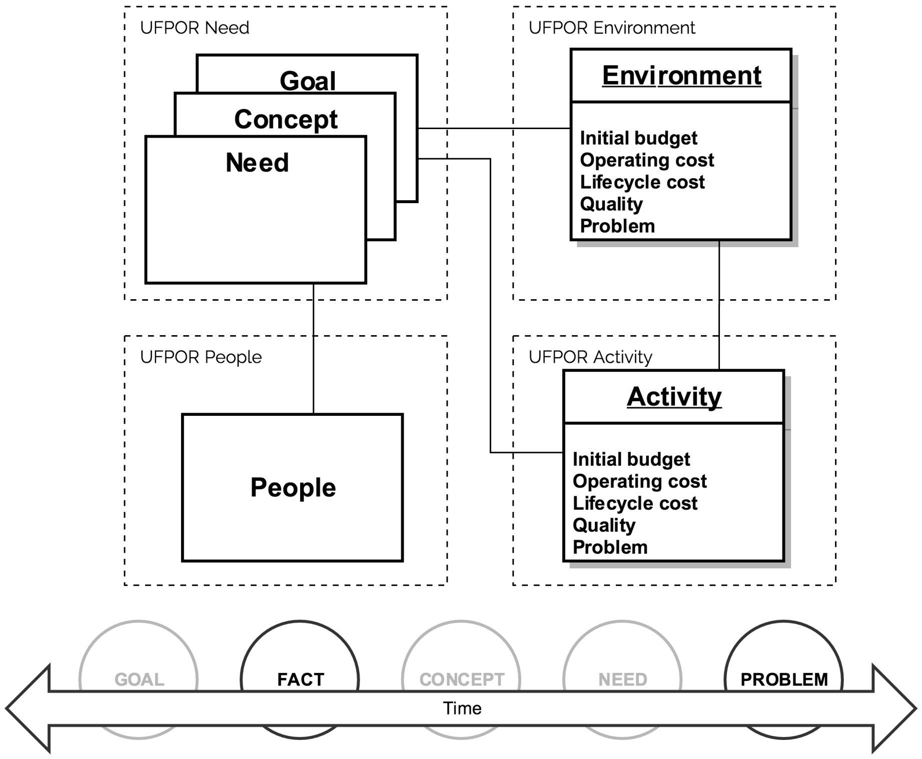


Figure 39, Using UFPOR to Support Pena's Model

5.4.3. DUERK'S MODEL

Preparing a POR by following Duerk's model starts with collecting facts in three different categories: Context, Site, and People/User. Facts will be grouped together based on issues. Solutions are proposed in a hierarchical tree structure where the Mission Statement is the root of the tree and concepts are the leaves.

Duerk defines two major steps in preparing a POR. The first step is Analyzing the Existing State, and the second step is Projecting the Future step. One can use the concept of Time in UFPOR to organize Duerk's model in UFPOR in a two-state-model.

The first state reflects how UFPOR can be used in analyzing the existing state (Figure 42), while the second state shows the projected future state (Figure 43).

In analyzing the existing state, an instance of Environment class is used to study the project site and another instance is used to study the urban context (if applicable). The instance representing the site forms a part-of-a relationship with the instance representing the urban context. Depending on the project, other instances of Environment class can be added to the tree structure. For example if studying the state or country where the project is located has any significance (raises issues) it can be added to the Environment tree structure.

People class is used to study the social context of the project. Similar to Environment class, a tree structure can be used is studying the society in different scales if needed. Activity class is used to document the interaction between the society and the urban context. Need class is not required to document the existing state in Duerk's model.

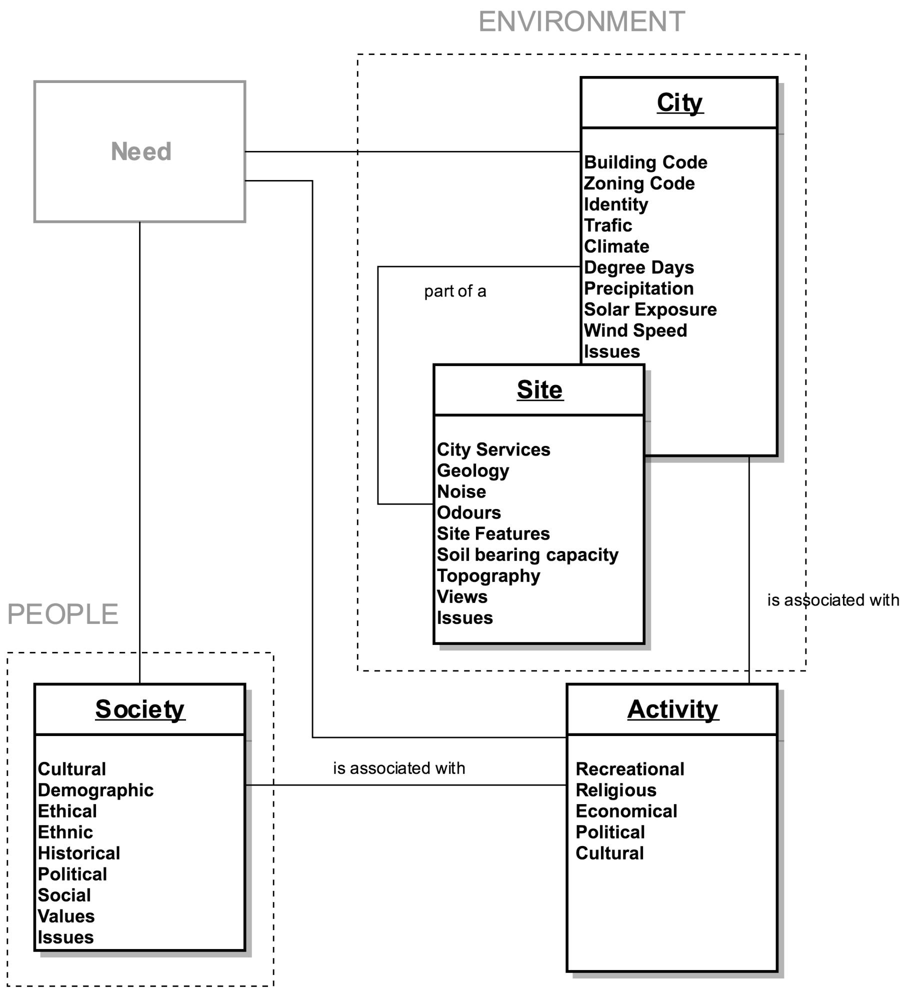


Figure 40, Duerk's Model Analyzing the Existing State in UFPOR

In Duerk's model Projecting the Future State is done through a tree hierarchy of missions, goals, performance requirements and concepts. UFPOR Need class can support such model since it implements the composition pattern. Environment tree hierarchy is developed as an outcome of analyzing the Needs and can be modeled by UFPOR Environment class. Users are studied in detail in Duerk's model and UFPOR supports modeling the users as well as the client in a tree hierarchy through People class.

One of the attributes that is analyzed for users is their activities. UFPOR provides a separate structure to document all the information related to the activities. Figure 43 shows how UFPOR can be used in modeling the Projection of the Future State in Duerk's model.

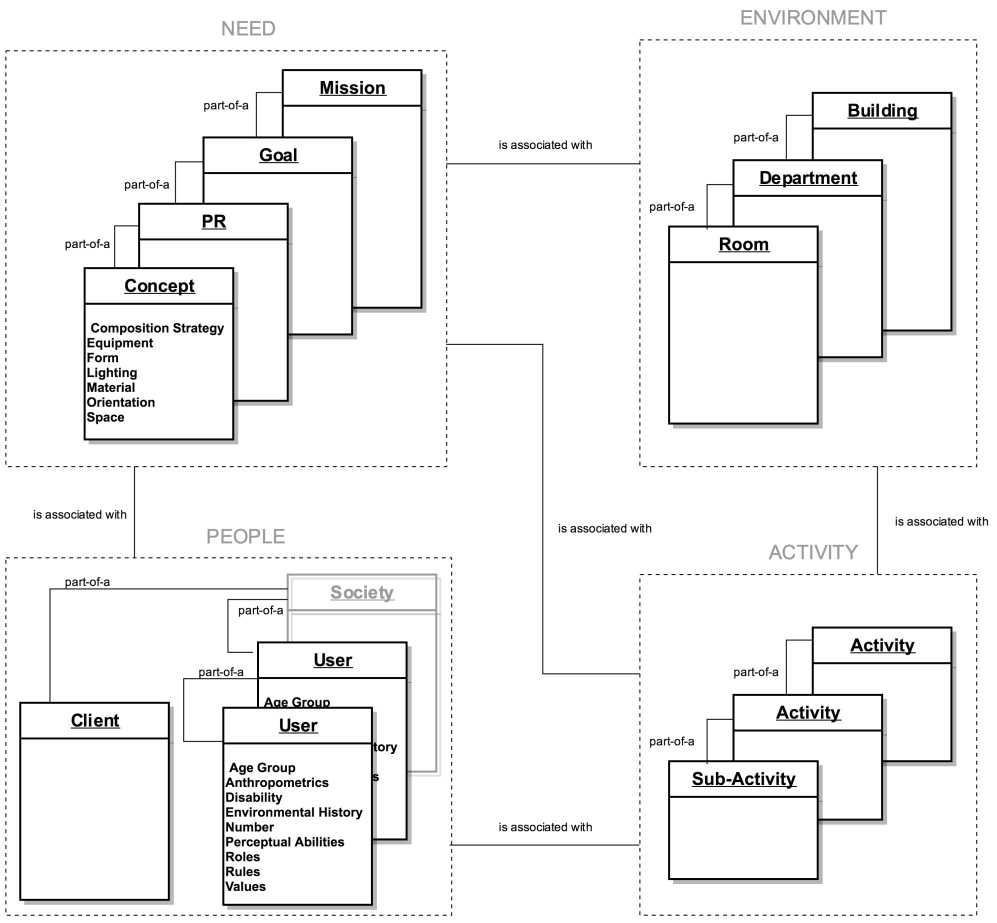


Figure 41, Duerk's Model, Projecting the Future State in UFPOR

5.5. CONCLUSION

The first part of this chapter showed how the three expert models reviewed in the previous chapter are all based on common principles that enables them to be represented through one model. People, Environment, Activity, and Need are four base classes that together form UFPOR. Different combinations of these four classes can be used to represent a POR. The concepts of Time and Composition are critical in using the four base classes in modeling a POR.

The second part of this chapter examined the capabilities of UFPOR in representing the expert models. UFPOR provided a flexible blueprint that could accommodate the three expert models. The three experts models are not equal in how they use UFPOR base classes to represent their content. UFPOR base classes provide a flexible toolset for modeling different aspect of the three expert models.

6. UFPOR IMPLEMENTATION FOR A POR EXCERPT

The previous chapter describes how the three expert models can be represented in UFPOR. By demonstrating UFPOR capabilities in modeling the three expert models, this is an indirect proof of UFPOR capabilities in representing POR models from the industry that conform into one of the three expert models. This chapter provides a concrete demonstration of UFPOR capabilities in modeling a POR excerpt from the AECO industry.

A POR excerpt from a real project was selected to investigate whether the UFPOR is adequate for achieving a comprehensive representation. The selected POR excerpt includes several attributes and instances for UFPOR base classes, requiring use of all UFPOR base classes.

6.1. INTRODUCING THE POR EXCERPT

A POR excerpt was provided by the CRS Center for Leadership & Management in Design & Construction Industry at Texas A&M University. The excerpt includes samples for typical instruction spaces from a detailed schematic design program for an art school (See Appendix 1).

The POR excerpt provides detailed requirements for five instructional spaces: 3D Foundations – Discussion Room, 3D Foundations – Making Space, Life Room, Traditional Drawing (a), and Traditional Drawing (b). For each instructional space, a list of attributes and amenities was provided. Below are the attributes for each instructional space.

- *Space Name,* The space name describes the space and its main role in a few words, examples are: “3D Foundations-Making Space”, and “Traditional Drawing”.
- *Description,* The space description is a short paragraph defining the activities and their spatial requirements for that particular space. The requirements include adjacency requirements, zoning requirements, security requirements, visibility requirements, lighting requirements, and special equipment requirements. Here are a few sentences from the description section: “This room is a priority for natural light”, “This classroom is intended for the instruction of traditional drawing”, and “Black-out shades should be provided at windows to control lighting when required.”
- *Qualities,* Prescribes the overall space quality through a collection of adjectives. Examples are: “Messy”, “Semi-open”, “Dirty”, “Clean”, and “Closed.”
- *Quantity,* The count for the required instances of this space.
- *Number of Occupants,* The required headcount of the students and instructors using this space. For example: “18 + 1” where “18” refers to the students and “1” refers to the instructor.
- *Similar To,* None of the five spaces provided in the excerpt use this field.
- *Category,* The space grouping to which the space belongs: Examples are: “Fabrication”, and “Classroom.”

- *Furniture,* Provides a list of furniture along with counts that the space is required to accommodate.
- *Equipment,* Provides a list of equipment along with counts that should be embedded in the space.
- *Storage,* Provides a list of requirements to guaranty the proper accommodation of tool and material when not in use. Examples are: “Metal painting storage shelves”, and “Coat hooks.”
- *Lighting & Ceiling,* Provides qualitative and quantitative requirements on natural and artificial space lighting. Examples are: “Day-light balanced wall-wash lighting”, and “Adjustable stage-type lighting at model platform.”
- *Power & Data,* Provides specific information about the electricity and Internet connections. Examples are: “Wireless connectivity throughout”, and “Hard-wire power and data connections at computer stations.”
- *Finishes & Surfaces,* Provides requirements on flooring and wall surfaces. Examples are: “Tackable wall surfaces”, and “Hard-surface flooring.”
- *Notes,* Provides requirements that did not fit into any of the above categories. Examples are: “Black-out shades at windows” and “Locate near foundation equipment.”

- *Location,* Provides requirements on location and adjacencies. Examples are: “Within fabrication facility”, and “No proximity to fabrication facility required.”
- *Diagram,* Provides a conceptual drawing of the space itself or how the space is connected with other spaces adjacent to it. Figure 44 and Figure 45 are two examples.

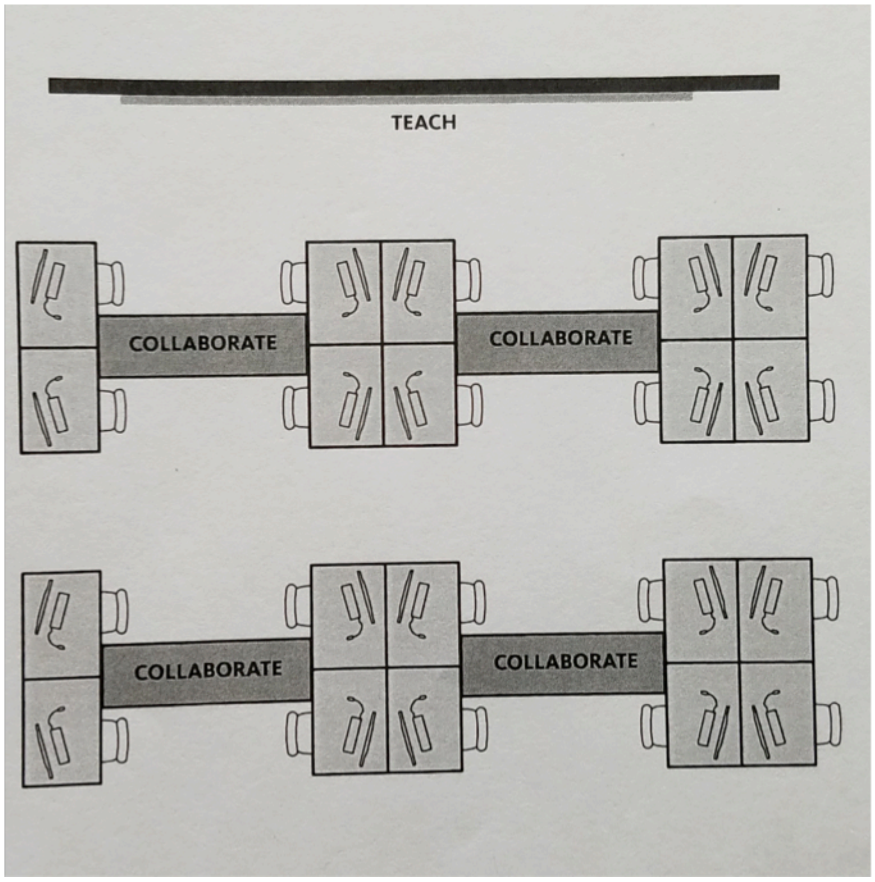


Figure 42, Sample Space Type Diagram from the POR Excerpt

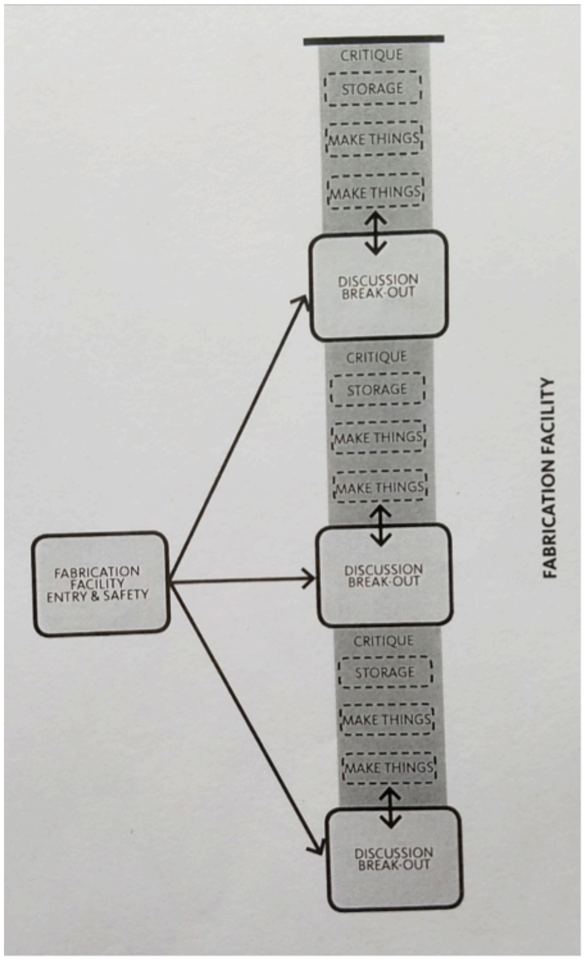


Figure 43, Sample Adjacency Diagram from the POR Excerpt

6.1.1. REPRESENTING THE EXCERPT USING UFPOR

The architectural programing requirements in the POR excerpt are organized based on the required spaces. For each space several attributes such as name, quantity, and occupant count are provided. The space and the attributes can be modeled through UFPOR Environment class.

Some of the attributes for each space describe the occupants/users (e.g. instructors and students.) The users can be modeled through UFPOR People.

The name of the project (Art School) indicates a high-scale goal for the users in using the facility, which is learning and teaching art. Sub-goals such as learning and teaching traditional drawing are implied in each space description. Such goals can be modeled through UFPOR Need.

Besides the direct space attributes, each space is provided with the activities that take place in the space, such as discussion, and drawing. Those activities are the result of analyzing the goals and can be modeled through UFPOR Activity.

For each activity a few requirements are mentioned. For example, for “Making” activity, “Collaboration” and “Storage Space” are mentioned as the requirements. Activity requirements can be modeled through UFPOR Need instances. Those UFPOR Need instances determine the spaces and the direct spatial requirements for each space such as “Wireless connectivity” and “Hard-surface flooring.”

Figure 46 is a simplified UML diagram that shows how the POR excerpt can be presented using UFPOR base classes. UFPOR People class is used to represent the occupants/users. Two instances of this class (student, instructor) were introduced in the excerpt.

UFPOR Need was used in three capacities. The first capacity (marked in Figure 46 as “Need (a)” is to capture occupants/users goals in using the facility. “Learning Art” is an instance of the Need (a) class representing a high-level goal. “Learning Drawing” and “Learning 3D Foundations” are two other instances that can maintain a part-of-a relationship with the “Learning Art” instance.

Each instance of Need (a) class can result in one or more activities that are performed by the users (represented by UFPOR People) in the facility. For example to achieve “Learning 3D Foundations” as a goal (represented by UFPOR Need), the facility should support activities such as “Discussion” and “Light Fabrication”. Such activities can be represented by instances of UFPOR Activity class.

Instances of the Activity class can impose spatial requirements. In other words, they have spatial needs. Such needs are captured in Need (b) class. For example “Light Fabrication” activity requires “Dedicated zone for fabrication.”

A list of required spaces (in this example the classrooms) results from the activities’ needs. Those spaces are represented through UFPOR Environment class. “Traditional Drawing Classroom” is an example for that. UFPOR Environment is shown as “Environment” in Figure 46 and captures a list of attributes for each classroom including Name, Quantity, and Occupant count.

The spatial requirements that are directly imposed on each instance of the Environment class are captured through instances of “Need (c)” class. “Wireless connectivity” and “Hard-surface flooring” are two examples for that.

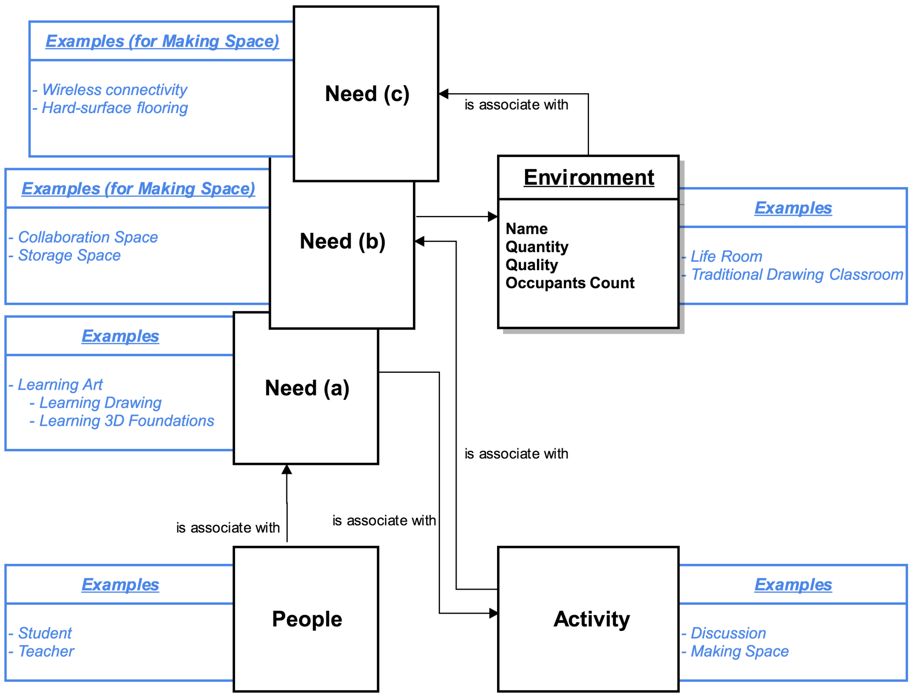


Figure 44, Simplified UML Diagram for POR Excerpt UFPOR Model

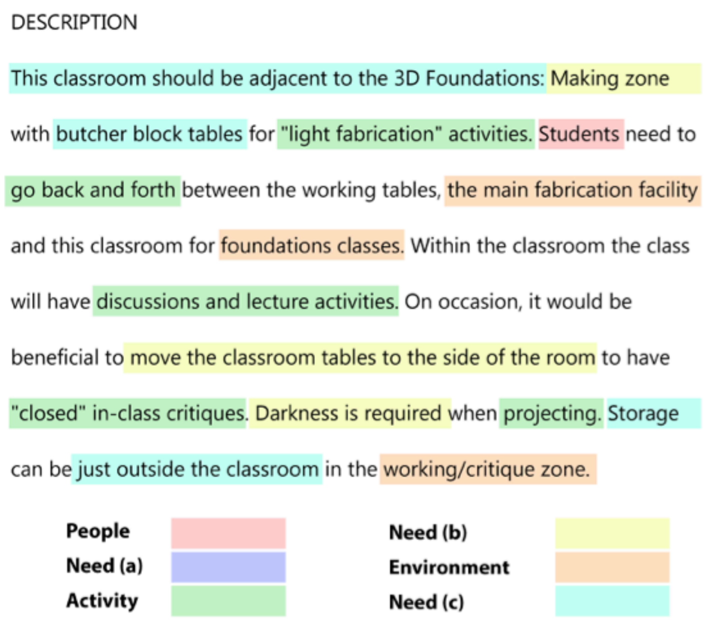
6.2. EXCERPT APPLICATION IMPLEMENTATION

There are two minimum requirements for an implementation platform for UFPOR. The first one is the capability to define data templates (representing classes) and second one is to create connections between the data templates. QuickBase is a cloud-based development platform similar to MS Access, which allows users to create data management applications that consist of connected data tables (QuickBase, 2016).

A data management application with six main data tables representing the six main tables of the excerpt UFPOR model was created in QuickBase. Additional auxiliary data tables were added to the application to better support the content of each table and the relationships between the tables. For example the adjacency requirements were captured through a separate table attached to Need (c) table.

6.3. REFORMATTING THE EXCERPT INTO UFPOR FORMAT

Each phrase from the excerpt corresponds with one of the tables of the UFPOR model. Figure 47 shows how data entities from the excerpt were identified for each table. A level of interpretation is involved in translating the text to the data models. For example, “Storage” can be modeled as an instance of Need (c) class which would be associated with an instance of the Environment class. In a different approach “Storage” can be directly modeled by an instance of Environment class.



DESCRIPTION

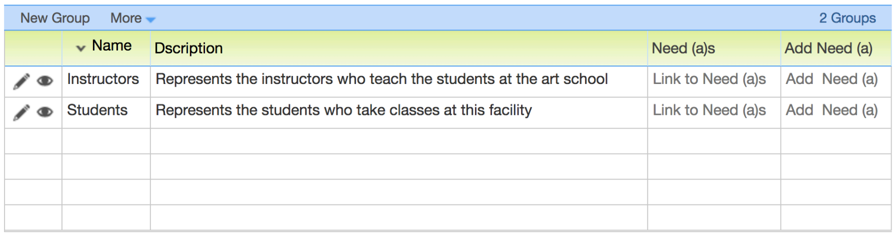
This classroom should be adjacent to the 3D Foundations: Making zone with butcher block tables for "light fabrication" activities. Students need to go back and forth between the working tables, the main fabrication facility and this classroom for foundations classes. Within the classroom the class will have discussions and lecture activities. On occasion, it would be beneficial to move the classroom tables to the side of the room to have "closed" in-class critiques. Darkness is required when projecting. Storage can be just outside the classroom in the working/critique zone.

People	🟥	**Need (b)**	🟨
Need (a)	🟪	**Environment**	🟧
Activity	🟩	**Need (c)**	🟦

Figure 45, Mapping Phrases from the POR Excerpt to UFPOR Classes

Besides the explicit entities that can be mapped to the UFPOR tables directly, the excerpt also includes implicit data entities for UFPOR data tables. For example, the excerpt implies that the students attend the Art School with the goal of learning art. Therefore "Learning Art" is an entity for Need (a) table. On the other hand, instructors have the goal of teaching art, which creates another entity for Need (a) table.

Figure 48 shows the People table with its content. The table includes fields for name and description along with RELATIONSHIP fields to add connections to Need (a) table.

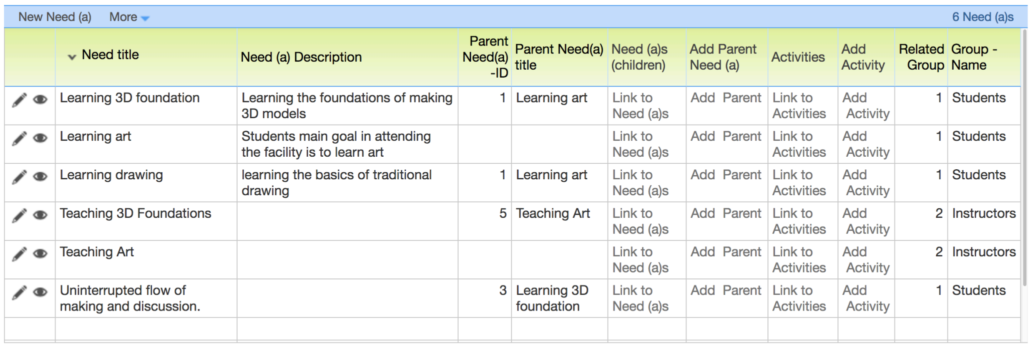


New Group More 2 Groups

	Name	Dscription	Need (a)s	Add Need (a)
✎ 👁	Instructors	Represents the instructors who teach the students at the art school	Link to Need (a)s	Add Need (a)
✎ 👁	Students	Represents the students who take classes at this facility	Link to Need (a)s	Add Need (a)

Figure 46, Art School Excerpt's People Table

Figure 49 shows Need (a) table. The table has six data rows representing the data entities and ten columns representing the available attributes for each entity. The first two attributes are title and description, which describe what the instance is. The other eight fields represent the internal relationships between the entities of the table as well as external relationships with Activity and People table. Through the internal relationships entities can form part-of-a relationships with each other. For example the need of "Learning drawing" has a part-of-a relationship with "Learning art" need. Through an external relationship with Activity table, several activities can be associated with each entity of Need (a) table. The table also has an external relationship with People table, which identifies with which group of people the entities correspond.

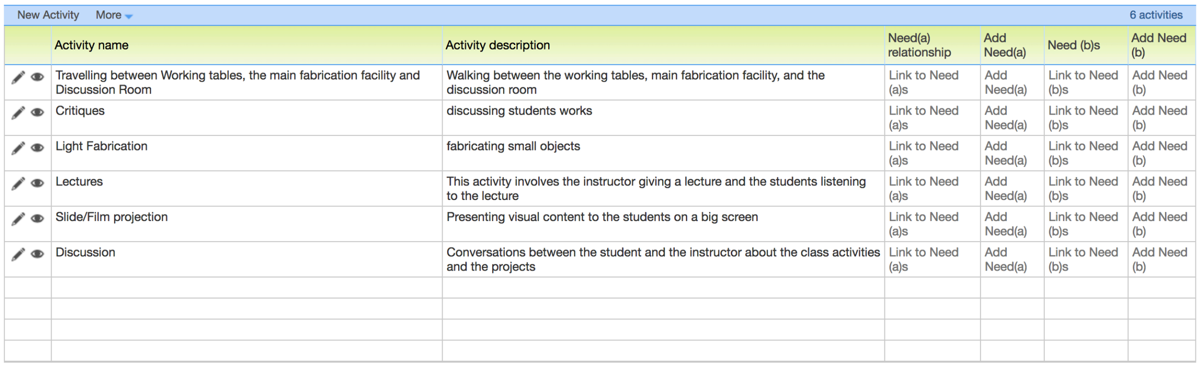


New Need (a) More 6 Need (a)s

	Need title	Need (a) Description	Parent Need(a) -ID	Parent Need(a) title	Need (a)s (children)	Add Parent Need (a)	Activities	Add Activity	Related Group	Group - Name
✎ 👁	Learning 3D foundation	Learning the foundations of making 3D models	1	Learning art	Link to Need (a)s	Add Parent	Link to Activities	Add Activity	1	Students
✎ 👁	Learning art	Students main goal in attending the facility is to learn art			Link to Need (a)s	Add Parent	Link to Activities	Add Activity	1	Students
✎ 👁	Learning drawing	learning the basics of traditional drawing	1	Learning art	Link to Need (a)s	Add Parent	Link to Activities	Add Activity	1	Students
✎ 👁	Teaching 3D Foundations		5	Teaching Art	Link to Need (a)s	Add Parent	Link to Activities	Add Activity	2	Instructors
✎ 👁	Teaching Art				Link to Need (a)s	Add Parent	Link to Activities	Add Activity	2	Instructors
✎ 👁	Uninterrupted flow of making and discussion.		3	Learning 3D foundation	Link to Need (a)s	Add Parent	Link to Activities	Add Activity	1	Students

Figure 47, Art School Excerpt's Need (a) Table

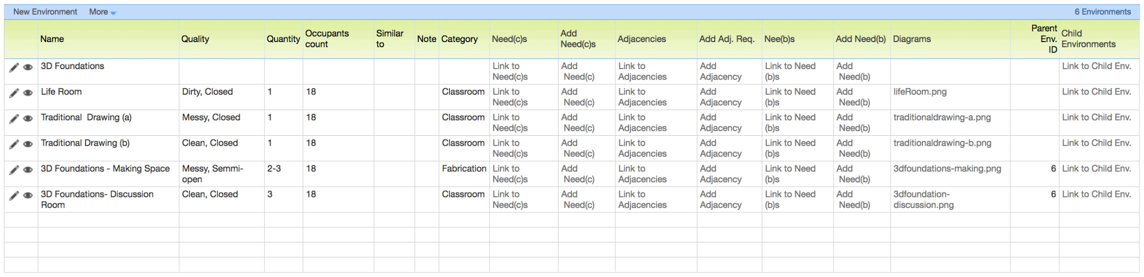
Figure 50 shows Activity table. In figure 50, the data rows representing the entities are only the activities that correspond with “3D Foundation, Discussion” space. The table provides six columns. The first two columns provide name and description attributes for each activity while the next four columns provide relationships between this table and both Need (a) and Need (b) tables.



Activity name	Activity description	Need(a) relationship	Add Need(a)	Need (b)s	Add Need (b)
Travelling between Working tables, the main fabrication facility and Discussion Room	Walking between the working tables, main fabrication facility, and the discussion room	Link to Need (a)s	Add Need(a)	Link to Need (b)s	Add Need (b)
Critiques	discussing students works	Link to Need (a)s	Add Need(a)	Link to Need (b)s	Add Need (b)
Light Fabrication	fabricating small objects	Link to Need (a)s	Add Need(a)	Link to Need (b)s	Add Need (b)
Lectures	This activity involves the instructor giving a lecture and the students listening to the lecture	Link to Need (a)s	Add Need(a)	Link to Need (b)s	Add Need (b)
Slide/Film projection	Presenting visual content to the students on a big screen	Link to Need (a)s	Add Need(a)	Link to Need (b)s	Add Need (b)
Discussion	Conversations between the student and the instructor about the class activities and the projects	Link to Need (a)s	Add Need(a)	Link to Need (b)s	Add Need (b)

Figure 48, Art School Excerpt's Need (b) Table

Figure 51 shows Environment table. With sixteen columns this table is the biggest table amongst the POR excerpt tables. Eight columns are dedicated to the direct attributes of each entity including Name, Quality, Quantity, Occupant Count, Note, Category, Similar to, and Diagram, while the other eight columns manage the relationships between entities from Environment table and other tables. Environment table is connected with Need (b) and Need (c) table. It also has a relationship with an auxiliary table to manage adjacency requirements as well as an internal relationship to define part-of-a relationship between the entities.

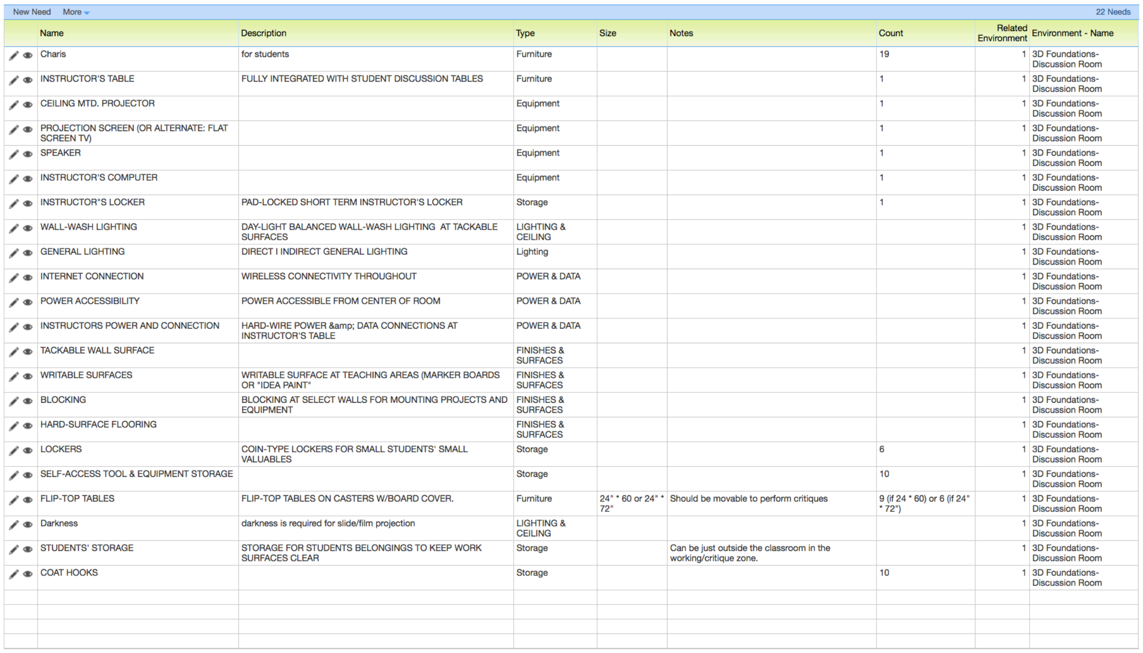


New Environment More 6 Environments

	Name	Quality	Quantity	Occupants count	Similar to	Note	Category	Need(c)s	Add Need(c)s	Adjacencies	Add Adj. Req.	Nee(b)s	Add Need(b)	Diagrams	Parent Env. ID	Child Environments
	3D Foundations							Link to Need(c)s	Add Need(c)	Link to Adjacencies	Add Adjacency	Link to Need (b)s	Add Need(b)			Link to Child Env.
	Life Room	Dirty, Closed	1	18			Classroom	Link to Need(c)s	Add Need(c)	Link to Adjacencies	Add Adjacency	Link to Need (b)s	Add Need(b)	lifeRoom.png		Link to Child Env.
	Traditional Drawing (a)	Messy, Closed	1	18			Classroom	Link to Need(c)s	Add Need(c)	Link to Adjacencies	Add Adjacency	Link to Need (b)s	Add Need(b)	traditionaldrawing-a.png		Link to Child Env.
	Traditional Drawing (b)	Clean, Closed	1	18			Classroom	Link to Need(c)s	Add Need(c)	Link to Adjacencies	Add Adjacency	Link to Need (b)s	Add Need(b)	traditionaldrawing-b.png		Link to Child Env.
	3D Foundations - Making Space	Messy, Semmi-open	2-3	18			Fabrication	Link to Need(c)s	Add Need(c)	Link to Adjacencies	Add Adjacency	Link to Need (b)s	Add Need(b)	3dfoundations-making.png	6	Link to Child Env.
	3D Foundations- Discussion Room	Clean, Closed	3	18			Classroom	Link to Need(c)s	Add Need(c)	Link to Adjacencies	Add Adjacency	Link to Need (b)s	Add Need(b)	3dfoundation-discussion.png	6	Link to Child Env.

Figure 49, Art School Excerpt's Environment Table

Figure 52 shows Need (c) table populated with the entities related to 3D Foundations-Discussion Room. This table provides eight columns with six of them providing direct attributes and two of them managing the relationship between this table and Environment table. The direct attributes are Name, Description, Type, Size, Notes, and Count. The Type attribute is selected from a collection of preselected values and determines the category of the requirement. The choices are Furniture, Equipment, Storage, Lighting & Ceiling, Lighting, Power & Data, Finishes & Surfaces, and Storage.



Name	Description	Type	Size	Notes	Count	Related Environment	Environment - Name
Charis	for students	Furniture			19	1	3D Foundations-Discussion Room
INSTRUCTOR'S TABLE	FULLY INTEGRATED WITH STUDENT DISCUSSION TABLES	Furniture			1	1	3D Foundations-Discussion Room
CEILING MTD. PROJECTOR		Equipment			1	1	3D Foundations-Discussion Room
PROJECTION SCREEN (OR ALTERNATE: FLAT SCREEN TV)		Equipment			1	1	3D Foundations-Discussion Room
SPEAKER		Equipment			1	1	3D Foundations-Discussion Room
INSTRUCTOR'S COMPUTER		Equipment			1	1	3D Foundations-Discussion Room
INSTRUCTOR"S LOCKER	PAD-LOCKED SHORT TERM INSTRUCTOR'S LOCKER	Storage			1	1	3D Foundations-Discussion Room
WALL-WASH LIGHTING	DAY-LIGHT BALANCED WALL-WASH LIGHTING AT TACKABLE SURFACES	LIGHTING & CEILING				1	3D Foundations-Discussion Room
GENERAL LIGHTING	DIRECT I INDIRECT GENERAL LIGHTING	Lighting				1	3D Foundations-Discussion Room
INTERNET CONNECTION	WIRELESS CONNECTIVITY THROUGHOUT	POWER & DATA				1	3D Foundations-Discussion Room
POWER ACCESSIBILITY	POWER ACCESSIBLE FROM CENTER OF ROOM	POWER & DATA				1	3D Foundations-Discussion Room
INSTRUCTORS POWER AND CONNECTION	HARD-WIRE POWER &amp; DATA CONNECTIONS AT INSTRUCTOR'S TABLE	POWER & DATA				1	3D Foundations-Discussion Room
TACKABLE WALL SURFACE		FINISHES & SURFACES				1	3D Foundations-Discussion Room
WRITABLE SURFACES	WRITABLE SURFACE AT TEACHING AREAS (MARKER BOARDS OR "IDEA PAINT"	FINISHES & SURFACES				1	3D Foundations-Discussion Room
BLOCKING	BLOCKING AT SELECT WALLS FOR MOUNTING PROJECTS AND EQUIPMENT	FINISHES & SURFACES				1	3D Foundations-Discussion Room
HARD-SURFACE FLOORING		FINISHES & SURFACES				1	3D Foundations-Discussion Room
LOCKERS	COIN-TYPE LOCKERS FOR SMALL STUDENTS' SMALL VALUABLES	Storage			6	1	3D Foundations-Discussion Room
SELF-ACCESS TOOL & EQUIPMENT STORAGE		Storage			10	1	3D Foundations-Discussion Room
FLIP-TOP TABLES	FLIP-TOP TABLES ON CASTERS W/BOARD COVER.	Furniture	24" * 60 or 24" * 72"	Should be movable to perform critiques	9 (if 24 * 60) or 6 (if 24" * 72")	1	3D Foundations-Discussion Room
Darkness	darkness is required for slide/film projection	LIGHTING & CEILING				1	3D Foundations-Discussion Room
STUDENTS' STORAGE	STORAGE FOR STUDENTS BELONGINGS TO KEEP WORK SURFACES CLEAR	Storage		Can be just outside the classroom in the working/critique zone.		1	3D Foundations-Discussion Room
COAT HOOKS		Storage			10	1	3D Foundations-Discussion Room

Figure 50, Art School Excerpt's Need (c) Table

6.4. CONCLUSION

UFPOR provided the required structure to capture all the information presented by the Art School POR excerpt. All four base classes of UFPOR were used in capturing data entities from Art School POR excerpt. The base classes were further adjusted by additional attributes to accommodate data fields for capturing all the attributes from the text.

According to the concept of Time in UFPOR, several instances of each base class can be used to capture POR information in different time spans. Three instances of Need class were used to capture all the Need-related entities of Art School POR excerpt.

While the three instances of Need class (Need (a), Need (b), Need(c)) did not represent distinct time spans, they did form a sequential relationship with each other. The concept of Time in UFPOR can be re-defined to better accommodate sequential instances of the same base class that may not necessarily exhibit distinct time spans, but form a sequential relationship with each other.

The concept of Composition was used in multiple occasions to form part-of-a relationships between the entities of a table with other entities from the same table.

The Art School POR excerpt provided several entities that were not explicitly included in the POR's text. Those entities were captured by the developed prototype. Capturing the implied requirements is not possible without knowing the basic concepts of UFPOR and making logical conclusions from the provided data. To convert an existing POR to UFPOR format the individuals in charge of the task need to be educated on the basic concepts of UFPOR.

7. ADAPTING THE UFPOR TO THE IFC

To examine the research hypothesis and evaluate the capabilities of IFC in representing the data supported by UFPOR, a subset of IFC (an MVD) that is capable of representing UFPOR should be identified.

To propose an MVD that can represent UFPOR, I found one or more entities in IFC for each UFPOR class that can represent the same data. IFC entities were evaluated in five capacities. First, their definition and the category that they belong to should match the respective UFPOR class; second, their capability in representing the same attributes either directly or through dynamically extendable or statically defined custom property sets; third, the selected entities should be able to represent UFPOR relationships; forth, the selected entities should be capable of implementing the composition pattern; fifth, the selected entities should be capable of representing different time spans such as past, present, and future.

Throughout this chapter the IFC entities, their attributes and property sets reflect the designed/built condition as opposed to the desired/programmed condition unless it is mentioned explicitly. For example IfcBuildingStorey entity provides a property set called Pset_BuildingStoreyCommon, which, as the name suggests, is meant to capture the most common attributes for a building story. Pset_BuildingStoreyCommon provides seven properties. Amongst them are GrossAreaPlanned, which captures total planned area for the building story and NetAreaPlanned, which captures the total planned net area. Besides these two attributes, which explicitly reflect planned areas, the rest of properties reflect the actual design/build values for the property.

In section two of this chapter I show how qualitative and quantitative requirements can be applied to the entities and their properties to add the desired/programmed state.

7.1. UFPOR ENVIRONMENT

Spaces, their combination, and their relationships are one of the concepts that are well explored in architectural programming textbooks. IFC provides a mature structure for defining spatial structures.

The “Product Extension” schema is where the entities related to UFPOR Environment class are defined. IfcProduct and IfcTypeProduct are the two main classes in the “Product Extension”. I discussed their differences and how they can be combined to define building elements in Chapter 2. IfcSpationalElement, a subclass of IfcProduct, and IfcSpatialElementType, a subclass of IfcTypeProduct are the base classes to define spatial structures (Figure 53).

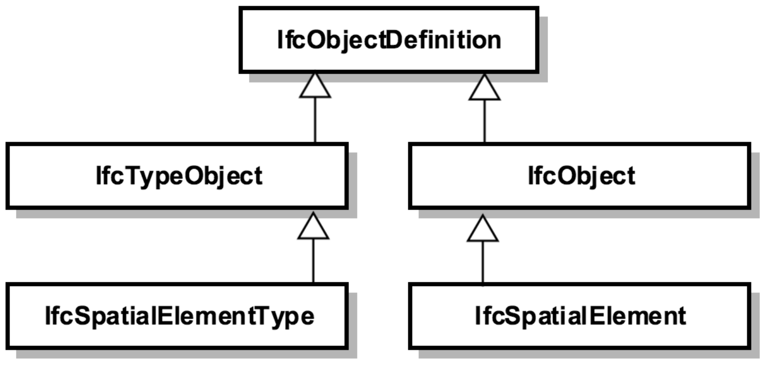


Figure 51, IfcObjectDefinition Subtypes

7.1.1. IfcSpatialElement

IfcSpatialElement is an abstract class that represents some of the most common attributes of the spatial structures. It has three subclasses. IfcExternalSpatialStructure, IfcSpatialStructureElement, and IfcSpatialZone. Code 11 shows the entity definition for IfcSpatialElement.

```
ENTITY IfcSpatialElement
 ENTITY IfcRoot
  GlobalId            : IfcGloballyUniqueId;
  OwnerHistory        : OPTIONAL IfcOwnerHistory;
  Name                : OPTIONAL IfcLabel;
  Description         : OPTIONAL IfcText;
 ENTITY IfcObjectDefinition
 INVERSE
  HasAssignments      : SET OF IfcRelAssigns FOR RelatedObjects;
  Nests               : SET [0:1] OF IfcRelNests FOR RelatedObjects;
  IsNestedBy          : SET OF IfcRelNests FOR RelatingObject;
  HasContext          : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
  IsDecomposedBy      : SET OF IfcRelAggregates FOR RelatingObject;
  Decomposes          : SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
  HasAssociations     : SET OF IfcRelAssociates FOR RelatedObjects;
 ENTITY IfcObject
  ObjectType          : OPTIONAL IfcLabel;
 INVERSE
  IsDeclaredBy        : SET [0:1] OF IfcRelDefinesByObject FOR RelatedObjects;
  Declares            : SET OF IfcRelDefinesByObject FOR RelatingObject;
  IsTypedBy           : SET [0:1] OF IfcRelDefinesByType FOR RelatedObjects;
  IsDefinedBy         : SET OF IfcRelDefinesByProperties FOR RelatedObjects;
 ENTITY IfcProduct
  ObjectPlacement     : OPTIONAL IfcObjectPlacement;
  Representation      : OPTIONAL IfcProductRepresentation;
 INVERSE
  ReferencedBy        : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
 ENTITY IfcSpatialElement
  LongName            : OPTIONAL IfcLabel;
 INVERSE
  ServicedBySystems   : SET OF IfcRelServicesBuildings FOR RelatedBuildings;
  ReferencesElements  : SET OF IfcRelReferencedInSpatialStructure FOR RelatingStructure;
END_ENTITY;
```

Code 11, IfcSpatialElement Inheritance Graph (BuildingSmart, 2016af)

Explaining the attributes of this entity and the patterns that it offers which are not applicable to UFPOR Environment is outside of the scope of this research. Here I only analyze the attributes that can be of use to document UFPOR Environment class through IFC.

7.1.2. SPATIAL DECOMPOSITION

UFPOR Environment class implements the composition pattern, which allows every instance of this class to have a part-of-a relationship with another instance. That is how one can form spatial packages and create a hierarchy of spaces where smaller spaces reside within bigger spaces. IFC also provides a way to implement the same pattern to some extend. IFC defines spatial structure as “A hierarchical tree of spatial structure elements (site, building, [storey], space) ultimately assigned to the project. (BuildingSmart, 2016at)”

IfcRoot provides two inverse relationships called “IsDecomposedBy” and “Decomposes”. The former is used to fill out the parent role and the latter is used for the children role. IfcRelAggregates is the entity that connects the spatial element together. IFC imposes some restrictions on how the spatial elements can relate to each other in the project’s spatial structure “The order of spatial structure elements being included in the concept for building projects are from high to low level: IfcProject, IfcSite, IfcBuilding, IfcBuildingStorey, IfcSpace. Therefore a spatial structure element can only have parts of an element at the same or lower level. (BuildingSmart, 2016a)”

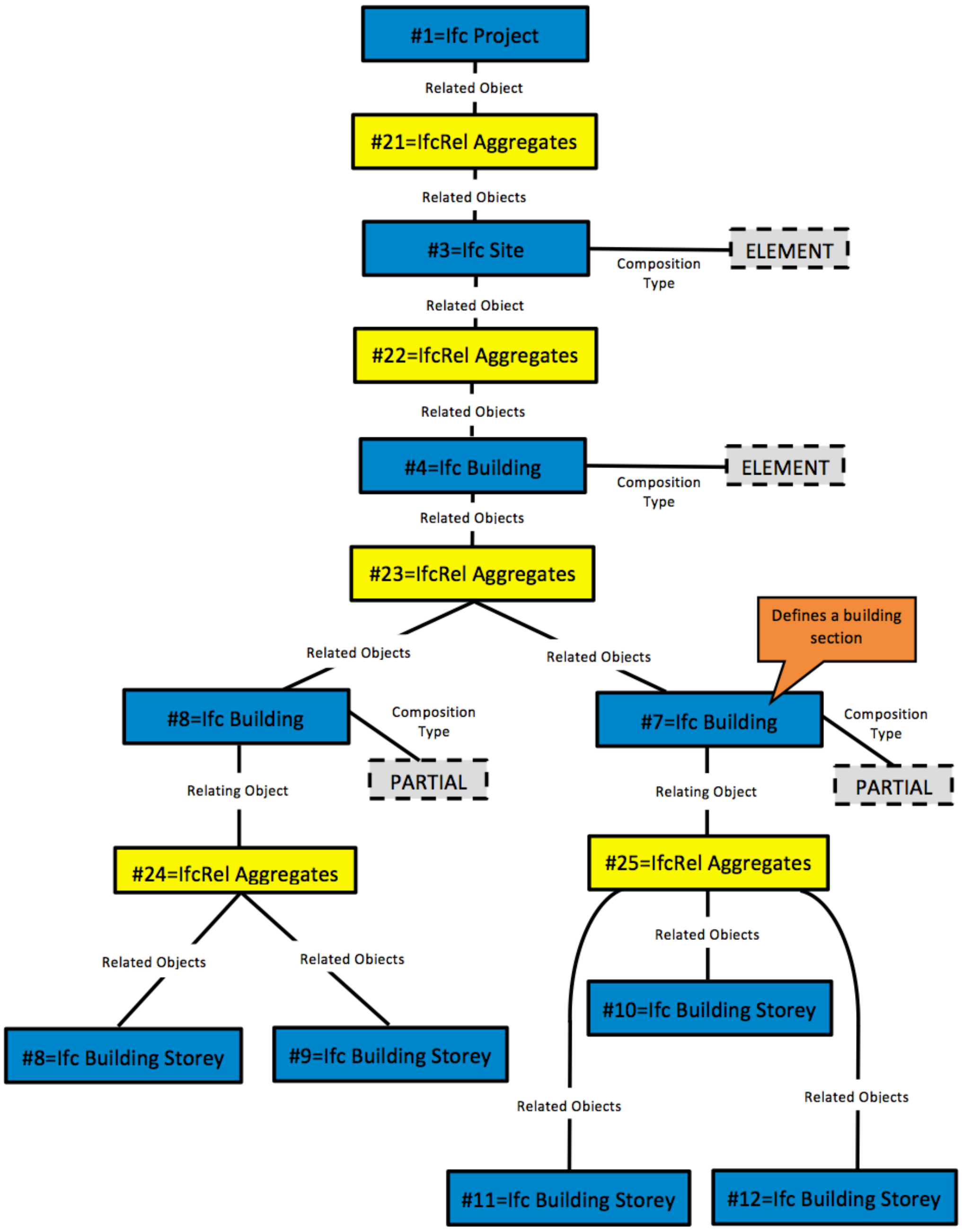


Figure 52, IFC Spatial Structure, (BuildingSmart, 2016ah)

Figure 54 shows how instances of IfcRelAggregate entity can be used to form the spatial structure. The root of every spatial structure is an instance of IfcProject, which is discussed later in this chapter.

7.1.3. IfcSpatialStructure PROPERTY SETS

IFC provides a collection of property sets to further define instances of IfcSpatialStructure, Pset_AirSideSystemInformation, Pset_SpaceFireSafetyRequirements, Pset_SpaceLightingRequirements, Pset_SpaceOccupancyRequirements, Pset_SpaceThermalRequirements, Pset_ThermalLoadAggregate, and Pset_ThermalLoadDesignCriteria are the standard property sets provided by IfcSpatialStructure. These property sets provide standard channels to attach requirements to instances of IfcSpatialStructure. Amongst these property sets Pset_SpaceOccupancyRequirements represents information about the users, who in UFPOR are captured by UFPOR People.
Pset_SpaceOccupancyRequirements is defined as "Properties concerning work activities occurring or expected to occur within one or a set of similar spatial structure elements. (BuildingSmart, 2016ao)" Below are the properties that reside within this property set.

```
OccupancyType (P_SINGLEVALUE / IfcLabel):
Occupancy type for this object. It is defined according to the presiding national code.

OccupancyNumber (P_SINGLEVALUE / IfcCountMeasure):
Number of people required for the activity assigned to this space.

OccupancyNumberPeak (P_SINGLEVALUE / IfcCountMeasure):
Maximal number of people required for the activity assigned to this space in peak time.

OccupancyTimePerDay (P_SINGLEVALUE / IfcTimeMeasure):
The amount of time during the day that the activity is required within this space.

AreaPerOccupant (P_SINGLEVALUE / IfcAreaMeasure):
Design occupancy loading for this type of usage assigned to this space.

MinimumHeadroom (P_SINGLEVALUE / IfcLengthMeasure):
Headroom required for the activity assigned to this space.

IsOutlookDesirable (P_SINGLEVALUE / IfcBoolean):
 An indication of whether the outlook is desirable (set TRUE) or not (set FALSE)
```

Code 12, Pset_SpaceOccupancyRequirements Properties

OccupancyType is defined based on the national building code (*The architect's handbook of professional practice,* 2014). The major types are Assembly (Group A), Business (Group B), Educational (Group E), Factory (Group D), High Hazard (Group H), Institutional (Group I), Mercantile (Group M), Residential (Group R), Storage (Group S), and Utility and Miscellaneous (Group U). This property set can be in accordance with the UFPOR People class. More details are covered in section 7.3.

7.1.4. COMPOSING VERSUS CONTAINING IN SPATIAL STRUCTURE

IfcSpatialElement provides an inverse attribute called ContainsElement. This relationship is with a set of IfcRelContainedInSpatialStructure instances (BuildingSmart, 2016y).

```
ENTITY IfcRelContainedInSpatialStructure
 ENTITY IfcRoot
  GlobalId              :IfcGloballyUniqueId;
  OwnerHistory          :OPTIONAL IfcOwnerHistory;
  Name                  :OPTIONAL IfcLabel;
  Description           :OPTIONAL IfcText;
 ENTITY IfcRelationship
 ENTITY IfcRelConnects
 ENTITY IfcRelContainedInSpatialStructure
  RelatedElements       :SET [1:?] OF IfcProduct;
  RelatingStructure     :IfcSpatialElement;
END_ENTITY;
```

Code 13, IfcRelContainedInSpatialStructure Inheritance Graph (BuildingSmart, 2016y)

IfcRelContainedInSpatialStructure and IfcRelAggregates have similar definitions. IFC specifications are very intentional about differentiating between the two. IfcRelContainedInSpatialStructure is used to connect a spatial element with building elements such as walls and stairs, which form that spatial element while IfcRelAggregates defines the spatial structure. Figure 55 shows the difference between these two relationsips. IfcRelAggregates is the one that is appropriate for creating a spatial structure as long as the spatial entities are subsets of IfcSpatialElement. To add more detailed entities such as walls, and stairs IfcRelContainedInSpatialStructure should be applied. UFPOR Environment class implements the Composition pattern with no scale limitations, therefore both IfcRelAggregates and IfcRelContainedInSpatialStructure are needed to recreate UFPOR Environment class in IFC.

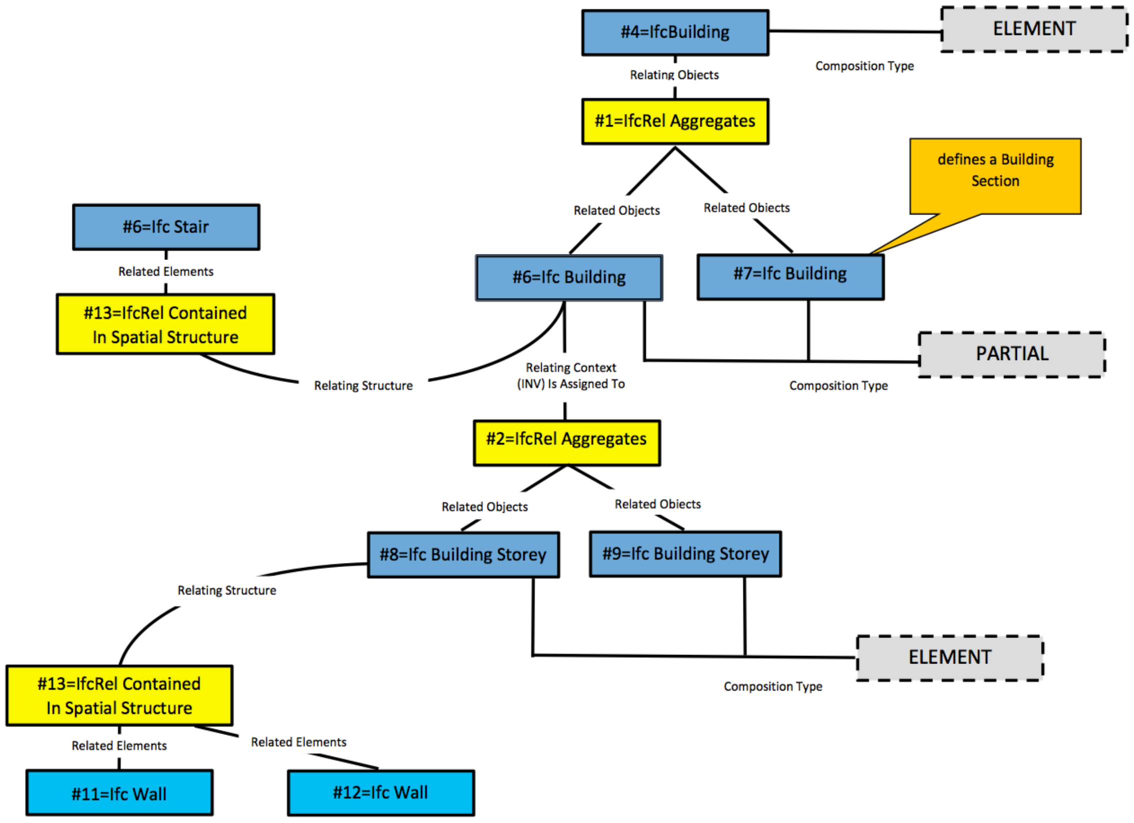


Figure 53, IfcRelAggregate VS IfcRelContained (BuildingSmart, 2016y)

7.1.5. IFCSPATIALSTRUCTUREELEMENT

IfcSpatialStructureElement is one of the three subsets of IfcSpatialElement and the super type of IfcBuilding, IfcBuildingStory, IfcSite, and IfcSpace (Code 14).

```
ENTITY IfcSpatialStructureElement
 ABSTRACT SUPERTYPE OF(ONEOF(IfcBuilding, IfcBuildingStorey, IfcSite, IfcSpace))
 SUBTYPE OF IfcSpatialElement;
  CompositionType         : OPTIONAL IfcElementCompositionEnum;
 INVERSE
  ContainsElements        : SET OF IfcRelContainedInSpatialStructure FOR RelatingStructure;
 WHERE
  WR41        EX(SELF\IfcObjectDefinition.Decomposes) = 1) AND (
              KERNEL.IFCRELAGGREGATES IN
              PEOF(SELF\IfcObjectDefinition.Decomposes[1])) AND (( IFCKERNEL.IFCPROJECT
              TYPEOF (SELF\IfcObjectDefinition.Decomposes[1].RelatingObject)) OR (
              PRODUCTEXTENSION.IFCSPATIALSTRUCTUREELEMENT IN TYPEOF
              LF\IfcObjectDefinition.Decomposes[1].RelatingObject)) )
END_ENTITY;
```

Code 14, IfcSpatialStructureElement Entity Definition (BuildingSmart, 2016ah)

IfcSpatialStructureElement adds only one optional attribute to its super class. CompositionType is an important attribute in forming the spatial structure. IFC puts some restrictions on spatial structure definitions. Amongst them, an instance of IfcSpatialElement (IfcBuilding, IfcSite, IfcBuildingStory, IfcSpace) can only decompose to the spatial element of similar or lower level. If it decomposes to elements of the same level, the CompositionType of the parent element should be set as Complex and the CompositionType of the children should be set as Partial.

```
TYPE IfcElementCompositionEnum = ENUMERATION OF
  (COMPLEX,
   ELEMENT,
   PARTIAL);
END_TYPE;
```

Code 15, IfcElementCompositionEnum (BuildingSmart, 2016h)

7.1.5.1. IFCSPATIALSTRUCTUREELEMENT PROPERTY SETS

Besides inheriting all the property sets from its super classes, IfcSpatialStructureElement provides a new property set called Pset_PropertyAgreement. This property set is to capture the occupancy type and other details and is useful for facility management purposes. According to this property set, the specifications of the users of the IfcSpatialStructureElement instances should be captured through IfcOccupant instances.

7.1.6. IfcProject

Even though IfcProject is not a subclass of IfcSpatialStructureElement, it is the root of IFC spatial composition. IfcProject also provides a shared context for subclasses of IfcObject (including subclasses of IfcSpatialStructureElement) and subclasses of IfcObjectType. While some of the properties can be attached directly to instances of IfcSpatialStructureElement, instances of IfcSpatialElementType can be used to define more generic attributes that are shared by several instances. Figure 56 shows the relationship between IfcProject, IfcObject, and IfcObjectType.

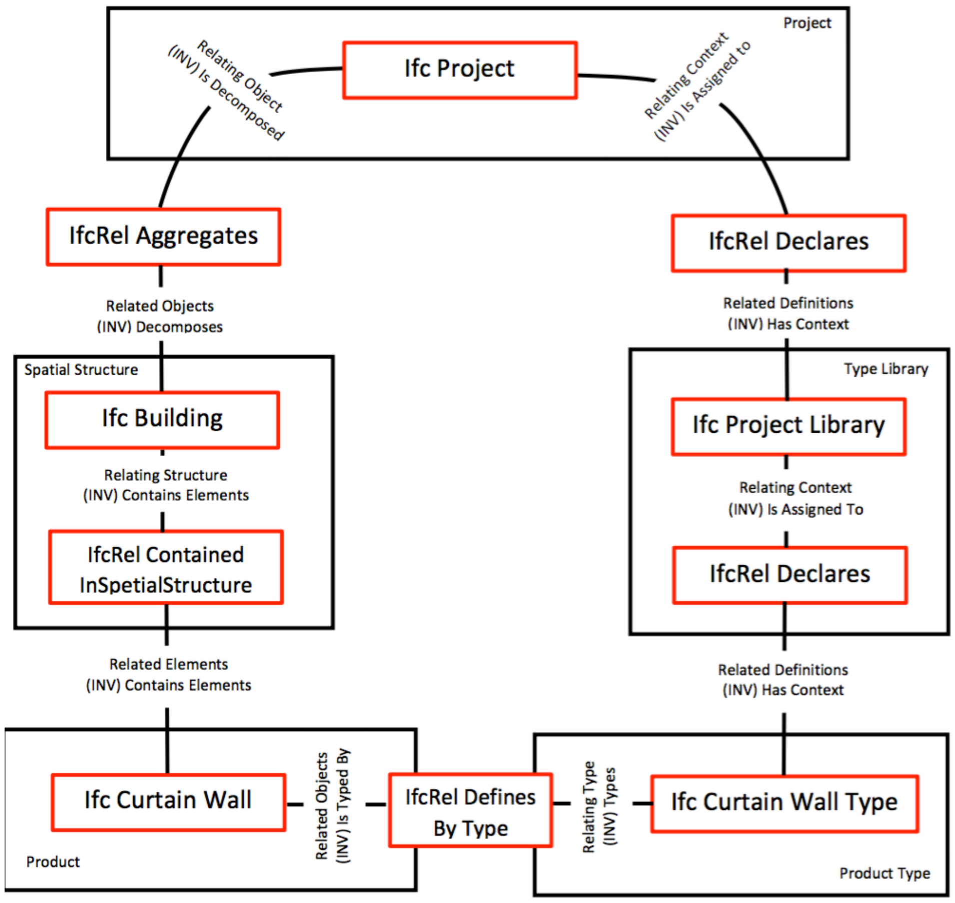


Figure 54, IfcProject as a Context for the Project, (BuildingSmart, 2016u)

IfcProject does not provide any predefined property or quantity sets. Custom property sets can always be used to attach programming related attributes to IfcProject. Code 16 shows the inheritance graph for IfcProject.

```
ENTITY IfcProject
 ENTITY IfcRoot
  GlobalId             :IfcGloballyUniqueId;
  OwnerHistory         :OPTIONAL IfcOwnerHistory;
  Name                 :OPTIONAL IfcLabel;
  Description          :OPTIONAL IfcText;
 ENTITY IfcObjectDefinition
 INVERSE
  HasAssignments       :SET OF IfcRelAssigns FOR RelatedObjects;
  Nests                :SET [0:1] OF IfcRelNests FOR RelatedObjects;
  IsNestedBy           :SET OF IfcRelNests FOR RelatingObject;
  HasContext           :SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
  IsDecomposedBy       :SET OF IfcRelAggregates FOR RelatingObject;
  Decomposes           :SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
  HasAssociations      :SET OF IfcRelAssociates FOR RelatedObjects;
 ENTITY IfcContext
  ObjectType            :OPTIONAL IfcLabel;
  LongName              :OPTIONAL IfcLabel;
  Phase                 :OPTIONAL IfcLabel;
  RepresentationContexts:OPTIONAL SET [1:?] OF IfcRepresentationContext;
  UnitsInContext        :OPTIONAL IfcUnitAssignment;
 INVERSE
  IsDefinedBy          :SET [0:?] OF IfcRelDefinesByProperties FOR RelatedObjects;
  Declares             :SET OF IfcRelDeclares FOR RelatingContext;
 ENTITY IfcProject
END_ENTITY;
```

Code 16, IfcProject Inheritance Graph (BuildingSmart, 2016u)

7.1.7. IfcSite

IFC specification defines a site as “A defined area of land, possibly covered with water, on which the project construction is to be completed. A site may be used to erect, retrofit or tear down building(s), or for other construction related developments. (BuildingSmart, 2016ae)”. The attributes added by IfcSite entity to its superclass are optional attributes focused on the location of the site.

```
ENTITY IfcSite;
  RefLatitude     : OPTIONAL IfcCompoundPlaneAngleMeasure;
  RefLongitude    : OPTIONAL IfcCompoundPlaneAngleMeasure;
  RefElevation    : OPTIONAL IfcLengthMeasure;
  LandTitleNumber  : OPTIONAL IfcLabel;
  SiteAddress     : OPTIONAL IfcPostalAddress;
```

Code 17, IfcSite Attributes (BuildingSmart, 2016ae)

IfcCompoundPlaneAngleMeasure is a compound measure of plane angle in degrees, minutes, seconds, and optionally millionth-seconds of arc. IfcPostalAddress is the equivalent of a conventional postal address (Code 18).

```
ENTITY IfcPostalAddress
 ENTITY IfcAddress
  Purpose             :OPTIONAL IfcAddressTypeEnum;
  Description         :OPTIONAL IfcText;
  UserDefinedPurpose  :OPTIONAL IfcLabel;
 INVERSE
  OfPerson            :SET OF IfcPerson FOR Addresses;
  OfOrganization      :SET OF IfcOrganization FOR Addresses;
 ENTITY IfcPostalAddress
  InternalLocation    :OPTIONAL IfcLabel;
  AddressLines        :OPTIONAL LIST [1:?] OF IfcLabel;
  PostalBox           :OPTIONAL IfcLabel;
  Town                :OPTIONAL IfcLabel;
  Region              :OPTIONAL IfcLabel;
  PostalCode          :OPTIONAL IfcLabel;
  Country             :OPTIONAL IfcLabel;
END_ENTITY;
```

Code 18, IfcPostalAddress Inheritance Graph (BuildingSmart, 2016t)

7.1.7.1. IfcSite PROPERTY SETS

Pset_SiteCommon provides several properties that are applicable to UFPOR Environment class (Code 19).

```
Reference (P_SINGLEVALUE / IfcIdentifier) :

Reference ID for this specified type in this project (e.g. type 'A-1'). Used to store the non-
classification driven internal project type.


BuildableArea (P_SINGLEVALUE / IfcAreaMeasure) :

The area of site utilization expressed as a maximum value according to local building codes.


SiteCoverageRatio (P_SINGLEVALUE / IfcPositiveRatioMeasure) :

The ratio of the utilization, TotalArea / BuildableArea, expressed as a maximum value. The ratio
value may be used to derive BuildableArea.


FloorAreaRatio (P_SINGLEVALUE / IfcPositiveRatioMeasure) :

The ratio of all floor areas to the buildable area as the maximum floor area utilization of the site as
a maximum value according to local building codes.


BuildingHeightLimit (P_SINGLEVALUE / IfcPositiveLengthMeasure ) :

Allowed maximum height of buildings on this site - according to local building codes.


TotalArea (P_SINGLEVALUE / IfcAreaMeasure) :

Total planned area for the site. Used for programming the site space.[1]
```

Code 19, Pset_SiteCommon (BuildingSmart, 2016am)

These properties capture some of the more common attributes on the project site scale.

7.1.7.2. IfcSite QUANTITY SETS

IfcSite provides one quantity set Qto_SiteBaseQuantities with only two attributes (Code 19).

```
GrossPerimeter :
Perimeter of the site boundary, measured in horizontal projection.

GrossArea :
Gross area for this site, measured in horizontal projections.
```

Code 20, Qto_SiteBaseQuantities (BuildingSmart, 2016ar)

7.1.8. IfcBuilding

According to IFC specifications "A building represents a structure that provides shelter for its occupants or contents and stands in one place. The building is also used to provide a basic element within the spatial structure hierarchy for the components of a building project (together with site, storey, and space). (BuildingSmart, 2016e)" IfcBuilding adds the address of the building as well as the elevation value of the terrain to its superclass IfcSpatialStructureElement.

7.1.8.1. IfcBuilding PROPERTY SETS

IfcBuilding provides five additional property sets to the ones it inherits from its super classes. Amongst those Pset_BuildingCommon is the only property set with "planned" properties (Code 21).

```
Reference (P_SINGLEVALUE / IfcIdentifier)  : Reference ID for this specified type in this project (e.g.
type 'A-1'). Used to store the non-classification driven internal project type.


BuildingID (P_SINGLEVALUE / IfcIdentifier)  : A unique identifier assigned to a building. A temporary
identifier is initially assigned at the time of making a planning application. This temporary identifier is
changed to a permanent identifier when the building is registered into a statutory buildings and
properties database.


IsPermanentID (P_SINGLEVALUE / IfcBoolean) : Indicates whether the identity assigned to a building
is permanent (= TRUE) or temporary (=FALSE).


ConstructionMethod (P_SINGLEVALUE / IfcLabel) : The type of construction action to the building,
the project deals with, e.g. new construction, renovation, refurbishment, etc.


FireProtectionClass (P_SINGLEVALUE / IfcLabel) : Main fire protection class for the building which is
assigned from the fire protection classification table as given by the relevant national building code.


SprinklerProtection (P_SINGLEVALUE / IfcBoolean) : Indication whether this object is sprinkler
protected (TRUE) or not (FALSE).


SprinklerProtectionAutomatic (P_SINGLEVALUE / IfcBoolean) : Indication whether this object has an
automatic sprinkler protection (TRUE) or not (FALSE).


OccupancyType (P_SINGLEVALUE / IfcLabel) : Occupancy type for this object. It is defined
according to the presiding national building code.


GrossPlannedArea (P_SINGLEVALUE / IfcAreaMeasure ) : Total planned gross area for the building
Used for programming the building.


NetPlannedArea (P_SINGLEVALUE / IfcAreaMeasure)  : Total planned net area for the building Used
for programming the building.


NumberOfStoreys (P_SINGLEVALUE / IfcInteger ) : The number of storeys within a building.
Captured for those cases where the IfcBuildingStorey entity is not used. Note that if IfcBuilingStorey
is asserted and the number of storeys in a building can be determined from it, then this approach
should be used in preference to setting a property for the number of storeys.


YearOfConstruction (P_SINGLEVALUE / IfcLabel) : Year of construction of this building, including
expected year of completion.


YearOfLastRefurbishment (P_SINGLEVALUE / IfcLabel) : Year of last major refurbishment, or
reconstruction, of the building (applies to reconstruction works).


IsLandmarked (P_SINGLEVALUE / IfcLogical) : This builing is listed as a historic building (TRUE), or
not (FALSE), or unknown.
```

Code 21, Pset_BuildingCommon (BuildingSmart, 2016an)

The other property sets are Pset_UtilityConsumptionPHistory which captures consumption of utility resources; PsetOutsideDesignCriteria which provides properties that are used as the basis for calculating thermal loads at peak conditions; Pset_BuildingUse and Pset_BuildingUseAdjacent provide information about the real estate context of the building itself and the buildings adjacent to it.

7.1.8.2. IfcBuilding QUANTITY SETS

IfcBuilding provides a single standard quantity set called Qto_BuildingBaseQuantities which contains high level dimensional information about the building (Code 21).

Height (Q_LENGTH): Standard gross height of this building, from the top surface of the construction floor, to the top surface of the construction floor or roof above. Only provided if there is a constant height.

EavesHeigth (Q_LENGTH): Standard net height of this storey, from the top surface of the construction floor, to the bottom surface of the construction floor or roof above. Only provided if there is a constant height.

FootprintArea (Q_AREA): Gross area of the site covered by the building(s).

GrossFloorArea (Q_AREA): Sum of all gross areas of spaces within the building. It includes the area of construction elements within the building. May be provided in addition to the quantities of the spaces and the construction elements assigned to the building. In case of inconsistencies, the individual quantities of spaces and construction elements take precedence.

NetFloorArea (Q_AREA): Sum of all net areas of spaces within the building. It excludes the area of construction elements within the building. May be provided in addition to the quantities of the spaces assigned to the building. In case of inconsistencies, the individual quantities of spaces take precedence.

GrossVolume (Q_VOLUME) Sum of all gross volumes of spaces enclosed by the building. It includes the volumes of construction elements within the building. May be provided in addition to the quantities of the spaces and the construction elements assigend to the building. In case of inconsistencies, the individual quantities of spaces and construction elements take precedence.

NetVolume (Q_VOLUME): Sum of all net volumes of spaces enclosed by the building. It excludes the volumes of construction elements within the building. May be provided in addition to the quantities of the spaces assigned to the building. In case of inconsistencies, the individual quantities of spaces take precedence.

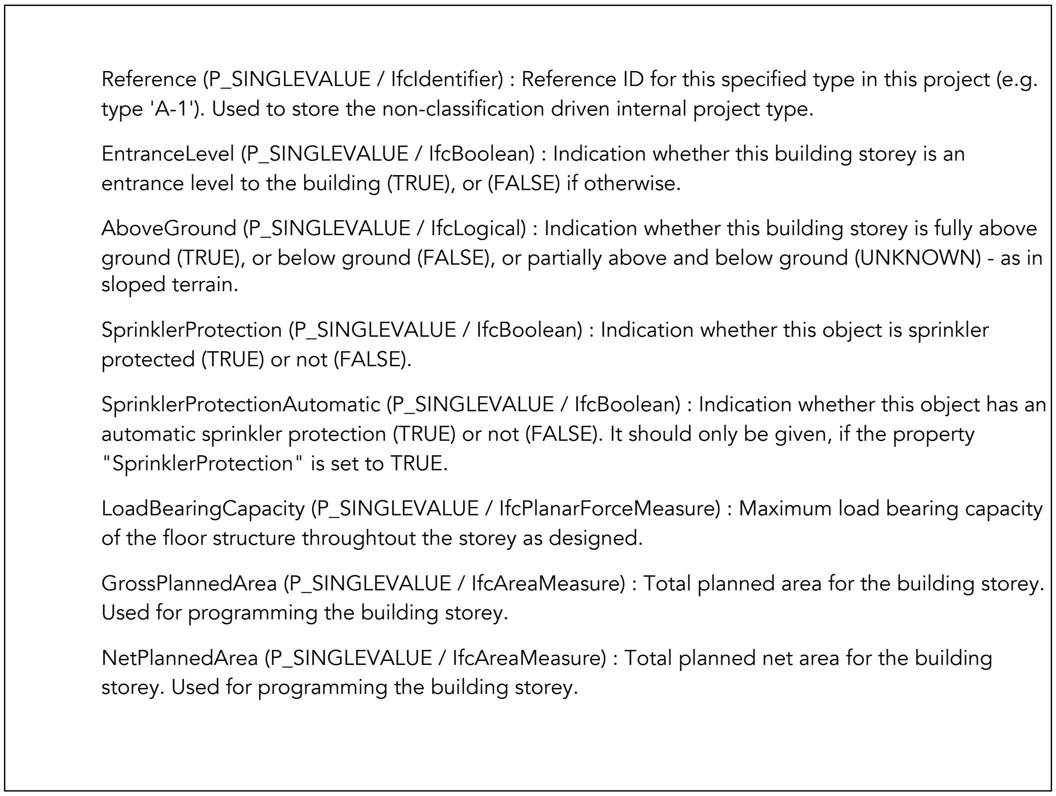
Code 22, Qto_BuildingBaseQuantities (BuildingSmart, 2016ap)

7.1.9. IfcBuildingStorey

IfcBuildingStroey breaks down a building into separate horizontal entities. Buildings with complex horizontal divisions can benefit from the composition type attribute that this entity inherits from IfcSpatialStructureElement by assigning any of COMPLEX, ELEMENT, or PARTIAL values to it. For example in split-level houses, two levels that are separated with a few stairs can be defined as PARTIAL stories. A single COMPLEX storey can host the partial storeys and connect them together.

7.1.9.1. IfcBuildingStorey PROPERTY SETS

IfcBuildingStorey provides similar property sets to IfcBuilding. Pset_BuildingStoreyCommon provides a collection of common building storey attributes and includes gross and net planned areas (Code 23).



Reference (P_SINGLEVALUE / IfcIdentifier) : Reference ID for this specified type in this project (e.g. type 'A-1'). Used to store the non-classification driven internal project type.

EntranceLevel (P_SINGLEVALUE / IfcBoolean) : Indication whether this building storey is an entrance level to the building (TRUE), or (FALSE) if otherwise.

AboveGround (P_SINGLEVALUE / IfcLogical) : Indication whether this building storey is fully above ground (TRUE), or below ground (FALSE), or partially above and below ground (UNKNOWN) - as in sloped terrain.

SprinklerProtection (P_SINGLEVALUE / IfcBoolean) : Indication whether this object is sprinkler protected (TRUE) or not (FALSE).

SprinklerProtectionAutomatic (P_SINGLEVALUE / IfcBoolean) : Indication whether this object has an automatic sprinkler protection (TRUE) or not (FALSE). It should only be given, if the property "SprinklerProtection" is set to TRUE.

LoadBearingCapacity (P_SINGLEVALUE / IfcPlanarForceMeasure) : Maximum load bearing capacity of the floor structure throughtout the storey as designed.

GrossPlannedArea (P_SINGLEVALUE / IfcAreaMeasure) : Total planned area for the building storey. Used for programming the building storey.

NetPlannedArea (P_SINGLEVALUE / IfcAreaMeasure) : Total planned net area for the building storey. Used for programming the building storey.

Code 23, Pset_BuildingStoreyCommon (BuildingSmart, 2016al)

7.1.9.2. IfcBuildingStorey QUANTITY SETS

IfcBuildingStorey provides a single quantity set called Qto_BuildingStoreyBaseQuantities similar to the one that IfcBuilding provides (Code, 24).

GrossHeight (Q_LENGTH) : Standard gross height of this storey, from the top surface of the construction floor, to the top surface of the construction floor or roof above. Only provided is there is a constant height.

NetHeigtht (Q_LENGTH) : Standard net height of this storey, from the top surface of the construction floor, to the bottom surface of the construction floor or roof above. Only provided is there is a constant height.

GrossPerimeter (Q_LENGTH) : Perimeter of the outer contour of the building story without taking interior slab openings into account.

GrossFloorArea (Q_AREA) : Sum of all gross areas of spaces within the building storey. It includes the area of construction elements within the building storey. May be provided in addition to the quantities of the spaces and the construction elements assigend to the storey. In case of inconsistencies, the individual quantities of spaces and construction elements take precedence.

NetFloorArea (Q_AREA) : Sum of all net areas of spaces within the building storey. It excludes the area of construction elements within the building storey. May be provided in addition to the quantities of the spaces assigend to the storey. In case of inconsistencies, the individual quantities of spaces take precedence.

GrossVolume (Q_VOLUME) : Sum of all gross volumes of spaces enclosed by the building storey. It includes the volumes of construction elements within the building storey. May be provided in addition to the quantities of the spaces and the construction elements assigend to the storey. In case of inconsistencies, the individual quantities of spaces and construction elements take precedence.

NetVolume (Q_VOLUME) : Sum of all net volumes of spaces enclosed by the building storey. It iexcludes the volumes of construction elements within the building storey. May be provided in addition to the quantities of the spaces assigend to the storey. In case of inconsistencies, the individual quantities of spaces take precedence.

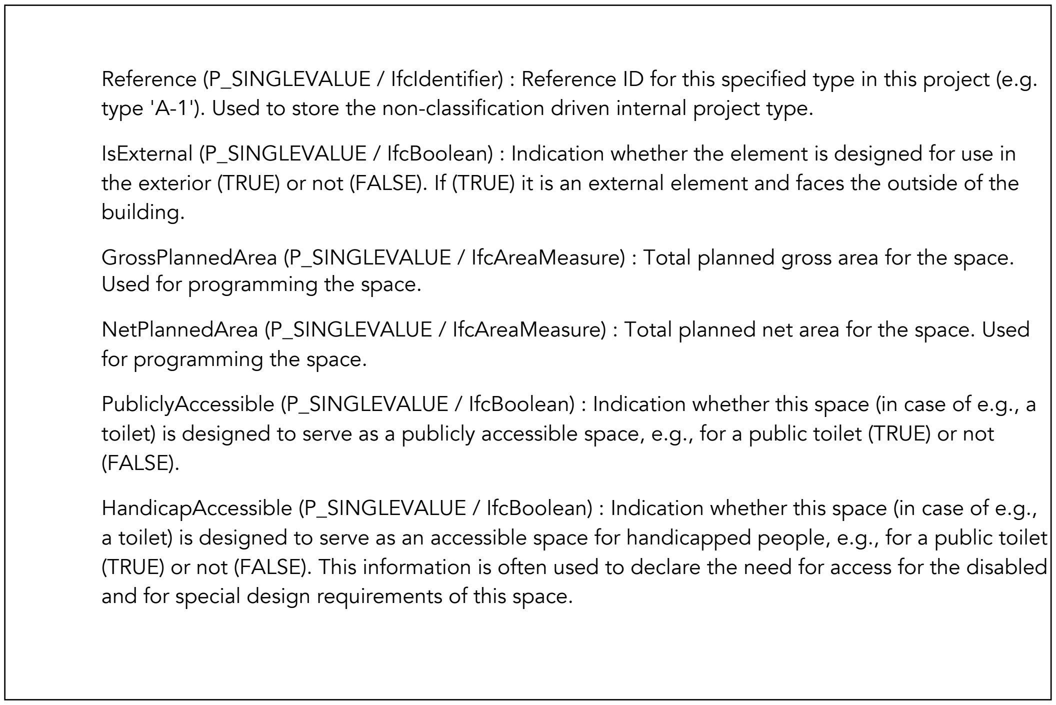
Code 24, Qto_BuildingStoreyBaseQuantities (BuildingSmart, 2016aq)

7.1.10. IfcSpace

IfcSpace is the last link in the chain of spatial elements. According to the definition it represents “an area or volume bounded actually or theoretically (BuildingSmart, 2016an)”. Spaces are also dedicated to a specific function within a building. Similar to the other subclasses of IfcSpatialStructureElement, spaces can have the composition type of COMPLEX, ELEMENT, or PARTIAL.

7.1.10.1. IfcSpace PROPERTY SETS

IfcSpace adds six new property sets on top of the ones that it inherits from IfcSpatialStructureElement. Amongst the new property sets Pset_SpaceCommon provides the most common properties for a space that includes NetPlannedArea and GrossPlannedArea (Code, 25).



Reference (P_SINGLEVALUE / IfcIdentifier) : Reference ID for this specified type in this project (e.g. type 'A-1'). Used to store the non-classification driven internal project type.

IsExternal (P_SINGLEVALUE / IfcBoolean) : Indication whether the element is designed for use in the exterior (TRUE) or not (FALSE). If (TRUE) it is an external element and faces the outside of the building.

GrossPlannedArea (P_SINGLEVALUE / IfcAreaMeasure) : Total planned gross area for the space. Used for programming the space.

NetPlannedArea (P_SINGLEVALUE / IfcAreaMeasure) : Total planned net area for the space. Used for programming the space.

PubliclyAccessible (P_SINGLEVALUE / IfcBoolean) : Indication whether this space (in case of e.g., a toilet) is designed to serve as a publicly accessible space, e.g., for a public toilet (TRUE) or not (FALSE).

HandicapAccessible (P_SINGLEVALUE / IfcBoolean) : Indication whether this space (in case of e.g., a toilet) is designed to serve as an accessible space for handicapped people, e.g., for a public toilet (TRUE) or not (FALSE). This information is often used to declare the need for access for the disabled and for special design requirements of this space.

Code 25, Pset_SpaceCommon (BuildingSmart, 2016an)

7.1.10.2. IfcSpace QUANTITY SETS

IfcSpace provides one single quantity set called Qto_SpaceBaseQuantities, which provides dimensional information about the space (Code, 26).

Height (Q_LENGTHt) : Total height (from base slab without flooring to ceiling without suspended ceiling) for this space (measured from top of slab below to bottom of slab above). To be provided only if the space has a constant height.

FinishCeilingHeight (Q_LENGTH) : Height of the suspended ceiling (from top of flooring to the bottom of the suspended ceiling). To be provided only if the space has a suspended ceiling with constant height.

FinishFloorHeight (Q_LENGTH) : Height of the flooring (from base slab without flooring to the flooring height). To be provided only if the space has a constant flooring height.

GrossPerimeter (Q_LENGTH) : Gross perimeter at the floor level of this space. It all sides of the space, including those parts of the perimeter that are created by virtual boundaries and openings (like doors).

NetPerimeter (Q_LENGTH) : Net perimeter at the floor level of this space. It excludes those parts of the perimeter that are created by by virtual boundaries and openings (like doors). It is the measurement used for skirting boards and may includes the perimeter of internal fixed objects like columns.

GrossFloorArea (Q_AREA) : Sum of all floor areas covered by the space. It includes the area covered by elementsinside the space (columns, inner walls, etc.) and excludes the area covered by wall claddings.

NetFloorArea (Q_AREA) : Sum of all usable floor areas covered by the space. It excludes the area covered by elements inside the space (columns, inner walls, built-in's etc.), slab openings, or other protruding elements. Varying heights are not taking into account (i.e. no reduction for areas under a minimum headroom).

GrossWallArea (Q_AREA) : Sum of all wall (and other vertically bounding elements, like columns) areas bounded by the space. It includes the area covered by elements inside the wall area (doors, windows, other openings, etc.).

NetWallArea (Q_AREA) : Sum of all wall (and other vertically bounding elements, like columns) areas bounded by the space. It excludes the area covered by elements inside the wall area (doors, windows, other openings, etc.).

GrossCeilingArea (Q_AREA) : Sum of all ceiling areas of the space. It includes the area covered by elementsinside the space (columns, inner walls, etc.). The ceiling area is the real (and not the projected) area (e.g. in case of sloped ceilings).

NetCeilingArea (Q_AREA) : Sum of all ceiling areas of the space. It excludes the area covered by elementsinside the space (columns, inner walls, etc.). The ceiling area is the real (and not the projected) area (e.g. in case of sloped ceilings).

GrossVolume (Q_VOLUME) : Gross volume enclosed by the space, including the volume of construction elements inside the space.

NetVolume (Q_VOLUME) : Net volume enclosed by the space, excluding the volume of construction elements inside the space.

Code 26, Qto_SpaceBaseQuantities (BuildingSmart, 2016as)

7.1.11. IfcSpatialZone

IfcSpatialZone provides the ability to create spatial groupings that do not fit into the building, floor, and space, hierarchy. IfcSpatialZone is a subclass of IfcSpatialElement and adds only one more attribute (PredefinedType) to its superclass (Code, 27).

```
TYPE IfcSpatialZoneTypeEnum = ENUMERATION OF (
 CONSTRUCTION,
 FIRESAFETY,
 LIGHTING,
 OCCUPANCY,
 SECURITY,
 THERMAL,
 TRANSPORT,
 VENTILATION,
 USERDEFINED,
 NOTDEFINED);
END_TYPE;
```

Code 27, IfcSpatialZoneTypeEnum (BuildingSmart, 2016ai)

IfcSpatialZone can be used to impose common properties on a group of spaces that may not be physically adjacent to each other but fall into the same category. While some of the predefined categories (e.g. Occupancy, and Security) may be useful to define groupings, "UserDefined" type may be more useful when none of the predefined types match the intent. If the PredefinedType is set as UserDefined, the ObjectType attribute (inherited from IfcObject) should be used to assign an arbitrary type to the instance of IfcSpatialZone.

It is worth mentioning that IfcSpatialZone should not be mistaken for IfcZone (BuildingSmart, 2016aj), which is a subclass of IfcSystem. While the former can carry geometrical attributes, the latter provides a more loose structure to create groupings. IfcZone is investigated later in this chapter.

7.1.12. IfcSpatialElementType

As mentioned in previous chapter, subclasses of IfcObjectType are used to capture common properties of their respective subclass of IfcObject. IfcSpatialElementType is a subclass of IfcObjectType, which is used to capture the common properties of IfcSpatialElement instances. Below is the inheritance graph for IfcSpatialElementType (Code, 28).

```
ENTITY IfcSpatialElementType
 ENTITY IfcRoot
  GlobalId              : IfcGloballyUniqueId;
  OwnerHistory          : OPTIONAL IfcOwnerHistory;
  Name                  : OPTIONAL IfcLabel;
  Description           : OPTIONAL IfcText;
 ENTITY IfcObjectDefinition
 INVERSE
  HasAssignments        : SET OF IfcRelAssigns FOR RelatedObjects;
  Nests                 : SET [0:1] OF IfcRelNests FOR RelatedObjects;
  IsNestedBy            : SET OF IfcRelNests FOR RelatingObject;
  HasContext            : SET [0:1] OF IfcRelDeclares FOR RelatedDefinitions;
  IsDecomposedBy        : SET OF IfcRelAggregates FOR RelatingObject;
  Decomposes            : SET [0:1] OF IfcRelAggregates FOR RelatedObjects;
  HasAssociations       : SET OF IfcRelAssociates FOR RelatedObjects;
 ENTITY IfcTypeObject
  ApplicableOccurrence  : OPTIONAL IfcIdentifier;
  HasPropertySets       : OPTIONAL SET [1:?] OF IfcPropertySetDefinition;
 INVERSE
  Types                 : SET [0:1] OF IfcRelDefinesByType FOR RelatingType;
 ENTITY IfcTypeProduct
  RepresentationMaps    : OPTIONAL LIST [1:?] OF IfcRepresentationMap;
  Tag                   : OPTIONAL IfcLabel;
 INVERSE
  ReferencedBy          : SET OF IfcRelAssignsToProduct FOR RelatingProduct;
 ENTITY IfcSpatialElementType
  ElementType           : OPTIONAL IfcLabel;
END_ENTITY;
```

Code 28, IfcSpatialElementType Inheritance Graph (BuildingSmart, 2016ag)

ApplicableOccurance is an important property for all subclasses of IfcTypeObject. ApplicableOccurance value specifies which subclasses of IfcObject can be enhanced. For example, IfcSpatialZoneType is to capture the common attributes for a group of IfcSpatialZone instances; therefore one must set the ApplicableType value in IfcSpatialZoneType as “IfcSpatialZone”.

IfcSpaceType and IfcSpatialZoneType are the only concrete subclasses of IfcSpatialElementType. IFC does not provide a type equivalent for IfcBuilding, and IfcBuildingStorey. The absence of the mentioned types creates inconvenience and redundancy in capturing properties in large projects where multiple buildings or building stories can share requirement.

7.1.13. CONCLUSION

IFC provides a wide range of entities and property sets that can be used to model the spatial structure of a project, but all those entities are designated to model the designed or built state. Some entities and property sets provide properties to capture information from the architectural programming phase, for example the property set Pset_SpaceCommon provides two properties GrossPlannedArea and NetPlannedArea to capture architectural programming information for a space. Such properties are attached to entities like IfcSpace that captures the designed and build state. Therefore they cannot be used independently until an instance of IfcSpace is placed in the project during the design phase.

The concept of Time is a key concept in UFPOR. It allows UFPOR Environment to gather information about the history of the project and the site (past), study the current state (present), and propose requirements for the future state. IFC provides a wide range of entities to model a designed spatial structure, but those entities do not cover different time spans. IFC provides solutions to attach correlated properties that each represents the same information but in different time phases (for example GrossPlannedArea and GrossArea) but all those properties are attached to entities that are added to the project in design phase. In this workflow such information cannot be captured until an instance of IfcSpace is added to the project.

Composition is another key concept in UFPOR. IFC provides several mechanisms to create a hierarchical spatial structure. The hierarchy starts with an instance of IfcProject at the root and can have leaves as small as building elements such as space walls and floors. The structure of the spatial tree must follow the IFC rules where predefined entities of IfcProject, IfcBuilding, IfcBuildingStorey, and IfcSpace can form a tree structure as long as the logical order for aggregation is maintained. IfcSpatialZone can be used to group the leaves together where they share common properties. IfcRelContainedInSpatialStructure expands the leaves of the hierarchy to include building elements such as walls, roofs, and floors, but there is no mechanism to extend the root of the tree to include entities beyond the project site such as region, city, and state.

7.2. UFPOR NEED

In representing a POR, UFPOR Need can take multiple roles. For example it can document clients goals and objectives in a hierarchical structure. UFPOR Need can also model the requirements that are directly imposed on instances of UFPOR Environments.

Area requirements are probably the most obvious examples. An instance of the Environment class should be in a certain area range to host the required activities. One can impose such instances of Need directly on a single property within the Environment class.

IfcConstraintResource, one of the schemas offered by Resource schemas, provides a complete set of classes to model different types if constraints and requirements. Through this schema constraints can be applied to the subclasses of IfcObjectDefinition and IfcPropertyDefinition. In addition to that, constraints can be directly placed on a property.

7.2.1. IfcConstraint

IfcConstraint is in the center of Constraint Schema. Code 29 is the inheritance graph for IfcConstraint.

```
ENTITY IfcConstraint
 ABSTRACT SUPERTYPE OF(ONEOF(IfcMetric, IfcObjective));
  Name : IfcLabel;
  Description : OPTIONAL IfcText;
  ConstraintGrade : IfcConstraintEnum;
  ConstraintSource : OPTIONAL IfcLabel;
  CreatingActor : OPTIONAL IfcActorSelect;
  CreationTime : OPTIONAL IfcDateTime;
  UserDefinedGrade : OPTIONAL IfcLabel;
 INVERSE
  HasExternalReferences
: SET OF IfcExternalReferenceRelationship FOR RelatedResourceObjects;
  PropertiesForConstraint
: SET OF IfcResourceConstraintRelationship FOR RelatingConstraint;
 WHERE
  WR11 : (ConstraintGrade <> IfcConstraintEnum.USERDEFINED) OR
((ConstraintGrade = IfcConstraintEnum.USERDEFINED) AND
EXISTS(SELF\IfcConstraint.UserDefinedGrade))(ConstraintGrade <>
IfcConstraintEnum.USERDEFINED) OR
((ConstraintGrade = IfcConstraintEnum.USERDEFINED) AND
EXISTS(SELF\IfcConstraint.UserDefinedGrade));
END_ENTITY;
```

Code 29, IfcConstraint Inheritance Diagram (BuildingSmart, 2016g)

Each constraint must have a name and optionally a description. Constraint grade should also be assigned to the constraint through one of the values represented by IfcConstraintEnum including HARD, SOFT, ADVISORY, USERDEFINED, and NOTDEFINED.

Constraint grade specifies the enforcement level of the constraint. HARD constraint must be satisfied; SOFT constraint should be satisfied whereas ADVISORY constraints do not posses any obligation power. If USERDEFINED is selected as the ConstraintGrade value, the UserDefinedGrade property should be assigned to the constraint. UserDefinedGrade should be defined by the user and is assigned to the constraint as a simple string value.

CreatingActor property specifies the relationship between the UFPOR NEED class and UFPOR PEOPLE class. Through this property one can specify who is imposing the constraint. CreatingActor is of IfcActorSelect type, which allows the CreatingActor type to be selected from an enumeration of IfcOrganization, IfcPerson, and IfcPersonAndOrganization.

In IFC constraints can be either qualitative or quantitative. IfcConstraint provides two subclasses, IfcMetric and IfcObjective. IfcMetric is used to model quantitative requirements whereas IfcObjective should be used to model qualitative requirements.

7.2.2. QUANTITATIVE REQUIREMENTS

IfcMetric is designed to capture quantitative requirements in IFC. Multiple instances of IfcMetric can be associated with one property to constrain the value of that property in different ways.

IfcMetric adds four additional attributes to IfcConstraint class (Code 30).

```
ENTITY IfcMetric
 SUBTYPE OF IfcConstraint;
  Benchmark          :IfcBenchmarkEnum;
  ValueSource        :OPTIONAL IfcLabel;
  DataValue          :IfcMetricValueSelect;
  ReferencePath      :OPTIONAL IfcReference;
END_ENTITY;
```

Code 30, IfcMetric Express Specification (BuildingSmart, 2016l)

7.2.2.1. BENCHMARK

Benchmark attribute is to capture the way that instances of IfcMetric restrict their respective objects or properties. An example can help to clarify this concept. Imagine a scenario where one need to impose an area constraint on a room. The constraint may impose a maximum allowed area on the space. Alternatively, one may want to impose a minimum allowed area on a space. Benchmark provides an enumeration of logical comparators to define the relationship between the constraint and the constrained object or property. Code 31 is a list of all possible values for Benchmark.

- **GREATERTHAN**: Identifies that a value must be greater than that set by the constraint.
- **GREATERTHANOREQUALTO**: Identifies that a value must be either greater than or equal to that set by the constraint.
- **LESSTHAN**: Identifies that a value must be less than that set by the constraint.
- **LESSTHANOREQUALTO**: Identifies that a value must be either less than or equal to that set by the constraint.
- **EQUALTO**: Identifies that a value must be equal to that set by the constraint.
- **NOTEQUALTO**: Identifies that a value must be not equal to that set by the constraint.
- **INCLUDES**: Identifies that an aggregation (set, list or table) must include the value (individual item) set by the constraint.
- **NOTINCLUDES**: Identifies that an aggregation (set, list or table) must not include the value (individual item) set by the constraint.
- **INCLUDEDIN**: Identifies that a value (individual item) must be included in the aggregation (set, list or table) set by the constraint.
- **NOTINCLUDEDIN**: Identifies that a value (individual item) must not be included in the aggregation (set, list or table) set by the constraint.

Code 31, IfcBenchmarkEnum (BuildingSmart, 2016d)

7.2.2.2. VALUESOURCE

ValueSource provides an optional field to capture a human readable description for the constraint.

7.2.2.3. DATAVALUE

DataValue captures the value provided by the constraint to be compared with the associated object. The type of DataValue can be selected from an enumeration of the types provided through IfcMetricValueSelect (Code 32).

```
TYPE IfcMetricValueSelect = SELECT (
   IfcMeasureWithUnit,
   IfcTable,
   IfcTimeSeries,
   IfcAppliedValue,
   IfcValue,
   IfcReference);
END_TYPE;
```

Code 32, IfcMetricValueSelect (BuildingSmart, 2016m)

Amongst the supported types for IfcMetricValueSelect, IfcMeasureWithUnit and IfcValue are applicable in capturing architectural constraints. The rest provide sophisticated channels to enforce more accurate constraint that are not typical to architectural programming.

The difference between IfcUnit and IfcMeasureWithUnit is an additional property in IfcMeasureWithUnit to capture the unit of the constraint. To avoid redundancy I only cover IfcMeasureWithUnit.

IfcMeasureWithUnit provides a way to associate a simple pair of value-unit measurement to the constraint (Code 33).

```
ENTITY IfcMeasureWithUnit;
   ValueComponent  : IfcValue;
   UnitComponent   : IfcUnit;
END_ENTITY;
```

Code 33, IfcMeasureWithUnit (BuildingSmart, 2016k)

Both ValueComponent and UnitComponent provide a large array of options to define the constraining values. IfcValue can be of one of the three types of IfcMeasureValue, IfcSimpleValue and IfcDerivedMeasureValue.

IfcDerivedMeasureValue is to capture values that are derived through mathematical equations and is useful in adding mechanical, electrical, and thermal constraint on building elements. IfcMeasureValue and IfcSimple value can be used to capture architectural constraints. IfcMeasureValue can be used to constrain any of the measurements shown in Code 34.

```
TYPE IfcMeasureValue = SELECT
  (IfcVolumeMeasure,
   IfcTimeMeasure,
   IfcThermodynamicTemperatureMeasure,
   IfcSolidAngleMeasure,
   IfcPositiveRatioMeasure,
   IfcRatioMeasure,
   IfcPositivePlaneAngleMeasure,
   IfcPlaneAngleMeasure,
   IfcParameterValue,
   IfcNumericMeasure,
   IfcMassMeasure,
   IfcPositiveLengthMeasure,
   IfcLengthMeasure,
   IfcElectricCurrentMeasure,
   IfcDescriptiveMeasure,
   IfcCountMeasure,
   IfcContextDependentMeasure,
   IfcAreaMeasure,
   IfcAmountOfSubstanceMeasure,
   IfcLuminousIntensityMeasure,
   IfcNormalisedRatioMeasure,
   IfcComplexNumber);
END_TYPE;
```

Code 34, IfcMeasureValue (BuildingSmart, 2016j)

IfcSimpleValue provides the basic data types.

```
TYPE IfcSimpleValue = SELECT
  (IfcInteger,
   IfcReal,
   IfcBoolean,
   IfcIdentifier,
   IfcText,
   IfcLabel,
   IfcLogical);
END_TYPE;
```

Code 35, IfcSimpleValue (BuildingSmart, 2016ad)

7.2.2.4. REFERENCEPATH

A constraint can either be associated directly with an object, or with a property of that object. When a constraint is associated with an object, it can be associated with a specific property of that object. ReferencePath allows to specify to which property the constraint is applied.

```
ENTITY IfcReference;
  TypeIdentifier          :OPTIONAL IfcIdentifier;
  AttributeIdentifier     :OPTIONAL IfcIdentifier;
  InstanceName            :OPTIONAL IfcLabel;
  ListPositions           :OPTIONAL LIST [1:?] OF INTEGER;
  InnerReference          :OPTIONAL IfcReference;
END_ENTITY;
```

Code 36, IfcReference (BuildingSmart, 2016v)

TypeIdentifier provides a way to specify the type of the attribute that the constraint is aiming at. AttributeIdentifier is to capture the name of the attribute. It is worth mentioning that AttributeIdentifier can identify both direct and reverse attributes. InstanceName and ListPosition are useful when the constraint is pointed toward a collection. The former identifies the right instance based on the name (If the instance has a “Name” attribute, otherwise the first STRING-base attribute is used), while the latter finds the right instance based on the location in the collection.

InnerReference can be used to create a chain of references if the constraint is placed on an indirect attribute of an IfcObject, for example an attribute of an associated object.

7.2.3. QUALITATIVE REQUIREMENTS

IfcObjective captures the qualitative requirements. IfcObjective has a more simple structure that IfcMetric since it is merely based on textual description of the requirements and does not deal with mathematical equation nor does with complex data types.

```
ENTITY IfcObjective
 ENTITY IfcConstraint
  Name                   : IfcLabel;
  Description            : OPTIONAL IfcText;
  ConstraintGrade        : IfcConstraintEnum;
  ConstraintSource       : OPTIONAL IfcLabel;
  CreatingActor          : OPTIONAL IfcActorSelect;
  CreationTime           : OPTIONAL IfcDateTime;
  UserDefinedGrade       : OPTIONAL IfcLabel;
 INVERSE
  HasExternalReferences  : SET OF IfcExternalReferenceRelationship FOR RelatedResourceObjects;
  PropertiesForConstraint: SET OF IfcResourceConstraintRelationship FOR RelatingConstraint;
 ENTITY IfcObjective
  BenchmarkValues        : OPTIONAL LIST [1:?] OF IfcConstraint;
  LogicalAggregator      : OPTIONAL IfcLogicalOperatorEnum;
  ObjectiveQualifier     : IfcObjectiveEnum;
  UserDefinedQualifier   : OPTIONAL IfcLabel;
END_ENTITY;
```

Code 37, IfcObjective Inheritance Graph (BuildingSmart, 2016o)

BenchmarkValues creates a way to aggregate other constraints under a qualitative constraint by providing a list of IfcConstraint. The list can include qualitative and quantitative constraints.

LogicalAggregator can be provided to complement the BenchMarkValues list. Through LogicalAggregator a logical operator can be attached to the BenchMarkValues list to clarify the relationship between all the constraints in the list.

```
TYPE IfcLogicalOperatorEnum = ENUMERATION OF (
   LOGICALAND,
   LOGICALOR,
   LOGICALXOR,
   LOGICALNOTAND,
   LOGICALNOTOR);
END_TYPE;
```

Code 38, IfcLogicalOperatorEnum (BuildingSmart, 2016i)

ObjectiveQualifier is an enumeration of the objective constraint type that clarifies the intention behind imposing the constraint by declaring the domain which imposed the constraint. This is an important attribute in modeling UFPOR Need instances through IFC since it enables us to clarify the intention of the constraint.

```
TYPE IfcObjectiveEnum = ENUMERATION OF (
 CODECOMPLIANCE,
 CODEWAIVER,
 DESIGNINTENT,
 EXTERNAL,
 HEALTHANDSAFETY,
 MERGECONFLICT,
 MODELVIEW,
 PARAMETER,
 REQUIREMENT,
 SPECIFICATION,
 TRIGGERCONDITION,
 USERDEFINED,
 NOTDEFINED);
END_TYPE;
```

Code 39, IfcObjectiveEnum (BuildingSmart, 2016p)

DesignIntent and Requirement are more applicable to architectural constraints. Here is a list of the definitions for all the constraint types provided by ObjectiveQualifier.

- **CODECOMPLIANCE:** A constraint whose objective is to ensure satisfaction of a code compliance provision.
- **CODEWAIVER:** A constraint whose objective is to identify an agreement that code compliance requirements (the waiver) will not be enforced.
- **DESIGNINTENT:** A constraint whose objective is to ensure satisfaction of a design intent provision.
- **EXTERNAL:** A constraint whose objective is to synchronize data with an external source such as a file.
- **HEALTHANDSAFETY:** A constraint whose objective is to ensure satisfaction of a health and safety provision.
- **MERGECONFLICT:** A constraint whose objective is to resolve a conflict such as merging data from multiple sources.
- **MODELVIEW:** A constraint whose objective is to ensure data conforms to a model view definition.
- **PARAMETER:** A constraint whose objective is to calculate a value based on other referenced values.
- **REQUIREMENT:** A constraint whose objective is to ensure satisfaction of a project requirement provision.

- **SPECIFICATION:** A constraint whose objective is to ensure satisfaction of a specification provision.
- **TRIGGERCONDITION:** A constraint whose objective is to indicate a limiting value beyond which the condition of an object requires a particular form of attention.

If the value of ObjectiveQualifier is set to USERDEFINED, UserDefinedQualifier attribute should be set for the constraint.

7.2.4. CONCLUSION

While one of the responsibilities of UFPOR Need is to impose direct qualitative and quantitative requirements on properties of the spatial structure, it plays a more fundamental role in forming an architectural programming model. UFPOR Need is one of the four main classes that represent an architectural programming document. It provides the required means to capture users' and clients' desire and mission. It also creates a hierarchical data structure where the mission statement for the project can be expanded to detailed requirements. It also helps to create the spatial structure by discovering spatial "needs" of the project activities.

IfcConstraint schema provides a comprehensive foundation to attach qualitative and quantitative requirements to a single property such as the space area or to an object such as an instance of IfcSpace. Neither IfcConstraint schema nor any other schema or entity in IFC provides a similar data structure to UFPOR Need where a hierarchical tree of needs can be modeled.

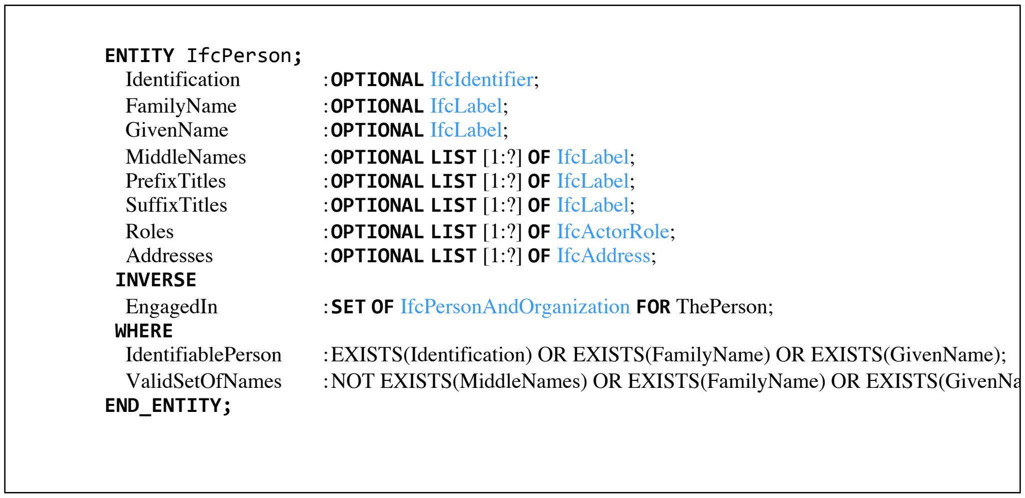
7.3. UFPOR PEOPLE

In the previous section I showed how instances of IfcConstraint can be associated with UFPOR People class through the CreatingActor attribute. In this section I go over different ways that one can create UFPOR People class through IFC.

IfcActorResource schema is dedicated to modeling a person or an organization (group of people). IfcActorResource schema is a relatively small schema with three main entities: IfcPerson, IfcOrganization, and IfcPersonAndOrganization. Besides these three main entities this schema provides other supporting entities such as IfcPostalAddress and IfcTelecomAddress that complement the three main entities of this schema.

7.3.1. IfcPerson

According to the definition IfcPerson represents an individual human being.



```
ENTITY IfcPerson;
  Identification        :OPTIONAL IfcIdentifier;
  FamilyName            :OPTIONAL IfcLabel;
  GivenName             :OPTIONAL IfcLabel;
  MiddleNames           :OPTIONAL LIST [1:?] OF IfcLabel;
  PrefixTitles          :OPTIONAL LIST [1:?] OF IfcLabel;
  SuffixTitles          :OPTIONAL LIST [1:?] OF IfcLabel;
  Roles                 :OPTIONAL LIST [1:?] OF IfcActorRole;
  Addresses             :OPTIONAL LIST [1:?] OF IfcAddress;
 INVERSE
  EngagedIn             :SET OF IfcPersonAndOrganization FOR ThePerson;
 WHERE
  IdentifiablePerson    :EXISTS(Identification) OR EXISTS(FamilyName) OR EXISTS(GivenName);
  ValidSetOfNames       :NOT EXISTS(MiddleNames) OR EXISTS(FamilyName) OR EXISTS(GivenNa
END_ENTITY;
```

Code 40, IfcPerson Inheritance Graph (BuildingSmart, 2016r)

Most of the attributes of IfcPerson are self-explanatory as they describe simple attributes of a person such as first and last name. EngagedIn attribute, which is an inverse relationship, allows IFC to connect the instances of IfcActor and IfcOrganization together.

The Roles attribute is an important one when it comes to modeling UFPOR People through IFC since it allows us to differentiate between different groups of people involved in a project.

7.3.2. IfcActorRole

IfcActorRole indicates the role that is performed by a person (IfcPerson) or an organization (IfcOrganization).

```
ENTITY IfcActorRole;
  Role          : IfcRoleEnum;
  UserDefinedRole : OPTIONAL IfcLabel;
  Description   : OPTIONAL IfcText;
INVERSE
HasExternalReference    :    SET OF IfcExternalReferenceRelationship FOR
                              RelatedResourceObjects;
WHERE
  WR1           : (Role <> IfcRoleEnum.USERDEFINED) OR ((Role =
                  IfcRoleEnum.USERDEFINED) AND
                  EXISTS(SELF.UserDefinedRole));
END_ENTITY;
```

Code 41, IfcActorRole Inheritance Graph (BuildingSmart, 2016b)

Role is the key attribute for IfcActorRole. The value for Role attribute is selected from IfcRoleEnum (Code 42).

```
TYPE IfcRoleEnum = ENUMERATION OF (
 SUPPLIER,
 MANUFACTURER,
 CONTRACTOR,
 SUBCONTRACTOR,
 ARCHITECT,
 STRUCTURALENGINEER,
 COSTENGINEER,
 CLIENT,
 BUILDINGOWNER,
 BUILDINGOPERATOR,
 MECHANICALENGINEER,
 ELECTRICALENGINEER,
 PROJECTMANAGER,
 FACILITIESMANAGER,
 CIVILENGINEER,
 COMMISSIONINGENGINEER,
 ENGINEER,
 OWNER,
 CONSULTANT,
 CONSTRUCTIONMANAGER,
 FIELDCONSTRUCTIONMANAGER,
 RESELLER,
 USERDEFINED);
END_TYPE;
```

Code 42, IfcRoleEnum (BuildingSmart, 2016ab)

IFC provides a long list of predefined roles. The last option on the list is USERDEFINED, if USEDEFINED is selected from the list, the optional UserDefinedRole on the instance of IfcActorRole should be defined.

7.3.3. IfcOrganization

IfcOrganization is similar to IfcActor with the difference of defining a group of actors.

```
ENTITY IfcOrganization
 ENTITY IfcOrganization
  Identification          :OPTIONAL IfcIdentifier;
  Name                    :IfcLabel;
  Description             :OPTIONAL IfcText;
  Roles                   :OPTIONAL LIST [1:?] OF IfcActorRole;
  Addresses               :OPTIONAL LIST [1:?] OF IfcAddress;
 INVERSE
  IsRelatedBy             :SET OF IfcOrganizationRelationship FOR RelatedOrganizations;
  Relates                 :SET OF IfcOrganizationRelationship FOR RelatingOrganization;
  Engages                 :SET OF IfcPersonAndOrganization FOR TheOrganization;
END_ENTITY;
```

Code 43, IfcOrganization Inheritance Graph (BuildingSmart, 2016q)

IfcOrganization provides two additional attributes in comparison with IfcActor. IsRelatedBy and Relates. They are two ends of an inverse relationship that can be established between different instances of IfcOrganization to associate them with each other. Notice that Engages attribute is the equivalent of EngagedIn in IfcActor.

7.3.4. IfcPersonAndOrganization

This entity provides a way to assign different roles to IfcActor by defining an organizational context for them. This entity is useful when an individual is evolved with the project in more than one capacity or through different organizations.

```
ENTITY IfcPersonAndOrganization
 ENTITY IfcPersonAndOrganization
  ThePerson            : IfcPerson;
  TheOrganization      : IfcOrganization;
  Roles                : OPTIONAL LIST [1:?] OF IfcActorRole;
END_ENTITY;
```

Code 44, IfcPersonAndOrganization Inheritance Graph (BuildingSmart, 2016s)

7.3.5. CONCLUSION

IfcActorResource schema provides the required means to model the individuals that are involved in the design and construction phase of a project. UFPOR People on the other hand, represents a more diverse spectrum. Not only the contractors and designers can be modeled through UFPOR People, the society where the project is placed, the potential users and the people who live within the project's vicinity are also subjects of UFPOR People.

UFPOR People provides a flexible structure where the users of a project can be categorized in a tree structure, for example in a library project the users are divided into the service providers and visitors. Visitors can be further categorized based on their age groups, interests and other attributes. The result is a tree structure of prototype individual groups whereas in IfcActorResource each entity is tied to a specific individual with a name and physical address.

IfcOrganization provides mechanisms to group individuals together by adding them to the same organization. This is quite different in comparison with UFPOR People groupings where the groups are formed based on their common attributes.

Generalizing IfcActorResource schema to the extent that covers the semantics of UFPOR People is a very challenging task. IfcActorResource schema is designed to model the individuals and organizations that are involved in design and construction of a project. On the other hand, the users are one of the main topics in UFPOR People, which cannot be modeled through IfcActorResource schema.

7.4. UFPOR ASSESSMENT

The requirements imposed through Constraint Resource schema need to go through an assessment process. In UFPOR, Assessment class is designated for this purpose. In IFC, Approval Resource schema is dedicated to manage the assessment process. It is worth mentioning that the quantitative requirements imposed through IfcMetric will be assessed through mathematic equations and Constraint Resource provides the infrastructure for evaluating the quantitative requirements.

While one of the main use cases for Approval Resource schema is in conjunction with Constraint Resource, there are cases for using this schema independently as well. For example the approval process of a plan, a proposal, or a change can be managed through the Approval Resource schema.

IfcApprovalResource schema contains three entities to manage the approval process. IfcApproval, IfcApprovalRelationship, and IfcResourceApprovalRelationship.

7.4.1. IfcApproval

IfcApproval is the entity at the heart of ApprovalResource schema (Code 45).

```
ENTITY IfcApproval;
  Identifier : OPTIONAL IfcIdentifier;
  Name : OPTIONAL IfcLabel;
  Description : OPTIONAL IfcText;
  TimeOfApproval : OPTIONAL IfcDateTime;
  Status : OPTIONAL IfcLabel;
  Level : OPTIONAL IfcLabel;
  Qualifier : OPTIONAL IfcText;
  RequestingApproval : OPTIONAL IfcActorSelect;
  GivingApproval : OPTIONAL IfcActorSelect;
 INVERSE
  HasExternalReferences
: SET OF IfcExternalReferenceRelationship FOR RelatedResourceObjects;
  ApprovedObjects : SET OF IfcRelAssociatesApproval FOR RelatingApproval;
  ApprovedResources : SET OF IfcResourceApprovalRelationship FOR RelatingApproval;
  IsRelatedWith : SET OF IfcApprovalRelationship FOR RelatedApprovals;
  Relates : SET OF IfcApprovalRelationship FOR RelatingApproval;
 WHERE
  HasIdentifierOrName : EXISTS (Identifier) OR EXISTS (Name);
END_ENTITY;
```

Code 45, IfcApproval Express Specification (BuildingSmart, 2016c)

Most of the direct attributes of this entity are self-explanatory and all of them are optional attributes. Identifier is dedicated to a computer interpretable Id by which the approval is known. “Status” attribute reflects the result of approval and can be any of the values of Requested, Processed, Approved, and Not Approved.

The RequestingApproval and GivingApproval attributes form the relationship between the Approval Resource and the Actor Resource schema and are the equivalent of the relationship between UFPOR Assessment and UFPOR People.

The relationship between instances of IfcApproval and the objects and properties in need of approval are formed through an inverse relationship called ApprovedObjects managed by IfcRelAssociatesApproval instances. Code 46 shows the inheritance graph for IfcRelAssociatesApproval.

```
ENTITY IfcRelAssociatesApproval
 ENTITY IfcRoot
  GlobalId            :IfcGloballyUniqueId;
  OwnerHistory        :OPTIONAL IfcOwnerHistory;
  Name                :OPTIONAL IfcLabel;
  Description         :OPTIONAL IfcText;
 ENTITY IfcRelationship
 ENTITY IfcRelAssociates
  RelatedObjects      :SET [1:?] OF IfcDefinitionSelect;
 ENTITY IfcRelAssociatesApproval
  RelatingApproval    :IfcApproval;
END_ENTITY;
```

Code 46, IfcRelAssociatesApproval Inheritance Graph (BuildingSmart, 2016x)

IfcDefinitionSelect includes all the subtypes of IfcObjectDefinition and IfcPropertyDefinition. However that does not include the Constraint Resource Schema. To attach instance of IfcApproval to any entity from the Resource schemas including Constraint Resource schema, instances of IfcResourceApprovalRelationship can be attached to IfcApproval through ApprovedResources inverse attribute. Through IfcResourceApprovalRelationship instances of IfcApproval can be attached to any of the entities from IfcResourceObjectSelect including IfcConstraint. Below is the full list of all the entities included in IfcResourceObjectSelect.

```
TYPE IfcResourceObjectSelect = SELECT (
 IfcPropertyAbstraction,
 IfcPhysicalQuantity,
 IfcAppliedValue,
 IfcContextDependentUnit,
 IfcConversionBasedUnit,
 IfcProfileDef,
 IfcActorRole,
 IfcApproval,
 IfcConstraint,
 IfcTimeSeries,
 IfcMaterialDefinition,
 IfcPerson,
 IfcPersonAndOrganization,
 IfcOrganization,
 IfcExternalReference,
 IfcExternalInformation);
END_TYPE;
```

Code 47, IfcResourceObjectSelect Express Specifications (BuildingSmart, 2016aa)

7.4.2. CONCLUSION

UFPOR Assessment provides the mechanisms to check the design versus the specific quantitative and qualitative requirements from the architectural programming document. Comparing to other UFPOR classes, UFPOR Assessment has a more simple structure. UFPOR Assessment neither implements the composition pattern since it does not need to form a tree structure nor covers different time spans.

IfcApprovalResource schema provides the required data structures that are needed to model UFPOR Assessment through IFC. That being said, IfcApprovalResource is limited by IfcActorResource schema. In previous section I discussed how IfcActorResource schema does not provide the semantics needed to model UFPOR People in IFC. Without the ability to model the UFPOR People in IFC one cannot assign approving roles to the intended individuals correctly, therefore with the current state of IFC IfcApprovalResource provides partial mechanisms to model UFPOR Assessment.

7.5. UFPOR ACTIVITY

In comparison with other UFPOR classes, Activity is perhaps the most exclusive one to PORs. Activity is an abstract concept that clarifies the intersection of UFPOR People and Environment. UFPOR Environment represents the final product of studying the activities. In that sense activities are more about how the product is utilized and not directly about the product itself. That might be why it is rather challenging to find appropriate data structures to model UFPOR Activities in IFC. I distinguished three IFC entities that can potentially be used in modeling UFPOR Activities. IfcProcess, IfcSpatialZone, and IfcZone.

7.5.1. IfcProcess

According to IFC (2016ak), IfcProcess represents "one individual activity or event, that is ordered in time, that has sequence relationships with other processes, which transforms input into output, and may connect to other processes through input output relationships. An IfcProcess can be an activity (or task), or an event. It takes usually place in building construction with the intent of designing, costing, acquiring, constructing, or maintaining products or other and similar tasks or procedures".

IfcProcess is conceptually similar to the definition of UFPOR Activity. The main differentiator is the instances of IfcProcess form a sequence of processes where each process consumes the outcome of another process and produces incomes for the next process. The relationship limits IfcProcess to the activities that occur during the construction phase and makes it a long stretch to use it for functional activities that occur in the building by users. Besides that, using the same data structure to model construction activities and how the users would use the facility creates semantic ambiguity for the project.

7.5.2. IfcSpatialZone

According to the Ifc specifications (2016ak) “A spatial zone is a non-hierarchical and potentially overlapping decomposition of the project under some functional consideration. A spatial zone might be used to represent a thermal zone, a construction zone, a lighting zone, a usable area zone. A spatial zone might have its independent placement and shape representation.”

Although IfcSpatialZone is a subclass of IfcSpatialElement and based on the definition Activities are independent entities, conceptually they are very close to each other. The composition of the activities can be accomplished through a hierarchical relationship similar to spaces. IfcRelAggregates will be used to create the hierarchy.

The downside is that IfcRelAggregates is semantically a strong relationship that is used to define composition/decomposition relationship. According to the IFC definition (BuildingSmart, 2016ak) : “Decompositions imply a dependency, implying that the whole depends on the definition of the parts and the parts depend on the existence of the whole.” The relationships between activities should be more flexible than a composition relationship.

The most significant downside of using IfcSpatialZone is the semantic ambiguity that would be introduced to the project. UFPOR activities are abstract concepts that are not bound to any physical form and shape. Using IfcSpatialZone implies the presence of shape and form, which does not fit UFPOR Activities. Besides that, one may need to apply instances of IfcSpatialZone to fully represent UFPOR Environment in IFC, using the same entity with two different semantics creates significant confusion and compromises the integrity of the data model.

7.5.3. IfcZone

In contrast with IfcSpatialZone, IfcZone provides a more loose association between different zones and does not provide geometrical representation attributes. According to the definition (2016aj)"A zone is a group of spaces, partial spaces or other zones". Zone structures may not be hierarchical (in contrary to the spatial structure of a project). That means one individual IfcSpace may be associated with zero, one, or several instances of IfcZone. IfcSpace instances are grouped into an IfcZone by using the objectified relationship IfcRelAssignsToGroup as specified at the super type IfcGroup."

While the data structure provided by IfcZone seems like a good fit to represent the required data, the entity definition still implies a different concept than UFPOR activity. UFPOR Activity is an independent entity from the spatial structure of the project. IfcZone provides a flexible mechanism to group entities of the spatial structure together.

7.5.4. CONCLUSION

The concept of activity is a key concept in the reviewed expert models. UFPOR Activity provides the mechanisms that are needed to model not only what will take place in a facility, but also why and how the users of a facility would use it. It also provides the required data structure to form a hierarchical tree of the activities.

None of the IFC entities that were discussed in this chapter semantically match UFPOR Activity. IfcZone and IfcSpatialZone are both part of the spatial structure, which in UFPOR is an independent concept from UFPOR Activity. Using those entities will create semantic confusion. IfcProcess is dedicated to model the activities that are performed during the construction phase by the individuals and parties involved in the construction phase and not the users of the project. IfcProcess does not provide the required attributes nor the semantics that are needed to model UFPOR Activity.

7.6. CONCLUSION

IFC provides comprehensive schemas to model not only the designed and built state of a facility, but also the organizations and individuals who are involved in the construction phase and their activities. IFC is less comprehensive in modeling the pre-design phase including the architectural programming phase.

UFPOR People includes the society, users of a facility and the individuals who are involved in the design and construction of the project. The current state of IFC only supports modeling of individuals and organizations who are the direct participants in the design and construction phase.

UFPOR Need creates hierarchical tree structures to gather information on areas such as users' needs, spatial needs of users' activities, and the mission statement and goals of the clients. The current state of IFC only supports modeling of the detailed and direct requirements that are mapped to a specific object (e.g. a specific space) or a specific attribute on that object (e.g. space area)

UFPOR Activity is not supported on any level in the current state of IFC. UFPOR Activity is a key aspect of architectural programming phase without which understanding the relationships between the users and the facility is near impossible.

UFPOR Environment captures information on the existing and proposed spatial structures related to the project. The current state of IFC provides comprehensive data structures to create a spatial tree structure of the designed facility. The same schemas can be used to document the existing structures within or around the project site as well. The current state of IFC does not provide separate schemas to create a spatial structure for the desired state that reflects the result of architectural programming phase.

The concept of Time is a key concept in UFPOR where the base classes should be able to capture information on different time spans such as present, past, and future. The current state of IFC can support a single state and falls short in supporting multiple states within the same model. To support the implementation of UFPOR such limitations should be removed.

The current state of IFC provides several techniques to enhance the entities through dynamic and static property sets. Therefore where entities exist that semantically match the UFPOR classes, their attribute scope can be enhanced to accommodate the ones of UFPOR classes.

8. UFPOR SOFTWARE IMPLEMENTATION

In the previous chapter the limitations of IFC in implementing UFPOR were discussed. IFC provides limited support for UFPOR Need and UFPOR Environment and does not provide any applicable support for UFPOR Activity and UFPOR People. Besides limited support for UFPOR base classes, the concept of Time that is fundamental in UFPOR is not supported by IFC. For all these limitations the current state of IFC does not fully support UFPOR. This chapter takes an empirical approach to explore the practical capabilities of IFC in supporting UFPOR on limited basis.

A prototype application was implemented to explore the capabilities of IFC in supporting UFPOR Need and UFPOR Environment. Implementing a prototype helped to answer three questions. First, with the current limited support for UFPOR in IFC, to what extent a POR can be presented in IFC? To answer this question, a software system was devised to partially implement the IFC compatible UFPOR model. Through logical argumentation the semantic and structural limitations of IFC were shown. Several tests were implemented to explore the robustness of the system.

Second, can the current status of UFPOR support in IFC provide the basis for a user facing application? To answer this question, I created an application that allows users to enter project information along with their requirements into the data model.

Third, how would the IFC compatible UFPOR support real projects? To answer this question, two programming documents produced by two well-known architecture firms were tested against the IFC compatible UFPOR model.

8.1. INTRODUCING UFPOR-IFC

Throughout this chapter, the acronym UFPOR-IFC is used to describe the subset of UFPOR that is supported by IFC. UFPOR-IFC has two main parts, first, the subset of UFPOR Need that represents direct qualitative and quantitative requirements on UFPOR Environment attributes, and second, the subset of UFPOR Environment that represent the desired state of the environment tree structure limited with IfcProject at the top (root of the tree structure), and the individual spaces (represented by IfcSpace) at the bottom.

The second part of UFPOR-IFC is based on the entities that form the spatial structure in IFC. These entities include IfcProject, subclasses of IfcSpatialElement (including IfcBuilding, IfcBuildingStorey, IfcSpatialZone, IfcSite and IfcSpace) and subclasses of IfcSpatialElementType (including IfcSpatialZoneType, and IfcSpaceType). All these entities were analyzed in detail in chapter 7.

The first part of UFPOR-IFC is based on the subclasses of IfcConstraints including IfcMetric and IfcObjective which were covered in chapter 7 as well.

In chapter 7 it was concluded that current state of IFC does not support explicit distinction between the desired/programmed and designed/built states; however UFPOR-IFC should represent the desired/programmed state. In UFPOR-IFC the desired/programmed state is implied by having non-filled properties on spatial structure entities that are bound by requirements. For example a POR may include maximum and minimum area limitations for a particular space. To model such space in UFPOR-IFC, said space is added to the spatial structure with a property representing the area. The area property will not exhibit any value, but is constraint by two instances of IfcMetric entities. The designed/build value is added to the IFC file during design/construction. At that point one can assess the fulfillment of the imposed requirements.

8.2. UFPOR-IFC IMPLEMENTATION

8.2.1. IMPLEMENTATION PLATFORM

IFC entities were initially developed based on a collection of open sourced C++ classes. The current version of IFC documented in EXPRESS is technology-independent and can be implemented in any programming language supporting object-oriented programming principles.

I chose Java programming language to implement the BIM compatible UFPOR because of my personal familiarity with the technology, its strong OOP features, and its popularity within the software development community.

Java is s general-purpose computer language that supports object-oriented, and multi-thread programming. Java was developed and released in 1990s by Sun Microsystem. Much of the syntax is driven from C and C++ programming languages (Bloch, 2001).

Over the next three sections, I describe the three steps required to implement the BIM compatible version of UFPOR-IFC in Java. The first section goes over creating the data models, the second section describes how to store the content of the data models in a database, the third section explains how to extract an IFC physical output from the database and the data models, and the fourth section describes how to test the whole system.

8.2.2. DATA MODELS

The first step is re-creating the required IFC schemas through Java classes. I replicated each IFC entity to a Java class and recreated the hierarchical structure required to implement the low level entities. Code 48 shows IfcRoot EXPRESS specification as an example.

```
ENTITY IfcRoot
 ENTITY IfcRoot
  GlobalId            : IfcGloballyUniqueId;
  OwnerHistory        : OPTIONAL IfcOwnerHistory;
  Name                : OPTIONAL IfcLabel;
  Description         : OPTIONAL IfcText;
END_ENTITY;
```

Code 48, IfcRoot Inheritance Graph (BuildingSmart, 2016ac)

Code 49 is the Java implementation of IfcRoot. The class methods are eliminated from the Code to increase code readability.

```
public abstract class IfcDecRoot {
        protected IfcDecGloballyUniqueId globalId;
        private IfcDecOwnerHistory ownerHistory;
        private IfcDecLabel nameText;
        private IfcDecText descriptionText;
}
```

Code 49, IfcRoot Java Implementation

IfcRoot example shows that IFC entities can be implemented using Java classes without compromising the specification.

8.2.3. OBJECT TRAVERSAL

In chapter 7 it was shown how IfcProject provides an entering point to the IFC object graph. To manage the model correctly and produce the IFC Physical file, it is critical to traverse from the instance of IfcProject to every IFC entity in the project. For that purpose the concept of “related objects” was introduced and implemented by the Java classes representing UFPOR-IFC. This capability enabled every UFPOR-IFC Java class to report back all of their relating objects. By iterating the same method, a tree-like structure can be created that includes all entities forming the project. Code 50 shows how IfcProject reports its related objects.

```
@Override
public ArrayList<IfcFileObject> getRelatedObjects() {
    ArrayList<IfcFileObject> results = super.getRelatedObjects();

    results.add(getUnitsInContext());
    results.addAll(getIsDefinedBy());

    ArrayList<IfcDecSpaceType> spaceTypes = null;
    if (getSpaceTypes() != null && getSpaceTypes().size() > 0) {
        try {
            spaceTypes =EnvironmentServiceImpl.getSpaceTypeByKey(getSpaceTypes());
            for (IfcDecSpaceType spaceType : spaceTypes) {
                addDeclare(spaceType);
            }
        } catch (NotLoggedInException e) {
            e.printStackTrace();
        }
    }
    results.add(declaresObject);
    return results;
}
```

Code 50, Object Traversal in Java UFPOR-IFC Classes

First, the list of relating objects are requested from the super class, in this case IfcContext is the super class. After attaining the list of relating objects from the super class, The content of UnitsInContext attribute (instance of IfcUnitAssignment) and the content of IsDefinedBy attribute (instance of IfcRelDefinedByProperties) are added to the list. In the next step all the instances of IfcSpaceType created in the project are retrieved from the database and added to the list. Instances of IfcSpaceType are only stored by their keys. This allows to reuse the instances of IfcSpaceType in more than one project; it also helps to keep the data models in a reasonable scale.

8.2.4. IFC PHYSICAL

To generate the IFC physical file, each IFC object needs to be translated into a textual descriptive string. Code 51 is an example showing the textual description of an IfcProject instance.

```
#3= IFCPROJECT ('340ryhzQ15d8rqO7t8Yu1r', $, TestProject, $, $, TestProject Long, $, $, #6);
```

Code 51, IfcProject IFC Physical

The first element is the item number; this number may be different every time the IFC physical file is generated and is used internally within the document to reference the object. All the values after the entity name within the parenthesis are separated with commas and come in the same order that are defined in the EXPRESS specification. The first attribute is the globalId attribute which is inherited from IfcRoot. The dollar sign is used for the void optional values. References to other objects are stored via the item numbers (e.g., #6).

Generating the IFC physical file for IFC compatible UFPOR requires two steps. First, A string template for each entity is created. The template is modified at runtime by using Java string library to embed the entity attribute values. Code 52 shows a few examples of the templates for IfcProject, IfcUnit and IfcPropertySingleValue.

```
"IFCPROJECT ('%1$s', %2$s, %3$s, %4$s, %5$s, %6$s, %7$s, %8$s, %9$s);";
"IFCSIUNIT(%1$s, %2$s, %3$s, %4$s);";
"IFCPROPERTYSINGLEVALUE(%1$s, %2$s, %3$s, %4$s);";
```

Code 52, IFC Physical File Templates

Before fill out the template for the entities, an element number should be assigned to every entity. Without knowing the element number for the entities, filling out the value for attributes that point to other entities is not possible, therefore assigning an element number to every entity is a prerequisite of creating IFC physical file.

The algorithm to assign a number to each object is based on a collection of all the entities related to the project, the algorithm starts by adding the instance of IfcProject to the collection. The collection then assigns a number to the instance of IfcProject that represents the project (#1). The algorithm proceeds by traversing to all the other objects through retrieving the related objects from each object. A number is assigned to each object as they are added to the collection.

Now that each item has a number ,other entities can referenced them by searching the collection. The IFC physical string for each entity can be generated by prepending their number followed by "=" to the beginning of their filled out template.

8.2.5. PERSISTENCE LAYER

There are several technologies available to assist with the persistence of the content of Java classes into a database. These technologies create a separation layer between the database and Java classes and facilitate the interaction. JDO (Java Data Objects) is one of these technologies developed by Sun Microsystems (Russell, 2010) that was used for UFPOR-IFC. According to Oracle “The Java Data Objects architecture defines a standard API to data contained in local storage systems and heterogeneous enterprise information systems, such as ERP, mainframe transaction processing and database systems. The architecture also refers to the Connector architecture which defines a set of portable, scalable, secure, and transactional mechanisms for the integration of EIS with an application server. (Oracle, 2009)”

After adding the required libraries to the project, the data fields of the Java classes were annotated to fulfill JDO requirements. The annotations determine whether the value of a field is persisted by JDO or not. Code 53 is the annotated version of IfcRoot. Variables annotated by “Persistent” annotation will be stored to the database.

```
@PersistenceCapable
@Inheritance(strategy = InheritanceStrategy.SUBCLASS_TABLE)
public abstract class IfcDecRoot {
        @Persistent
        protected IfcDecGloballyUniqueId globalId;
        @Persistent
        private IfcDecOwnerHistory ownerHistory;
        @Persistent
        private IfcDecLabel nameText;
        @Persistent
        private IfcDecText descriptionText;
}
```

Code 53, IfcRoot with JDO Annotations

More details on JDO are outside of the scope of this research and irrelevant to the findings. By using JDO the application is capable of working with a wide range of database/data storage technologies. Google App Engine was selected for the database layer . According to Google "Google App Engine lets you build and run applications on Google's infrastructure. App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change. With App Engine, there are no servers for you to maintain. You simply upload your application and it's ready to go. (Zahariev, 2009)"

8.2.6. UNIT TESTING

Before evaluating the performance of the system in handling real POR data, One should examine that all parts of the system including storing data in Java classes, storing them in Google App Engine through JDO, and creating IFC physical file from the retrieved data work as expected.

Unit testing is a software testing methodology that runs units of the system with control data and checks the outcome against the control data to assure the expected results were produced (Hamill, 2004). The definition of “unit” is context dependent and depends on the objectives of the test. There are several libraries and frameworks that work with Java, JDO, and Google App Engine. I used JUnit to handle the nuances of testing. I defined three test scenarios that I explain over the next three sections (Cheon & Leavens, 2002).

8.2.6.1. PROJECT

The objective for this test was to determine if the system can correctly perform the following steps.

- Create an instance of IfcProject.
- Add area units to the created IfcProject.
- Add area requirement to the created IfcProject.
- Store the instance in Google App Engine.
- Retrieve the instance.
- Produce the IFC physical file from the retrieved instance.

Code 54 shows the content of an instance of IfcClientProject that was passed to the test as the control data.

```
public static IfcClientProject getNewTestProject() {
 IfcClientProject testProject = new IfcClientProject()
 testProject.setMinArea(200);
testProject.setMaxArea(400);
testProject.setName("TestProject");
testProject.setLongName(new IfcClientLabel("TestProject" + " Long"));


IfcClientSIUnit.IfcSIUnitName unitName1 = IfcClientSIUnit.IfcSIUnitName.valueOf("METRE");
IfcClientSIUnit lengthUnit = new IfcClientSIUnit(IfcClientNamedUnit.IfcUnitEnum.LENGTHUNIT, null,
unitName1);
testProject.getUnitsInContext().addUnit(lengthUnit);
IfcClientSIUnit.IfcSIUnitName unitName2 =
IfcClientSIUnit.IfcSIUnitName.valueOf("SQUARE_METRE");
IfcClientSIUnit areaUnit = new IfcClientSIUnit(IfcClientNamedUnit.IfcUnitEnum.AREAUNIT, null,
unitName2);
testProject.getUnitsInContext().addUnit(areaUnit);


IfcClientSIUnit.IfcSIUnitName unitName3 = IfcClientSIUnit.IfcSIUnitName.valueOf("CUBIC_METRE");
IfcClientSIUnit volumeUnit =
new IfcClientSIUnit(IfcClientNamedUnit.IfcUnitEnum.VOLUMEUNIT, null, unitName3);
testProject.getUnitsInContext().addUnit(volumeUnit);


return testProject;
}
```

Code 54, Project Unit Test Input

The project contains name, longname, maximum and minimum area, as well as the SI units used in the project. The system could successfully perform all the defined objectives. Appendix 2 is the IFC physical outcome of the test that was produced in the final step.

8.2.6.2. SPACE TYPE

In this unit test the objectives from the previous unit test were expanded to include the following item.

- Create an instance of IfcSpaceType.
- Attach a property set to the created space type.
- Attach a quantity set to the created space type.
- Attach properties to the defined property set.
- Attach quantities to the defined quantity set.
- Attach constraints to the defined properties and quantities.
- Persist all the objects in Google App Engine.
- Retrieve the project.
- Create a physical IFC file from the retrieved project.
- Assert all the values from the IFC outcome match the control data.

The test starts by grabbing the project that was created in the previous unit test and then attaching an instance on IfcSpaceType along with the quantity and property sets and their constraints.

The system could successfully perform the test and the retrieved values from the IFC physical file matched the control data. Appendix 3 is the IFC physical output from the test.

8.2.6.3. SPACE INSTANCE

In this scenario the objectives from the previous test were expanded by adding an instance of IfcSpace to the project. Below is the list of additional objectives for this test scenario.

- Create an instance of IfcSpace.
- Attach the instance of IfcSpace to the instance of IfcSpaceType.
- Attach the instance of IfcSpace to the instance of IfcProject.
- Persist the project through Google App Engine.
- Retrieve the project for Google App Engine.
- Check the project and IfcSpace instance against the control data.

The test starts by retrieving the project created in the previous section. An instance of IfcSpace was created and was enhanced by attaching it to the instance of IfcSpaceType. The IfcSpace instance was then attached to the project. The project was persisted to the Google App Engine and then retrieved. The IFC physical output produced from the retrieved project matched the control data. Appendix 4 is the IFC physical file that was produced as the output of this test.

8.2.6.4. CONCLUSION

By following the unit testing principals I could prove that the devised system is capable of representing UFPOR-IFC entities and producing IFC physical files from them. By generating the IFC physical files in each test scenario, I also produced three examples showing how IFC physical file can handle UFPOR-IFC information.

8.3. USER INTERFACE PROTOTYPE

8.3.1. IMPLEMENTATION PLATFORM

In previous section by using unit testing techniques I showed that the devised system could successfully handle the creation of an IFC physical file that carries UFPOR-IFC information. In this section I move the focus toward usability. The main objective for this section is to answer this question: Can the devised system be used in user facing systems where the user can interact with the UFPOR-IFC information stored in IFC entities? To answer the question, a graphical user interface was developed to communicate with the system that I tested in previous section.

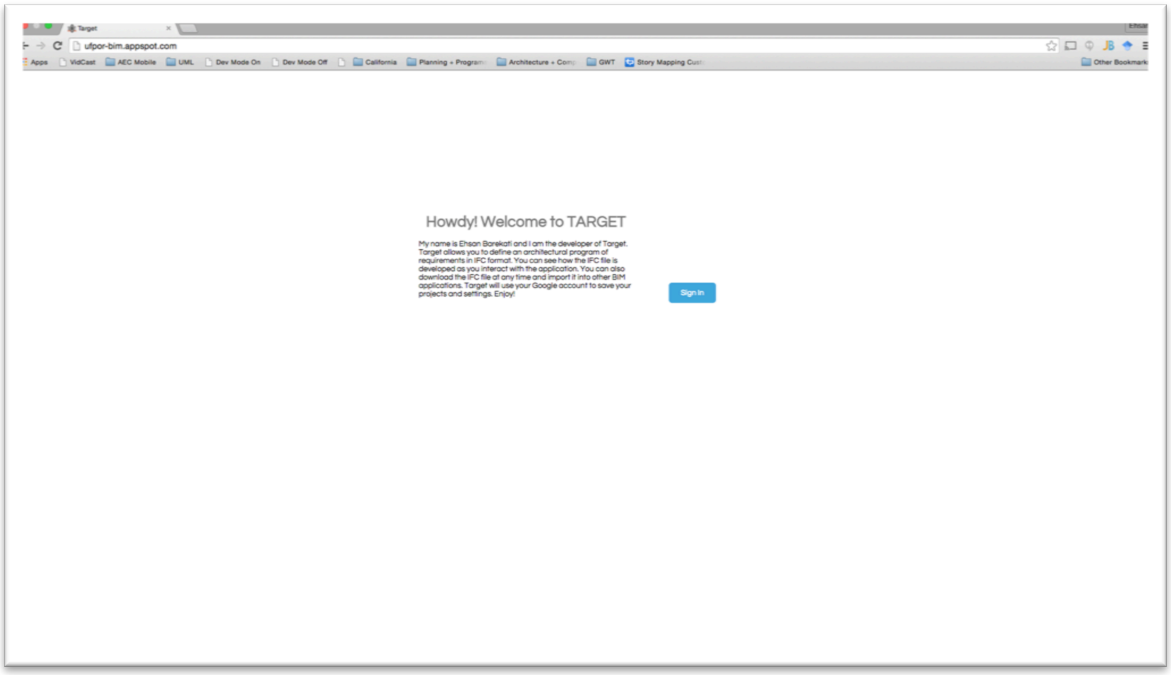
Any user interface framework that supports HTTP protocol can be used for the prototype. HTTP stands for Hypertext Transfer Protocol and is the foundation of data communication for the World Wide Web ecosystem (Fielding et al., 1999). That includes applications on Windows OS, Mac OS, iOS, Android, and Web applications running in browser.

To maximize accessibility I decided to create a Web application. The application is called Target and can be accessed through ufpor-bim.appspot.com.

GWT (Google Web Toolkit) technology was used to implement the framework. GWT is a development toolkit that facilitates development of complicated applications in a browser (Dewsbury, 2007). By using GWT developers can write every layer of a Web application in Java programming language. GWT compiles the Java code to Javascript which is the language that Web browsers understand. By using GWT some of the data models from the system that were created in previous section were reused.

8.3.2. INTRODUCING THE USER INTERFACE

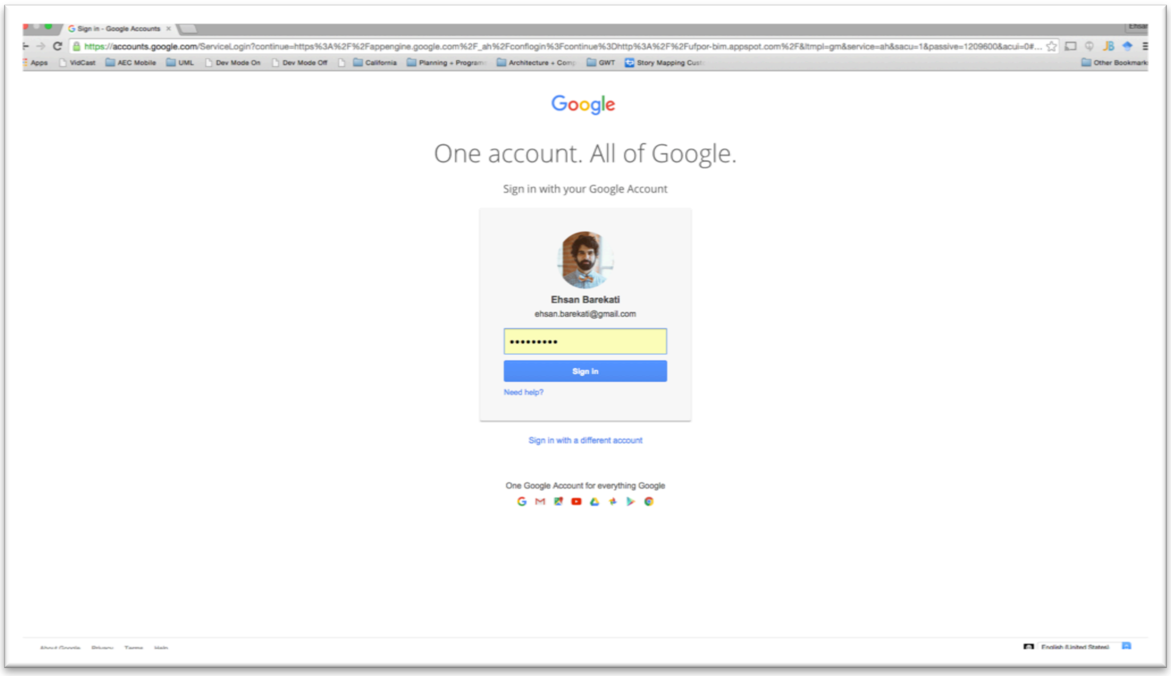
Figure 60 shows the first page of Target.


Howdy! Welcome to TARGET

My name is Ehsan Barekati and I am the developer of Target. Target allows you to define an architectural program of requirements in IFC format. You can see how the IFC file is developed as you interact with the application. You can also download the IFC file at any time and import it into other BIM applications. Target will use your Google account to save your projects and settings. Enjoy!

Figure 55, Target Homepage

Google authentication services were used to manage user information and keep track of created projects. Users can log into Target with any Google account as shown in the Figure 61.


Google

One account. All of Google.

Sign in with your Google Account

Ehsan Barekati

ehsan.barekati@gmail.com

Sign in
Need help?
Sign in with a different account
One Google Account for everything Google

Figure 56, Target User Authentication

After signing in with a Google account, the user is redirected to the main page where they can create a new project (IfcProject).

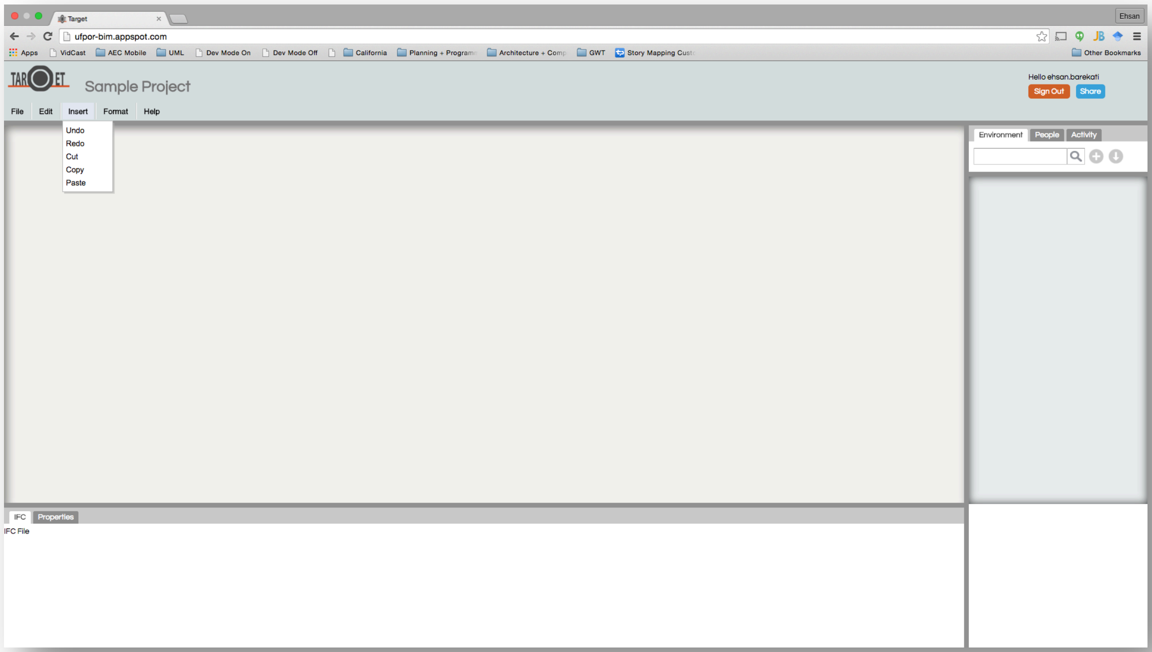


Figure 57, Target Main Page

The screen is divided into four main sections. The top area hosts the menus to control user and project related information. The right section hosts a panel that allows users to add new UFPOR Environments by creating instances of subclasses of IfcSpatialStructureElementType. The current implementation is limited to the instances of IfcSpaceType entity. The bottom section displays the IFC output based on user actions. The middle section allows the user to create a spatial hierarchy by dragging and dropping created IfcSpaceType instances from the right panel. To start a new project, the user should start “New” option from the file menu. Figure 63 shows the window that will open upon clicking on “New” from the file menu.

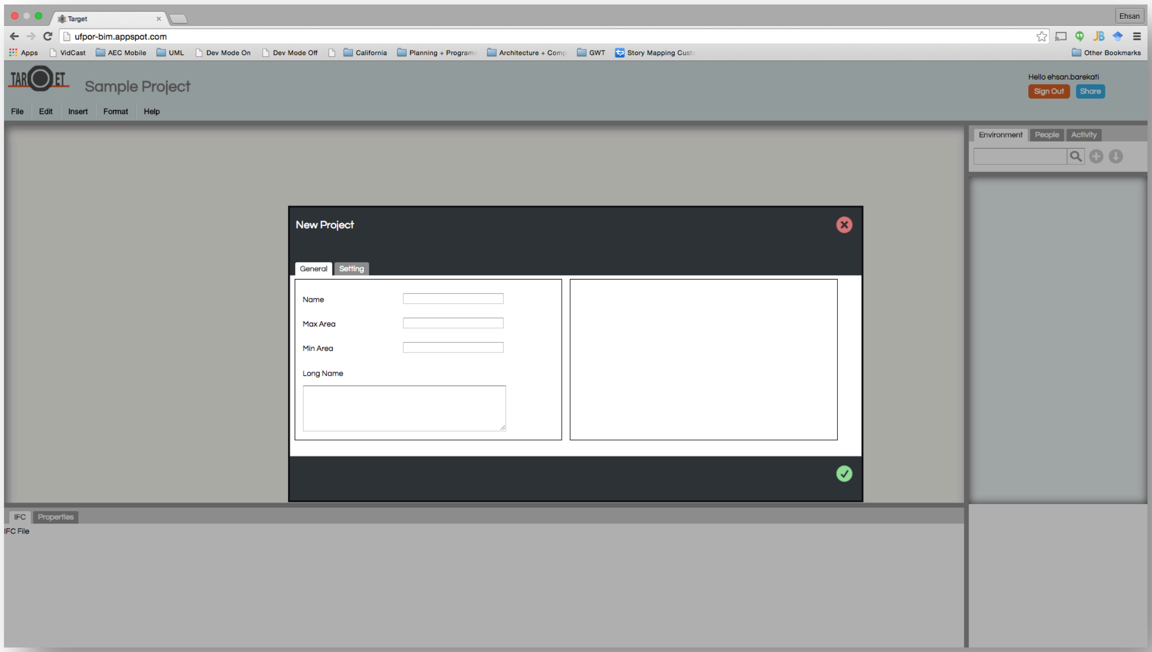
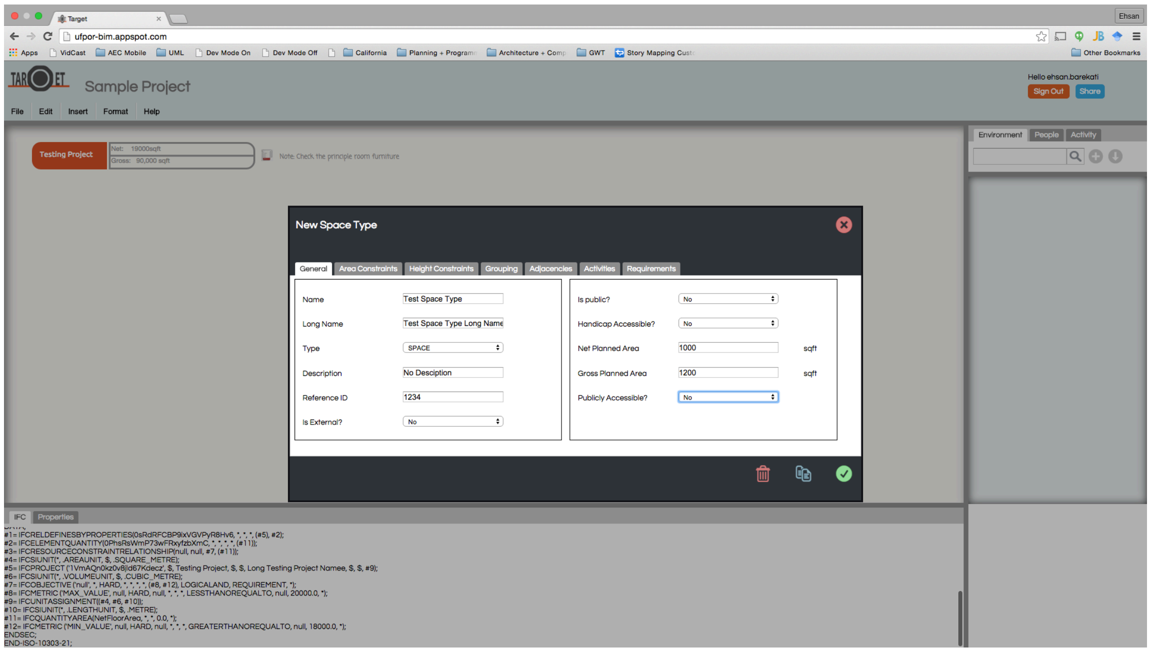


Figure 58, New Project Window in Target

The "New Project" window allows provide two tabs to define a new project. From the "General" tab users can define project name, maximum and minimum area, and a long name for the project. From the setting tab, the users can select their desired unit for length, area, and volume. After defining all the required fields, Target will send the information to Google App Engine over an HTTP request and receives the produced IFC physical file. The output then will be displayed in the bottom section. Also a root node will be added to the tree structure as shown in Figure 64.

After creating the project, the user can add new IfcSpaceType entities to the project. By clicking on the "plus" sign under the Environment panel in the right section.

A new window will pop up to define a new instance of IfcSpaceType as shown in Figure 64.

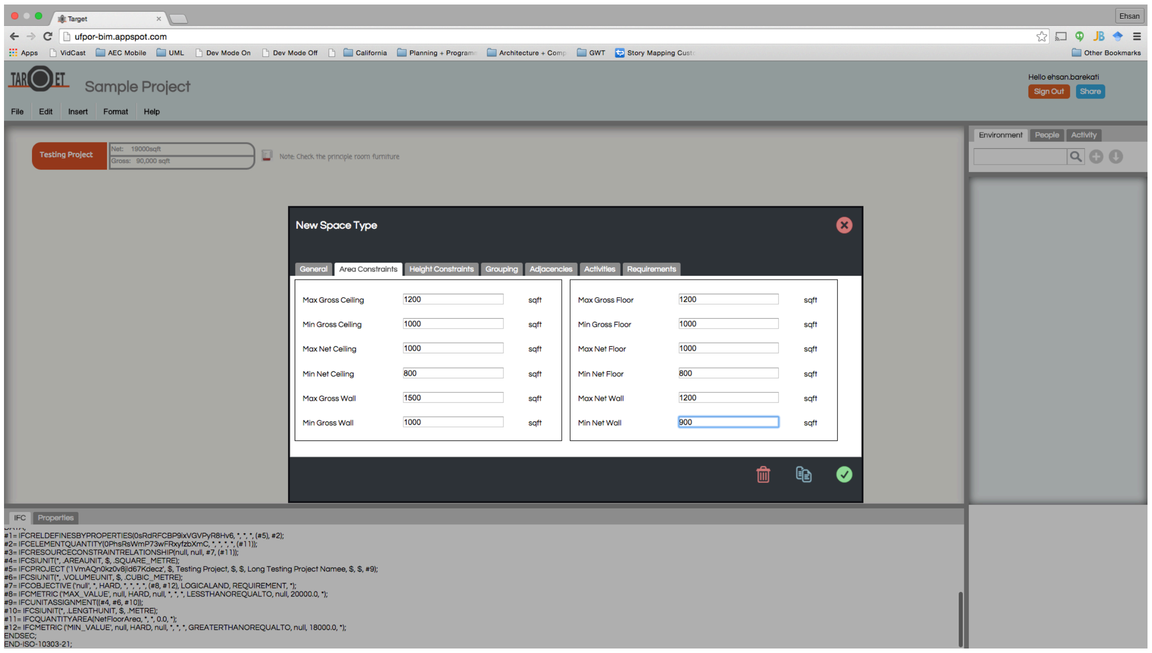


Name	Test Space Type	Is public?	No	
Long Name	Test Space Type Long Name	Handicap Accessible?	No	
Type	SPACE	Net Planned Area	1000	sqft
Description	No Desciption	Gross Planned Area	1200	sqft
Reference ID	1234	Publicly Accessible?	No	
Is External?	No			

```
#1= IFCRELDEFINESBYPROPERTIES(0sRdRFCBP9ixVGVPyR8Hv6, *, *, *, (#5), #2);
#2= IFCELEMENTQUANTITY(0PhsRsWmP73wFRxyfzbXmC, *, *, *, *, (#11));
#3= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #7, (#11));
#4= IFCSIUNIT(*, .AREAUNIT, $, .SQUARE_METRE);
#5= IFCPROJECT ('1VmAQn0kz0v8jld67Kdecz', $, Testing Project, $, $, Long Testing Project Namee, $, $, #9);
#6= IFCSIUNIT(*, .VOLUMEUNIT, $, .CUBIC_METRE);
#7= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#8, #12), LOGICALAND, REQUIREMENT, *);
#8= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO, null, 20000.0, *);
#9= IFCUNITASSIGNMENT((#4, #6, #10));
#10= IFCSIUNIT(*, .LENGTHUNIT, $, .METRE);
#11= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);
#12= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO, null, 18000.0, *);
ENDSEC;
END-ISO-10303-21;
```

Figure 59, Target New Space Type Window

From the "New Space Type" window the user can enter general information about the new space type through "General" tab. Besides General tab there are other tabs for additional requirements such as area constraints and height constraint. Figure 65 shows the tab related to Area Constraints.

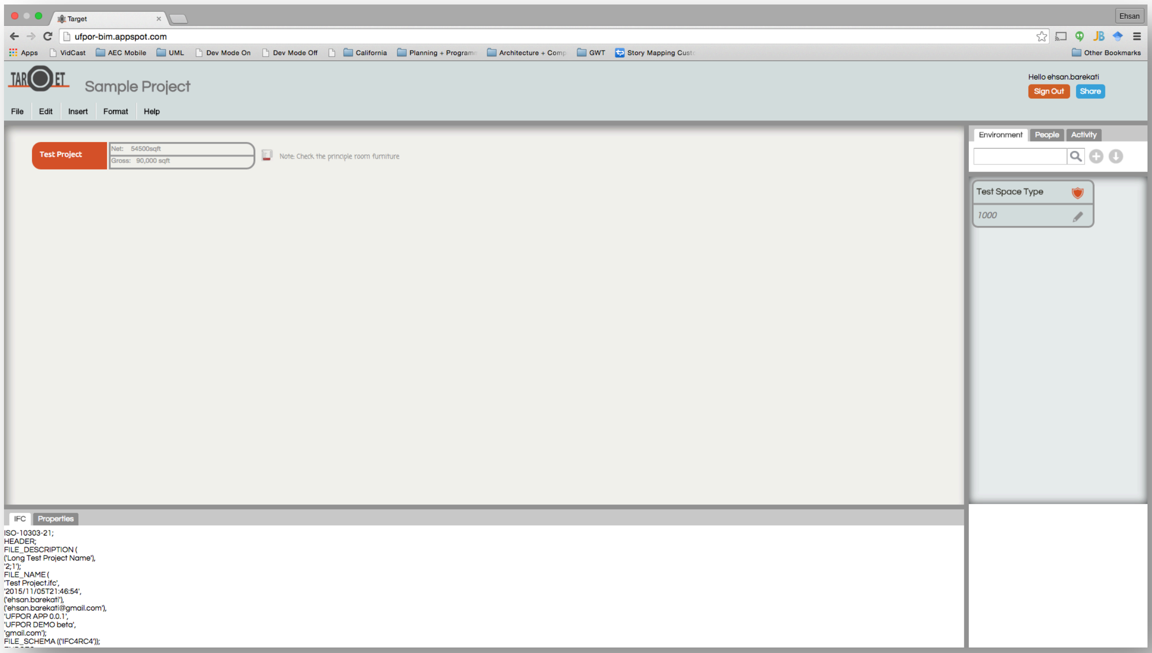


Max Gross Ceiling	1200	sqft	Max Gross Floor	1200	sqft
Min Gross Ceiling	1000	sqft	Min Gross Floor	1000	sqft
Max Net Ceiling	1000	sqft	Max Net Floor	1000	sqft
Min Net Ceiling	800	sqft	Min Net Floor	800	sqft
Max Gross Wall	1500	sqft	Max Net Wall	1200	sqft
Min Gross Wall	1000	sqft	Min Net Wall	900	sqft

```
#1= IFCRELDEFINESBYPROPERTIES(0sRdRFCBP9ixVGVPyR8Hv6, *, *, *, (#5), #2);
#2= IFCELEMENTQUANTITY(0PhsRsWmP73wFRxyfzbXmC, *, *, *, *, (#11));
#3= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #7, (#11));
#4= IFCSIUNIT(*, .AREAUNIT, $, .SQUARE_METRE);
#5= IFCPROJECT ('1VmAQn0kz0v8jld67Kdecz', $, Testing Project, $, $, Long Testing Project Namee, $, $, #9);
#6= IFCSIUNIT(*, .VOLUMEUNIT, $, .CUBIC_METRE);
#7= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#8, #12), LOGICALAND, REQUIREMENT, *);
#8= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO, null, 20000.0, *);
#9= IFCUNITASSIGNMENT((#4, #6, #10));
#10= IFCSIUNIT(*, .LENGTHUNIT, $, .METRE);
#11= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);
#12= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO, null, 18000.0, *);
ENDSEC;
END-ISO-10303-21;
```

Figure 60, Target New Space Type, Space Constraint

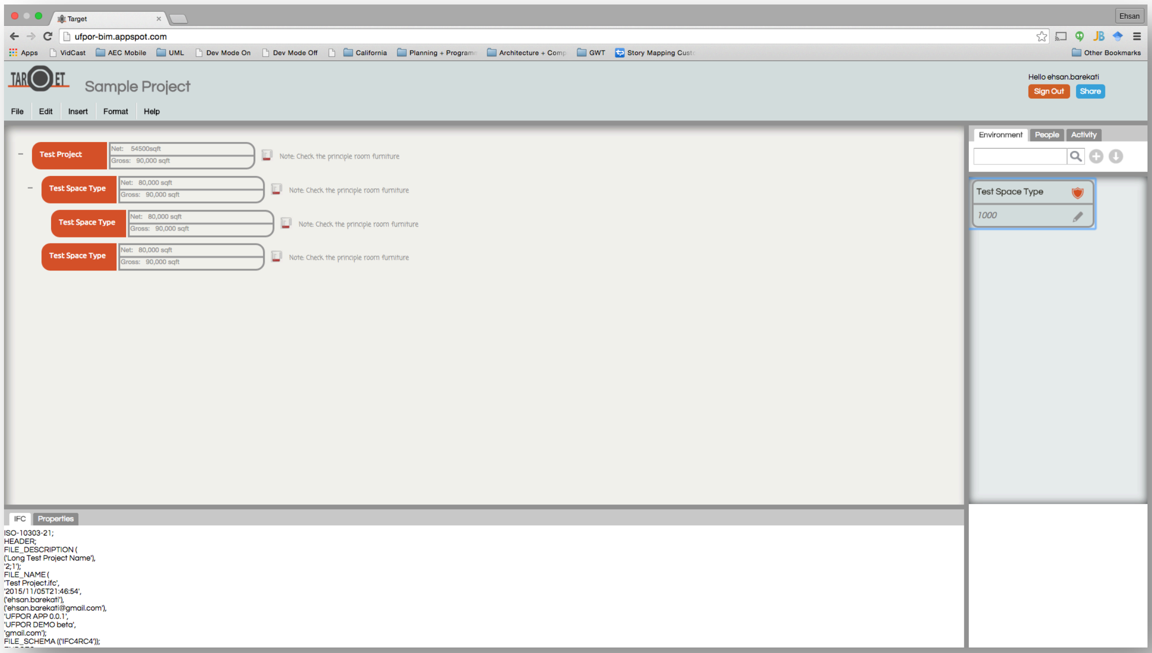
After entering all the required information and by clicking on the green check button, the information will be send the Google App Engine through an HTTP request and a new version of the IFC Physical file will be returned and displayed in the bottom section. Also, the newly created IfcSpaceType will be added to the Environment tab as shown in Figure 66. The user can add as many instances of IfcSpaceType as they desire to the project.



```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (
('Long Test Project Name'),
'2;1');
FILE_NAME (
'Test Project.ifc',
'2015/11/05T21:46:54',
('ehsan.barekati'),
('ehsan.barekati@gmail.com'),
'UFPOR APP 0.0.1',
'UFPOR DEMO beta',
'gmail.com');
FILE_SCHEMA (('IFC4RC4'));
```

Figure 61, New Space Type Added to the Document

The user can drag and drop the created Space Types from the Environment tab to the space hierarchy tree. Upon dropping the Space Type to the tree structure an HTTP request is sent to Google App Engine and a new instance of IfcSpace is added to the project's spatial structure, also a new version of IFC physical file is returned and added to the bottom section. Figure 67 shows a space tree hierarchy that includes three instance of IfcSpace.



Test Project — Net: 54500sqft — Gross: 90,000 sqft — Note: Check the principle room furniture
Test Space Type — Net: 80,000 sqft — Gross: 90,000 sqft — Note: Check the principle room furniture
Test Space Type — Net: 80,000 sqft — Gross: 90,000 sqft — Note: Check the principle room furniture
Test Space Type — Net: 80,000 sqft — Gross: 90,000 sqft — Note: Check the principle room furniture

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (
('Long Test Project Name'),
'2;1');
FILE_NAME (
'Test Project.ifc',
'2015/11/05T21:46:54',
('ehsan.barekati'),
('ehsan.barekati@gmail.com'),
'UFPOR APP 0.0.1',
'UFPOR DEMO beta',
'gmail.com');
FILE_SCHEMA (('IFC4RC4'));
```

Figure 62, New Spaces Added to the Project

8.4. CONCLUSION

The current state of IFC provides limited support for UFPOR. UFPOR-IFC, which is a subset of UFPOR was defined based on these limitations. UFPOR-IFC consists of a spatial structure and a collection of requirements that are directly imposed on the entities of the spatial structure and their properties.

Through running the unit tests, it was proven that the current state of IFC is capable of supporting UFPOR-IFC entities. It was also proven that the content of UFPOR-IFC can be transferred to IFC physical files. The generated IFC physical file can be processed by IFC compatible BIM computer applications, the applicability of the processed information is defined by the scope of each BIM computer application.

While UFPOR-IFC does not fully support the content of PORs it is valuable in creating a bridge between POR information and BIM models. UFPOR-IFC is the overlap of UFPOR and the current state of IFC. UFPOR-IFC can be used as a link between UFPOR and other BIM applications where the content of PORs are stored in a fully UFPOR compatible format and are linked to the rest of BIM model through UFPOR-IFC.

By developing a graphical user interface it was proven that UFPOR-IFC can be the base for user facing applications where the users can enter and manipulate POR documents that include a link with BIM models.

9. TESTING AGAINST ARCHITECTURAL PROGRAM SAMPLES

In previous section UFPOR-IFC was defined and tested by implementing a software prototype. In this section I examine the capabilities of UFPOR-IFC in capturing architectural programming requirements by analyzing two real-world PORs. The two PORs are selected from two well-known architectural design firms. The question I try to answer is “”In which capacity and to what extent can UFPOR-IFC support POR examples from the industry?”. I answer this question by analyzing architectural programming documents, classifying the information they provide into separate categories and finding respective data fields to capture the data in UFPOR-IFC.

Two different well-known architectural firms provide both of the architectural programs used in this section. Due to their request I cannot share the names and all the data revealing their names have been eliminated from the documents.

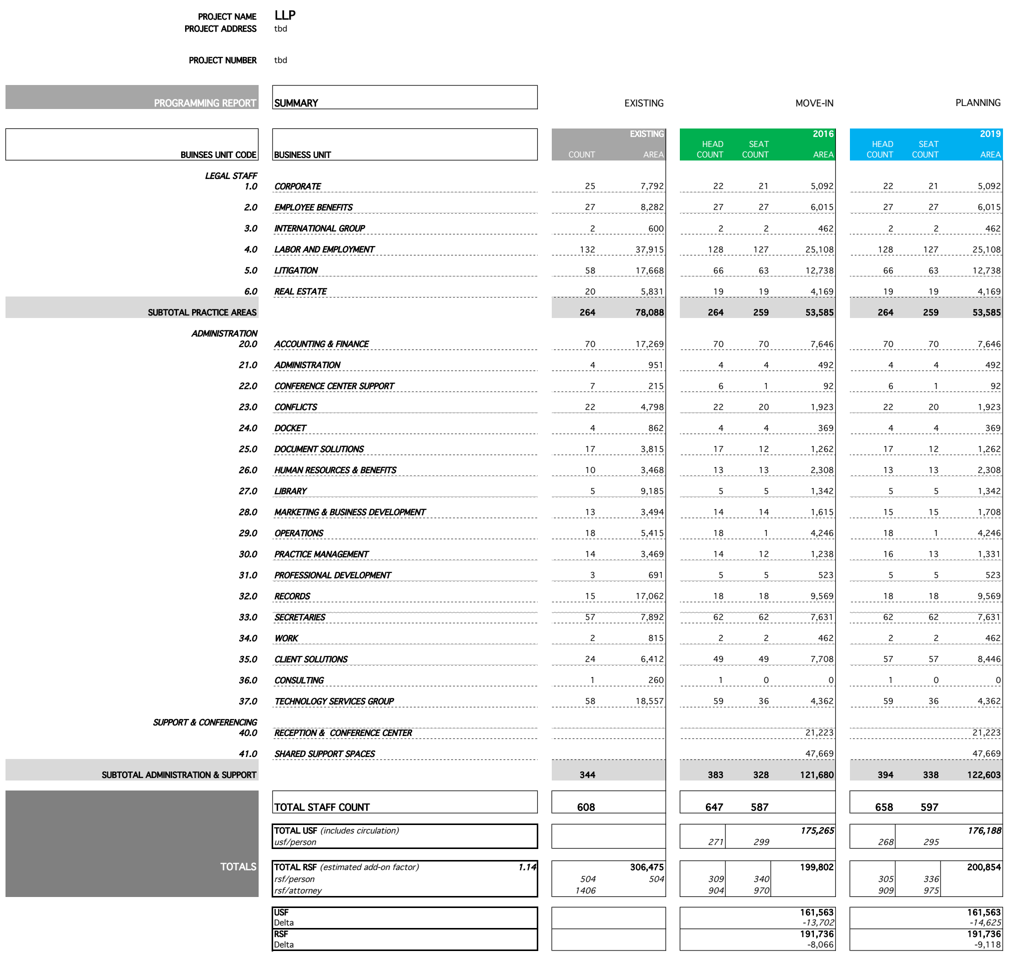
To showcase the capabilities of UFPOR-IFC that were not revealed by the two PORs, a third case study was added in form of a collection of requirements. The goal was to showcase the capabilities of UFPOR-IFC in modeling qualitative requirements as well as complicated quantitative requirements.

9.1. CASE STUDY 1. OFFICE BUILDING

The first case study is an expansion program for a 190,000 sqft office building. The office building is used for legal affairs and provides space for lawyers and administrative staff.

9.1.1. PROGRAM STRUCTURE

The first program is for a two-phase expansion for an office building. The program is organized into a collection of MS Excel worksheets. The first worksheet summarizes the project by Code all the departments and sections within each department (Figure 56). The project is divided into three main departments, “Legal Staff”, “Administration”, and “Support and Conferencing”. Each main department is further defines into business units.



PROJECT NAME	LLP
PROJECT ADDRESS	tbd
PROJECT NUMBER	tbd

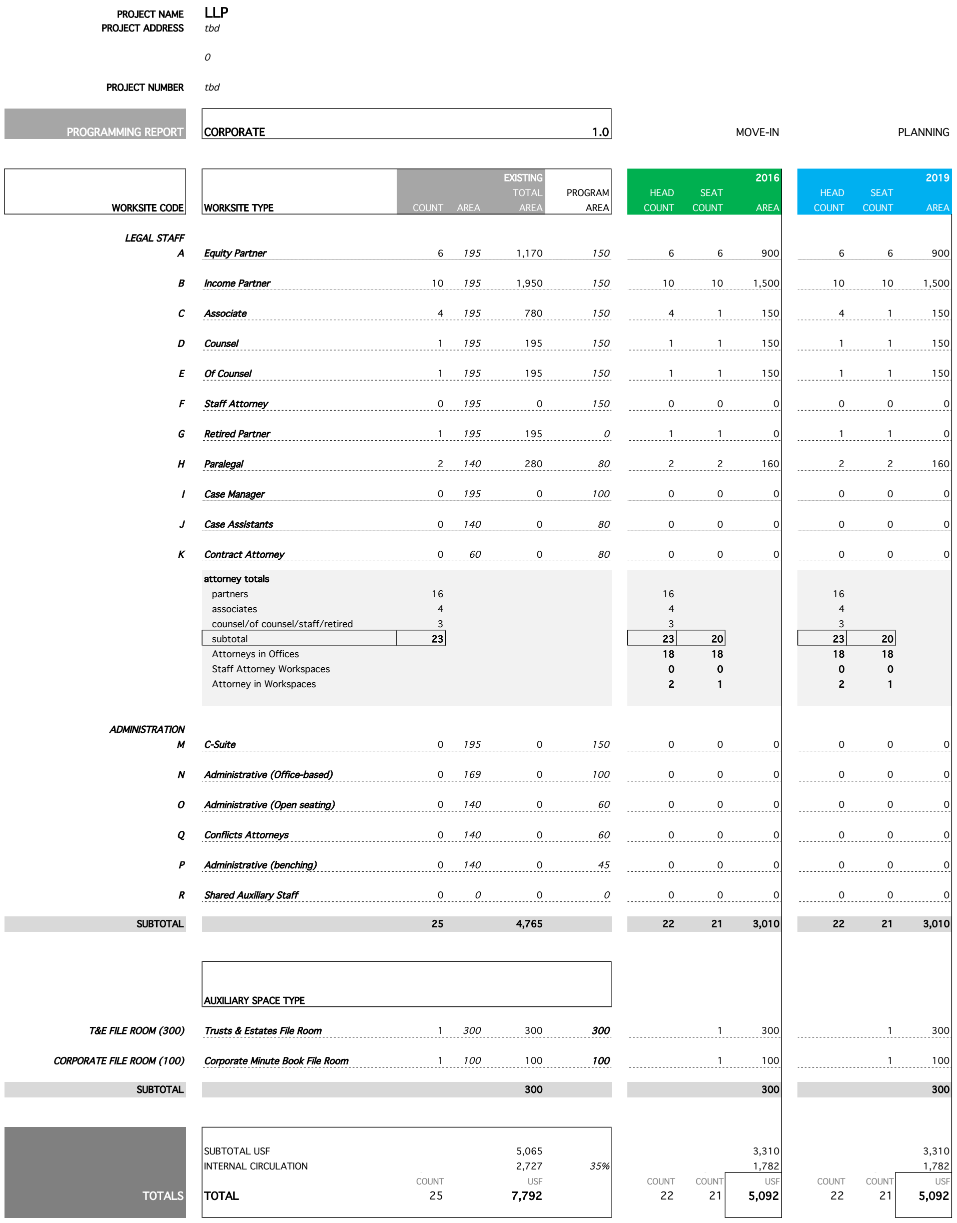
PROGRAMMING REPORT	SUMMARY	EXISTING		MOVE-IN			PLANNING		
			EXISTING			2016			2019
BUINSES UNIT CODE	BUSINESS UNIT	COUNT	AREA	HEAD COUNT	SEAT COUNT	AREA	HEAD COUNT	SEAT COUNT	AREA
LEGAL STAFF									
1.0	*CORPORATE*	25	7,792	22	21	5,092	22	21	5,092
2.0	*EMPLOYEE BENEFITS*	27	8,282	27	27	6,015	27	27	6,015
3.0	*INTERNATIONAL GROUP*	2	600	2	2	462	2	2	462
4.0	*LABOR AND EMPLOYMENT*	132	37,915	128	127	25,108	128	127	25,108
5.0	*LITIGATION*	58	17,668	66	63	12,738	66	63	12,738
6.0	*REAL ESTATE*	20	5,831	19	19	4,169	19	19	4,169
SUBTOTAL PRACTICE AREAS		**264**	**78,088**	**264**	**259**	**53,585**	**264**	**259**	**53,585**
ADMINISTRATION									
20.0	*ACCOUNTING & FINANCE*	70	17,269	70	70	7,646	70	70	7,646
21.0	*ADMINISTRATION*	4	951	4	4	492	4	4	492
22.0	*CONFERENCE CENTER SUPPORT*	7	215	6	1	92	6	1	92
23.0	*CONFLICTS*	22	4,798	22	20	1,923	22	20	1,923
24.0	*DOCKET*	4	862	4	4	369	4	4	369
25.0	*DOCUMENT SOLUTIONS*	17	3,815	17	12	1,262	17	12	1,262
26.0	*HUMAN RESOURCES & BENEFITS*	10	3,468	13	13	2,308	13	13	2,308
27.0	*LIBRARY*	5	9,185	5	5	1,342	5	5	1,342
28.0	*MARKETING & BUSINESS DEVELOPMENT*	13	3,494	14	14	1,615	15	15	1,708
29.0	*OPERATIONS*	18	5,415	18	1	4,246	18	1	4,246
30.0	*PRACTICE MANAGEMENT*	14	3,469	14	12	1,238	16	13	1,331
31.0	*PROFESSIONAL DEVELOPMENT*	3	691	5	5	523	5	5	523
32.0	*RECORDS*	15	17,062	18	18	9,569	18	18	9,569
33.0	*SECRETARIES*	57	7,892	62	62	7,631	62	62	7,631
34.0	*WORK*	2	815	2	2	462	2	2	462
35.0	*CLIENT SOLUTIONS*	24	6,412	49	49	7,708	57	57	8,446
36.0	*CONSULTING*	1	260	1	0	0	1	0	0
37.0	*TECHNOLOGY SERVICES GROUP*	58	18,557	59	36	4,362	59	36	4,362
SUPPORT & CONFERENCING									
40.0	*RECEPTION & CONFERENCE CENTER*					21,223			21,223
41.0	*SHARED SUPPORT SPACES*					47,669			47,669
SUBTOTAL ADMINISTRATION & SUPPORT		**344**		**383**	**328**	**121,680**	**394**	**338**	**122,603**
TOTALS	TOTAL STAFF COUNT	**608**		**647**	**587**		**658**	**597**	
	TOTAL USF *(includes circulation)*					***175,265***			***176,188***
	usf/person			*271*	*299*		*268*	*295*	
	TOTAL RSF *(estimated add-on factor)* ***1.14***		**306,475**			**199,802**			**200,854**
	rsf/person	*504*	*504*	*309*	*340*		*305*	*336*	
	rsf/attorney	*1406*		*904*	*970*		*909*	*975*	
	USF					**161,563**			**161,563**
	Delta					*-13,702*			*-14,625*
	RSF					**191,736**			**191,736**
	Delta					-8,066			-9,118

Figure 63, Office Building Program Summary

For example “Practice Areas” department includes six units: Corporate, Employee Benefits, International Group, Labor and Employment, Litigation, and Real Estate. Administration department includes eighteen units, and Support and Conferencing includes two.

The summary page provides summary information for each business unit. At the bottom at the worksheet total area for the project under each column is calculated. The calculation includes and additional 14% for the building elements. The information is divided into three categories, “Existing”, “Move-In”, and Planning. Each category provides three data points, Head Count, Seat Count, and Area.

Each business unit is further defined in a separate worksheet. Figure 57 shows the detail view for “Corporate” business unit.



PROJECT NAME	LLP
PROJECT ADDRESS	*tbd*
	0
PROJECT NUMBER	*tbd*

PROGRAMMING REPORT	CORPORATE				1.0	MOVE-IN			PLANNING		
		EXISTING				2016			2019		
				TOTAL	PROGRAM	HEAD	SEAT		HEAD	SEAT	
WORKSITE CODE	WORKSITE TYPE	COUNT	AREA	AREA	AREA	COUNT	COUNT	AREA	COUNT	COUNT	AREA
LEGAL STAFF											
A	*Equity Partner*	6	*195*	1,170	*150*	6	6	900	6	6	900
B	*Income Partner*	10	*195*	1,950	*150*	10	10	1,500	10	10	1,500
C	*Associate*	4	*195*	780	*150*	4	1	150	4	1	150
D	*Counsel*	1	*195*	195	*150*	1	1	150	1	1	150
E	*Of Counsel*	1	*195*	195	*150*	1	1	150	1	1	150
F	*Staff Attorney*	0	*195*	0	*150*	0	0	0	0	0	0
G	*Retired Partner*	1	*195*	195	*0*	1	1	0	1	1	0
H	*Paralegal*	2	*140*	280	*80*	2	2	160	2	2	160
I	*Case Manager*	0	*195*	0	*100*	0	0	0	0	0	0
J	*Case Assistants*	0	*140*	0	*80*	0	0	0	0	0	0
K	*Contract Attorney*	0	*60*	0	*80*	0	0	0	0	0	0
	attorney totals										
	partners	16				16			16		
	associates	4				4			4		
	counsel/of counsel/staff/retired	3				3			3		
	subtotal	**23**				**23**	**20**		**23**	**20**	
	Attorneys in Offices					**18**	**18**		**18**	**18**	
	Staff Attorney Workspaces					**0**	**0**		**0**	**0**	
	Attorney in Workspaces					**2**	**1**		**2**	**1**	
ADMINISTRATION											
M	*C-Suite*	0	*195*	0	*150*	0	0	0	0	0	0
N	*Administrative (Office-based)*	0	*169*	0	*100*	0	0	0	0	0	0
O	*Administrative (Open seating)*	0	*140*	0	*60*	0	0	0	0	0	0
Q	*Conflicts Attorneys*	0	*140*	0	*60*	0	0	0	0	0	0
P	*Administrative (benching)*	0	*140*	0	*45*	0	0	0	0	0	0
R	*Shared Auxiliary Staff*	0	*0*	0	*0*	0	0	0	0	0	0
SUBTOTAL		**25**		**4,765**		**22**	**21**	**3,010**	**22**	**21**	**3,010**
	AUXILIARY SPACE TYPE										
T&E FILE ROOM (300)	*Trusts & Estates File Room*	1	*300*	300	***300***		1	300		1	300
CORPORATE FILE ROOM (100)	*Corporate Minute Book File Room*	1	*100*	100	***100***		1	100		1	100
SUBTOTAL				**300**				**300**			**300**
TOTALS	SUBTOTAL USF			5,065				3,310			3,310
	INTERNAL CIRCULATION			2,727	*35%*			1,782			1,782
		COUNT		USF		COUNT	COUNT	USF	COUNT	COUNT	USF
	TOTAL	25		**7,792**		22	21	**5,092**	22	21	**5,092**

Figure 64, Office Building Program, Department

Each "Business Unit" is divided into multiple "Work Sites". Each "Work Site" is dedicated to a user group. For example Corporate "Business Unit" contains seventeen "Work Sites". For each "Work Site" the worksheet provides the same information that is provided in the “Summary” sheet for business units. The information is similarly divided into three columns under “Existing”, “Moving”, and “Planning”. At the bottom of each Business Unit worksheet is a summary row that adds area, head count and other information from every row. it also adds an additional area (35%) for “Internal Circulation”.

9.1.2. UFPOR COMPATIBILITY

The most important objects in this POR are “Business Units”, and “Work Sites”. They can be both modeled with a combination of IfcSpace and IfcSpatialZone instances. Both IFC entities are capable of carrying multiple quantity and property sets which is very helpful in facilitating gathering requirements and properties in three different categories (Existing, Move In, and Planning). The programming document does not include any information that requires use of UFPOR Activity class. The add-on factors for building elements and internal circulation can be added as custom properties to the project. Since the only information about the users is a plain headcount associated with Work Units, it can be added as a property directly to the instances of IfcSpace and IfcSpatialZone and avoid adding instances of IfcActor to the project.

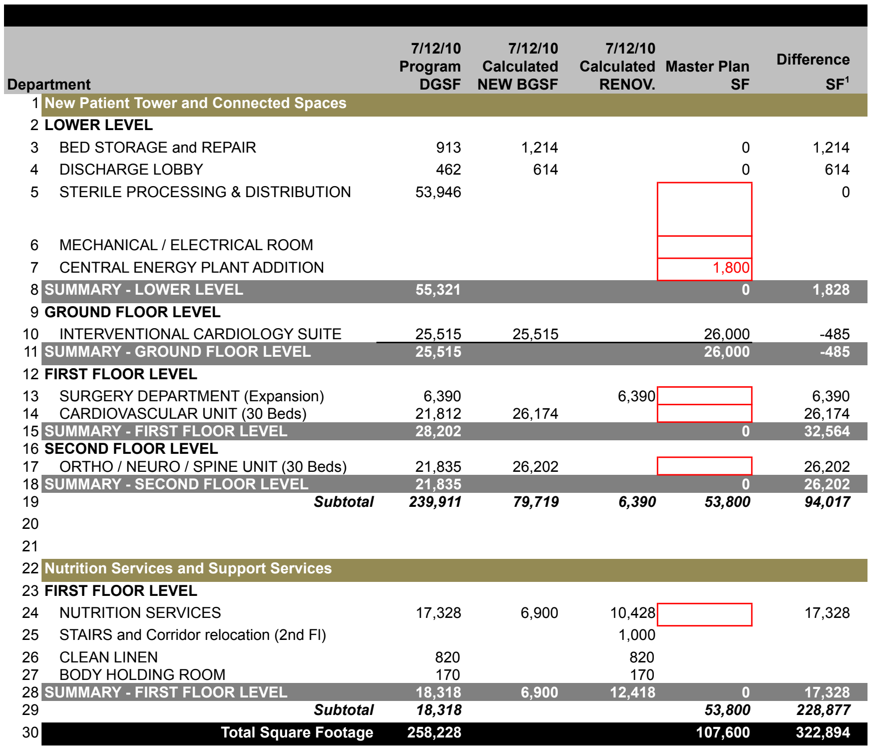
Given the absence of UFPOR Activity and limited use of UFPOR Need and UFPOR People, UFPOR-IFC is capable of capturing all the information presented in this POR.

9.2. CASE STUDY 2. HEALTHCARE FACILITY

The second case study is an expansion to an existing 258,000 sqft healthcare facility. The expansion includes a new patient tower and also a new building for nutrition services and support services.

9.2.1. PROGRAM STRUCTURE

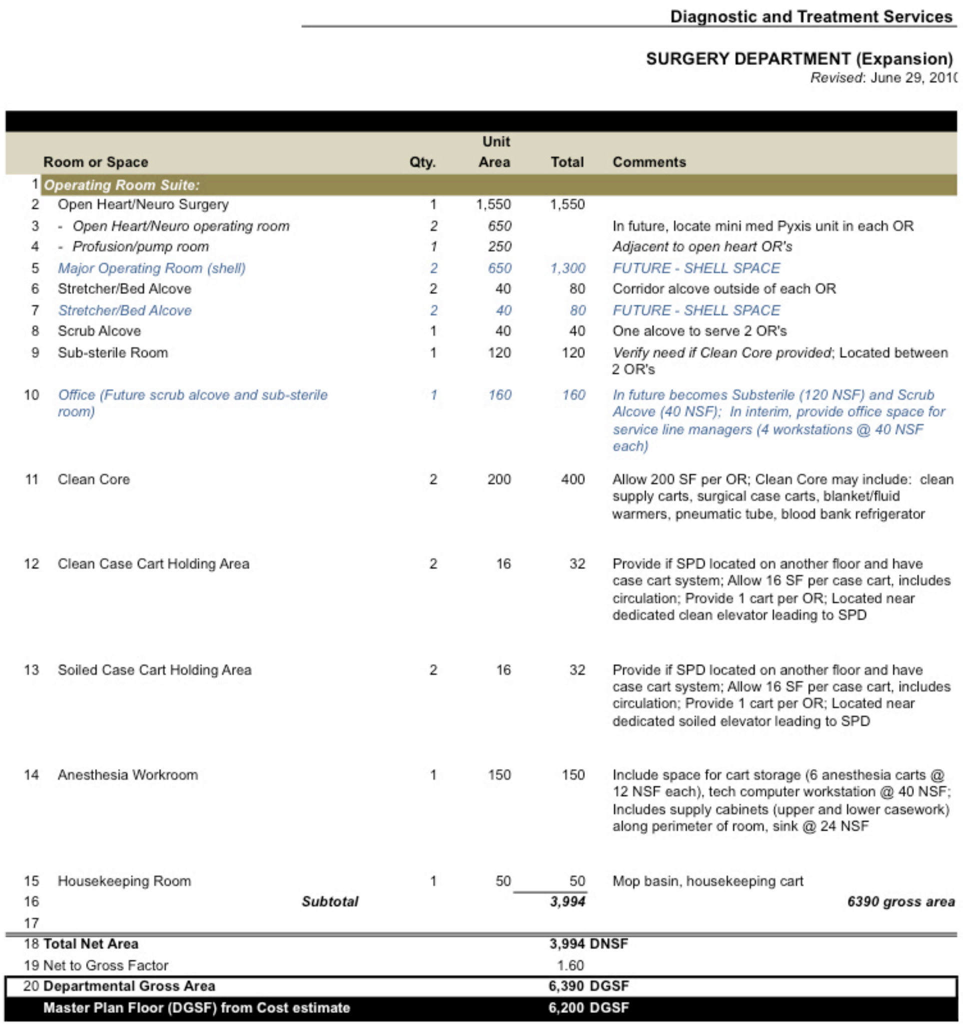
Similar to the example, this POR is composed of multiple Excel worksheets and starts with a summary page (Figure 70). The summary page breaks down the program into buildings and building stories. There are one or more departments under each building story. For each department the summary page provides five programming properties and an optional comment. The properties include departmental gross square footage (DGSF), building gross square footage, calculated renovation square footage, master plan square footage, and the difference square footage, which is obtained by subtracting Master plan SF from BGSF.



	Department	7/12/10 Program DGSF	7/12/10 Calculated NEW BGSF	7/12/10 Calculated RENOV.	Master Plan SF	Difference SF[1]
1	**New Patient Tower and Connected Spaces**					
2	**LOWER LEVEL**					
3	BED STORAGE and REPAIR	913	1,214		0	1,214
4	DISCHARGE LOBBY	462	614		0	614
5	STERILE PROCESSING & DISTRIBUTION	53,946				0
6	MECHANICAL / ELECTRICAL ROOM					
7	CENTRAL ENERGY PLANT ADDITION				1,800	
8	**SUMMARY - LOWER LEVEL**	**55,321**			**0**	**1,828**
9	**GROUND FLOOR LEVEL**					
10	INTERVENTIONAL CARDIOLOGY SUITE	25,515	25,515		26,000	-485
11	**SUMMARY - GROUND FLOOR LEVEL**	**25,515**			**26,000**	**-485**
12	**FIRST FLOOR LEVEL**					
13	SURGERY DEPARTMENT (Expansion)	6,390		6,390		6,390
14	CARDIOVASCULAR UNIT (30 Beds)	21,812	26,174			26,174
15	**SUMMARY - FIRST FLOOR LEVEL**	**28,202**			**0**	**32,564**
16	**SECOND FLOOR LEVEL**					
17	ORTHO / NEURO / SPINE UNIT (30 Beds)	21,835	26,202			26,202
18	**SUMMARY - SECOND FLOOR LEVEL**	**21,835**			**0**	**26,202**
19	***Subtotal***	***239,911***	***79,719***	***6,390***	***53,800***	***94,017***
20						
21						
22	**Nutrition Services and Support Services**					
23	**FIRST FLOOR LEVEL**					
24	NUTRITION SERVICES	17,328	6,900	10,428		17,328
25	STAIRS and Corridor relocation (2nd Fl)			1,000		
26	CLEAN LINEN	820		820		
27	BODY HOLDING ROOM	170		170		
28	**SUMMARY - FIRST FLOOR LEVEL**	**18,318**	**6,900**	**12,418**	**0**	**17,328**
29	***Subtotal***	***18,318***			***53,800***	***228,877***
30	**Total Square Footage**	**258,228**			**107,600**	**322,894**

Figure 65, Healthcare Facility Program, Summary Page

Each department is broken down into sections, sub-sections, and rooms in a separate worksheet. Figure 71 shows the detail on the surgery department.



Diagnostic and Treatment Services

SURGERY DEPARTMENT (Expansion)

Revised: June 29, 201(

	Room or Space	Qty.	Unit Area	Total	Comments
1	*Operating Room Suite:*				
2	Open Heart/Neuro Surgery	1	1,550	1,550	
3	*- Open Heart/Neuro operating room*	*2*	*650*		In future, locate mini med Pyxis unit in each OR
4	*- Profusion/pump room*	*1*	*250*		*Adjacent to open heart OR's*
5	*Major Operating Room (shell)*	*2*	*650*	*1,300*	*FUTURE - SHELL SPACE*
6	Stretcher/Bed Alcove	2	40	80	Corridor alcove outside of each OR
7	*Stretcher/Bed Alcove*	*2*	*40*	*80*	*FUTURE - SHELL SPACE*
8	Scrub Alcove	1	40	40	One alcove to serve 2 OR's
9	Sub-sterile Room	1	120	120	*Verify need if Clean Core provided*; Located between 2 OR's
10	*Office (Future scrub alcove and sub-sterile room)*	*1*	*160*	*160*	*In future becomes Substerile (120 NSF) and Scrub Alcove (40 NSF); In interim, provide office space for service line managers (4 workstations @ 40 NSF each)*
11	Clean Core	2	200	400	Allow 200 SF per OR; Clean Core may include: clean supply carts, surgical case carts, blanket/fluid warmers, pneumatic tube, blood bank refrigerator
12	Clean Case Cart Holding Area	2	16	32	Provide if SPD located on another floor and have case cart system; Allow 16 SF per case cart, includes circulation; Provide 1 cart per OR; Located near dedicated clean elevator leading to SPD
13	Soiled Case Cart Holding Area	2	16	32	Provide if SPD located on another floor and have case cart system; Allow 16 SF per case cart, includes circulation; Provide 1 cart per OR; Located near dedicated soiled elevator leading to SPD
14	Anesthesia Workroom	1	150	150	Include space for cart storage (6 anesthesia carts @ 12 NSF each), tech computer workstation @ 40 NSF; Includes supply cabinets (upper and lower casework) along perimeter of room, sink @ 24 NSF
15	Housekeeping Room	1	50	50	Mop basin, housekeeping cart
16	***Subtotal***			***3,994***	***6390 gross area***
17					
18	**Total Net Area**			**3,994 DNSF**	
19	Net to Gross Factor			1.60	
20	**Departmental Gross Area**			**6,390 DGSF**	
	Master Plan Floor (DGSF) from Cost estimate			**6,200 DGSF**	

Figure 66, Healthcare Facility Program, Surgery Department

For each departmental section, sub-section, and room, the program defines four assets: quantity, unit area, total area and comments. The areas from all the spaces in a department are added together to produce the total net area. Then net area is multiplied by “Net to Gross Factor” to produce the departmental gross area. The departmental gross area is included in the summary page.

9.2.2. UFPOR COMPATIBILITY

This POR is similar to the previous POR in terms of organization and content. The main difference is the fact that this program not only provides a list of departments along with their spaces, it also assign each department to a building floor in a specific building. The addition of buildings and building stories provides an opportunity to benefit from IfcBuilding and IfcBuildingStorey classes to classify the data. In addition to those two entities, IfcSpace, and IfcSpatialZone can be used to create the spatial structure.

Similar to the previous case study, the program does not contain any entities that would require the usage of UFPOR Activity. The program does not provide any information about the users, clients, or any other group that would require instances UFPOR People to model.

UFPOR-IFC is capable of capturing all the information presented in this POR.

9.3. CASE STUDY 3. COMMON PROGRAMING REQUIRMENTS

Two PORs were analyzed in the previous section and showed how IFC_UFPOR can support the imposed spatial structure and quantitative requirements. Neither of the PORs contained qualitative requirements. In this section sample on how some of the more complicated and qualitative requirements can be modeled through UFPOR-IFC are presented. Examples were selected to showcase the capabilities of UFPOR-IFC in modeling complex requirements.

9.3.1. ADJACENCY

Similar to other spatial requirements, adjacency requirements can be imposed through subclasses of IfcConstraint and associated to subclasses of IfcSpatialElement. Subclasses of IfcSpatialElement do not provide a list of their adjacencies; therefore the adjacency requirement cannot be imposed directly as IfcMetric instances. Adjacencies for an instance of IfcSpace are determined by analyzing BoundedBy inverse relationship (Figure 72).

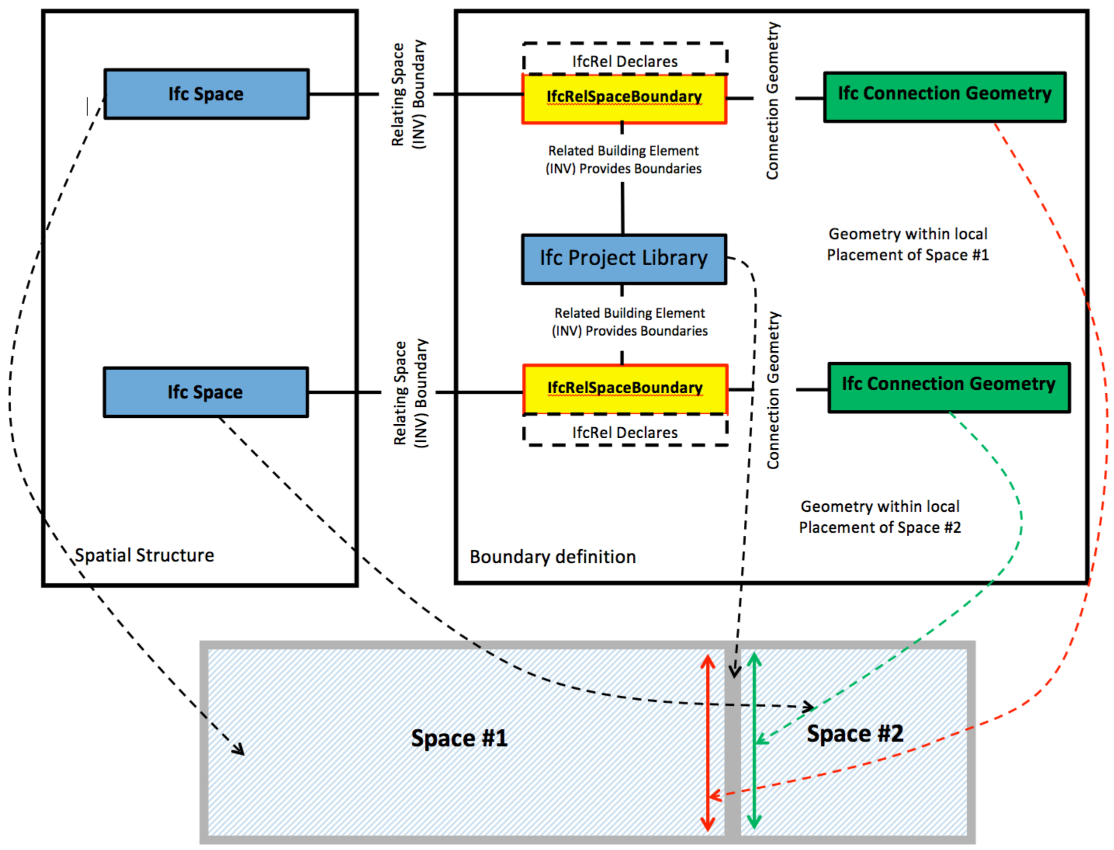

Ifc Space
Relating Space (INV) Boundary
IfcRel Declares
IfcRelSpaceBoundary
Connection Geometry
Ifc Connection Geometry
Related Building Element (INV) Provides Boundaries
Ifc Project Library
Geometry within local Placement of Space #1
Related Building Element (INV) Provides Boundaries
Ifc Space
Relating Space (INV) Boundary
IfcRelSpaceBoundary
IfcRel Declares
Connection Geometry
Ifc Connection Geometry
Geometry within local Placement of Space #2
Spatial Structure
Boundary definition
Space #1
Space #2

Figure 67, Spatial Adjacency in IFC

That makes adjacency requirements a special case of IfcObjective where they can be assessed by analyzing the IFC model. The limitation is that the adjacency requirements can be only imposed on instances of IfcSpace.

To clarify how this pattern can be implemented imagine a small office suite with a reception, conference room, a break room and a private office with a requirement on the break room to be adjacent to the private office. Here is how the adjacency requirement and the assessment process can be implemented.

The requirement can be imposed on either or both the break room and the private office through an instance of IfcObjective. Four properties should be used on IfcObjective Entity, Name, Description, and UserDefinedGrade. Name property content will be “adjacencyRequirement” and Description property should store a list of space GUIDs that the space is required to be adjacent to. Let’s imagine the adjacency requirement is for the office, and the break room’s GUID is 1234. The office will have an instance of IfcObjective attached to it with the name property set as “adjacencyRequirement” and the description property set as “[1234]”.

This adjacency requirement is assessed based on the shared building elements between the two spaces. Therefore the acceptance criteria should be clarified by providing a list of acceptable building elements. A list of acceptable building elements can be added as an array to the UserDefinedGrade. For example if the adjacency should provide physical accessibility, the list can be defined as: [IfcVirtualElement, IfcDoor, IfcRamp], but if visual connection is enough to make two spaces adjacent, IfcWindow can be added to the qualifying list. If the adjacency requirement is imposed to define wet and dry zones by keeping the wet spaces next to each other, IfcWall can be added to the list since neither visual nor physical accessibility is required.

To assess the requirement, all the instances of IfcRelSpaceBoundary attached to both the office and the break room should be examined to find a qualifying building element in common. A shared building element is qualifying only if it is mentioned in the list of building elements in UserDefinedGrade.

9.3.2. FURNITURE SIZE

To return to the office example, imagine a scenario where one wants to add a requirement for the private office to have a desk not smaller that a certain size. Such requirement is in fact two separate requirements, one on the private office and one on the desk itself.

Adding a size requirement to a desk is no different that adding a size requirement to a space. A custom property set with the required properties such as size can be defined. The desk will be represented by an instance of IfcFurniture where the PredefinedType property is set to Desk. The rest is similar to adding area requirements to the instances of IfcSpace.

The second requirement is on the private office itself where, similar to adjacency requirements, one is monitoring one of the IfcSpace inverse relationship properties. Instances of IfcFurniture are attached to instances of IfcSpatialElement through ContainsElements inverse property. An instance of IfcObjective entity can be used to impose the requirement with its Name property set to “furniture” and its Description property presenting an array of furniture GUIDs that are required in the private office.

To assess the requirement the software will search all the instances of IfcRelContainedInSpatialStructure in ContainsElements inverse property for the required GUIDs. The requirement is satisfied as soon as the GUID is found.

9.3.2.1. HANDICAP ACCESSIBILITY

Going back to the office example, imagine a scenario where one wants the whole office to be accessible for handicapped individuals. While this might be documented as one requirement on the project, handicap accessibility is a collection of several requirements from ADA and ABA standards (*ADA & ABA handbook,* 2004). The project needs to be accessible for the handicapped in terms of accessible routes, entrances, ramps and curbs, accessible means of egress, accessible toilet rooms and etc.

Let us focus on controlling the ramps for the private office space. The logic is based on having ramp accessibility wherever two spaces are connected through stairs. The process is similar to controlling adjacencies where common instances of IfcElement between the two spaces are of interest. This time instances of IfcStair are of interest. If there are any instances of IfcStair adjacent to the private office, at least one instance of IfcRamp should be adjacent to the office as well. Failure in finding an instance of IfcRamp adjacent to the office is equal to failure in satisfying the requirement. In reality the process might be more complicated where an indirect ramp access, provided through one of the adjacent spaces, is also acceptable.

To check the project for accessible toilet rooms, the requirements should be broken down into smaller requirements that can be assessed individually. For example area requirements should be imposed on bathroom area to insure a flawless wheelchair maneuvering. That can be accomplished the same way the area requirements were imposed on the instances of IfcSpace. A separate requirement should be imposed to guarantee the inclusion of grab-bars on the side and rear walls next to the toilet. That is also similar to imposing requirements to have a desk of a certain size in the office space.

9.3.2.2. QUALITATIVE CHARACTERISTICS OF A SPACE

Requesting a specific quality for a space is not out of ordinary in architectural projects. IFC provides all the required data models and processes to model qualitative requirements and their evaluation; therefore UFPOR-IFC can support the creation and assessment of such requirements. To return to the office example, imagine the client wants the break room to have certain qualities such as playfulness and coziness.

The requirement can be modeled through an instance of IfcObjective, the client can verbalize his or her requirement in detail through UserDefinedQualifier property on the instance of IfcObjective. In this example the value may be set to "The break room should provide a cozy and playful environment". ObjectiveQualifier property can be used to further classify why such requirement is imposed, in this case the user may select DESIGNINTENT as the value of ObjectiveQualifier property (other possible values were described in chapter 7). The instance of IfcObjective will be attached to the instance of IfcSpace that represents the break room through an instance of IfcRelAssociatesConstraint. The client will be modeled through an instance of IfcPerson and referenced through the CreatingActor on IfcConstraint instance.

As the project moves forward and the break room starts to form the assessment process begins. Several instances of IfcApproval can be attached to the instance of IfcConstraint representing the requirements through IfcResourceApprovalRelationship. Through IfcApproval the assessment sessions can be documented. The client can approve or reject the playfulness and coziness of the break room by looking at the drawings, 3d renderings or other sources of design representations. The client may go through several iterations until he or she is satisfied with the design, at that point an instance of IfcApproval representing the satisfactory status of the design will be attached to the requirement.

9.4. CONCLUSION

The first two case studies that were reviewed in this section both have less sophisticated structure and more limited data types when compared with UFPOR. While IFC had some limitations in implementing some of the data types from UFPOR (e.g. Activity) the current version of IFC was capable of capturing all the attributes of both of the samples.

In the last case study I could show how UFPOR-IFC is capable of documenting more complicated requirements such as adjacencies and handicapped accessibility.

The current version of IFC even though needs some improvements to capture all the data types that may be presented by a POR, is adequate in providing support for a large array of programming documents in architectural firms. It also provides the means to capture and control more sophisticated requirements such as adjacencies and handicapped accessibility.

10. CONCLUSION

The study set out to explore the capabilities of the current BIM standard data models to capture architectural programming information. As a prerequisite the study sought a parent format that can bring together the existing formats for a program of requirement (UFPOR). The study showed how UFPOR can represent three expert models. UFPOR was further tested by representing a POR excerpt from the industry. After that, the study explored the capabilities of IFC in presenting UFPOR. A separate data model was devised as the result of the analysis (UFPOR-IFC). UFPOR-IFC was implemented though a computer application. The integrity of the computer application was tested through several test scenarios. UFPOR-IFC was further analyzed by documenting two PORs from the industry and a collection of qualitative and complex requirements. The key questions that were answered by this study were:

- Can a universal model for architectural program of requirements (UFPOR) be devised that can be used as a super model to capture the information from different formats?
- If the answer is yes, can IFC as the standard data exchange format for BIM support the UFPOR in capturing the requirements and controlling the design against the requirements (UFPOR-IFC)?
- If the answer is yes, can UFPOR-IFC support a computer application that can capture the requirements and evaluate the design?

10.1. FINDINGS

Detailed findings for each chapter were summarized at the end of each chapter. This section synthesizes the findings for each chapter and brings them together to answer the research questions.

The study answered the first question by analyzing a collection of three POR formats. Despite different terminologies and different structures suggested by different scholars, a common data model blueprint (UFPOR) can support the content of all the PORs documented based on the reviewed POR formats.

An analysis of the IFC RC4 schema provided answers to the second question. IFC provides counterpart structures for some of the UFPOR classes. The UFPOR-IFC model is a subset of IFC that is capable of representing limited aspects of UFPOR Environment and UFPOR Need. UFPOR-IFC does not provide any support for UFPOR People and UFPOR Activity due to the limitations of the current state of IFC.

The study answered the third question by creating a prototype application to capture UFPOR-IFC compatible entities in IFC physical format. The prototype showed that UFPOR-IFC can be used to capture and exchange architectural programming information.

10.2. THEORETICAL IMPLICATION

Several books have been written on architectural programming. Besides them, there have been numerous conference papers and book chapters dedicated to the subject. Small and large architectural companies have also dedicated resources to the matter. One of the topics that have been discussed widely is the content and structure of a programming document. Different structures have been proposed and new terminologies have been introduced. The study showed that a single data model can support a wide range of those structures and terminologies.

One format may be superior to another format in providing a more human readable document by better naming and organization, it does not provide superiority by providing additional information.

As the number of BIM computer applications grows, the need for a solid exchange data format is becoming more crucial. IFC provides a framework where such exchange formats can be created. IFC provides the grammar and glossary needed for communication, yet it provides enough flexibility to create data exchange formats for the wide range of applications used in the AECO industry.

The current state of IFC does not provide all the required schemas to fully support architectural programming information. While the support for UFPOR Environment and UFPOR Need are limited but significant, IFC does not provide any support for UFPOR People and UFPOR Activity. IFC does not support documenting a project in a sequential basis or in different time spans; such capability is fundamental in UFPOR.

10.3. PRACTICAL IMPLICATION

Computer applications to document architectural programming requirements and further control the design versus the programming requirements have been a need in the industry. As this field grows, having a common standard to capture and transfer such data becomes more important. The study showed the limitations of IFC in providing such standard. The study also showed the current state of IFC is capable of providing limited support of architectural programming information. While the current state of IFC cannot provide a data exchange platform for complete architectural programming information, it can be used as a link between architectural programming information management applications and other BIM tools.

By adopting a standard such as UFPOR-IFC, architects can link architectural programing information to their desired collection of BIM computer applications. Given the current state of IFC, the architectural programming information cannot be fully captured in the IFC model and a separate data repository should maintain architectural programming information throughout the lifecycle of the project.

10.4. LIMITATIONS OF THE RESEARCH

For practical reasons the study was limited to analyzing a small collection of the literature on architectural programming. Thus from a theoretical standpoint the proposed UFPOR and UFPOR-IFC models are compatible with the formats derived from the literature analyzed in this study. While the methodology provided by the study can be used in adding other programming structures to the collection, minor modifications may be required as the collection grows.

IFC provides a large framework in which data exchange formats can be delimited and defined. The inherent flexibilities in IFC schemas and entities such as custom properties are fundamental in defining data exchange standards. Similarly UFPOR-IFC is not a finalized standard for the exchange of architectural programming information. It is rather a blueprint that shows how such standard can be created. Finalizing a standard based on UFPOR-IFC requires defining additional custom properties for UFPOR base classes.

The study was largely limited to logical argumentation in examining the hypothesis, supplemented by the empirical tests of implementing software and testing the software on some simple cases. While the nature of the study made logical argumentation the right methodology, if the industry adopts the results of the study, more experimental methods can be used to further test the hypothesis.

10.5. FUTURE WORKS

This study opens up the doors for future work for both practitioners and researchers in the AECO industry.

The study revealed shortcomings in the current state of IFC in supporting UFPOR, but shows that small modifications could lead to much greater utility. Consideration of these changes by standards organizations could lead to advances of significant benefit to the industry. The shortcomings should be communicated with buildingSMART. BuildingSMART can choose to make the required changed in the future versions of IFC so that the architectural programming information can be fully supported by IFC.

The main outcome of this study was UFPOR-IFC, a BIM compatible framework for architectural programming information. A consortium of key stakeholders in the AECO industry should standardize the proposed UFPOR-IFC. Such act would open the doors for architectural programming information to become accessible in the BIM computer applications.

After standardization on the proposed UFPOR-IFC, more experimental tests should be conducted to further refine the model based on the AECO industry needs and expectations.

The capabilities of UFPOR-IFC to create meaningful connections between architectural programing information stored in UFPOR and other BIM computer applications should be analyzed through case studies. Based on the findings, the enhancements on IFC to fully support UFPOR can be ordered by prioritizing the enhancements that create a stronger connection between UFPOR and other BIM computer applications.

The study showed that UFPOR-IFC can support both quantitative and qualitative requirements on the spatial elements. The study also showed examples of how different types of requirements can be documented and evaluated using UFPOR-IFC. Further studies on specific types of requirements such as adjacencies and handicap accessibility should analyze the capabilities and limitations of UFPOR-IFC in terms of computer software implementation and data modeling in more detail.

REFERENCES

About dRofus AS. (2015). Retrieved from http://drofus.no/en/product.html.

About Onuma. (2015). Retrieved from http://onuma-bim.com/

About Trelligence. (2015). Retrieved from http://www.trelligence.com/index.php

ADA & ABA handbook (2004). Los Angeles, CA: BNI Building News.

The American Institute of Architects (2014). *The architect's handbook of professional practice: Edition 15*. Hoboken, NJ: Wiley.

Arvidsson, F. & Flycht-Eriksson, A. (2008). *Ontologies I.* Retrieved from http://www.ida.liu.se/~janma56/SemWeb/Slides/ontologies1.pdf.

Barekati, E. (2012). HKS BIM+Programming. *Fiatech 2012 Technology Conference and Showcase,* Miami.

Bernstein, H. M., Jones, S. A., Russo, M., Laquidara-Carr, D., Taylor, W., Ramos, J., Fitch, E. (2012). The business value of BIM in north america. *SmartMarket Report,* 64.

Björk, B. C. & Laakso, M. (2010). CAD standardisation in the construction industry—A process view. *Automation in Construction, 19*(4), 398-406.

Bloch, J. (2001). *Effective java : Programming language guide*. Boston, MA: Addison-Wesley.

Bruegge, B. & Dutoit, A. H. (2004). *Object-oriented software engineering using UML, patterns and java*. Upper Saddle River, NJ: Prentice Hall.

buildingSMART. (2015a). IfcPropertyDefinition. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifckernel/lexical/ifcpropertysetdefinition.htm

buildingSMART. (2015b). IfcPropertySetDefinition. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifckernel/lexical/ifcpropertysetdefinition.htm

buildingSMART. (2015c). IfcPropertyTemplate. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc1/html/ifckernel/lexical/ifcpropertytemplate.htm

buildingSMART. (2015d). IfcSimplePropertyTemplate. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifckernel/lexical/ifcsimplepropertytemplate.htm

buildingSMART. (2016a). Composition. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/schema/templates/composition.htm buildingSMART. (2016b). IfcActorRole. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc1/html/ifcactorresource/lexical/ifcactorrole.htm

buildingSMART. (2016c). IfcApproval. Retrieved from http://www.buildingsmart-tech.org/ifc/review/IFC4Add1/rc1/html/schema/ifcapprovalresource/lexical/ifcapproval.htm

buildingSMART. (2016d). IfcBenchmarkEnum. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcconstraintresource/lexical/ifcbenchmarkenum.htm

buildingSMART. (2016e). IfcBuilding. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcproductextension/lexical/ifcbuilding.htm

buildingSMART. (2016f). IfcBuildingElement. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/lexical/ifcbuildingelement.htm

buildingSMART. (2016g). IfcConstraint. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4x1/html/schema/ifcconstraintresource/lexical/ifcconstraint.htm

buildingSMART. (2016h). IfcElementCompositionEnum. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcproductextension/lexical/ifcelementcompositionenum.htm

buildingSMART. (2016i). IfcLogicalOperatorEnum. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcconstraintresource/lexical/ifclogicaloperatorenum.htm

buildingSMART. (2016j). IfcMeasureValue. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcmeasureresource/lexical/ifcmeasurevalue.htm

buildingSMART. (2016k). IfcMeasureWithUnit. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcmeasureresource/lexical/ifcmeasurewithunit.htm

buildingSMART. (2016l). IfcMetric. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcconstraintresource/lexical/ifcmetric.htm

buildingSMART. (2016m). IfcMetricValueSelect. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcconstraintresource/lexical/ifcmetricvalueselect.htm buildingSMART. (2016n). IfcObjectDefinition. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifckernel/lexical/ifcobjectdefinition.htm

buildingSMART. (2016o). IfcObjective. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/schema/ifcconstraintresource/lexical/ifcobjective.htm

buildingSMART. (2016p). IfcObjectiveEnum. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4x1/html/schema/ifcconstraintresource/lexical/ifcobjectiveenum.htm

buildingSMART. (2016q). IfcOrganization. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcactorresource/lexical/ifcorganization.htm

buildingSMART. (2016r). IfcPerson. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcactorresource/lexical/ifcperson.htm

buildingSMART. (2016s). IfcPersonAndOrganization. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcactorresource/lexical/ifcpersonandorganization.htm

buildingSMART. (2016t). IfcPostalAddress. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/schema/ifcactorresource/lexical/ifcpostaladdress.htm buildingSMART. (2016u). IfcProject. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifckernel/lexical/ifcproject.htm

buildingSMART. (2016v). IfcReference. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/schema/ifcconstraintresource/lexical/ifcreference.htm

buildingSMART. (2016w). IfcRelAssigns. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifckernel/lexical/ifcrelassigns.htm#definition

buildingSMART. (2016x). IfcRelAssociatesApproval. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifccontrolextension/lexical/ifcrelassociatesapproval.htm

buildingSMART. (2016y). IfcRelContainedInSpatialStructure . Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcproductextension/lexical/ifcrelcontainedinspatialstructure.htm

buildingSMART. (2016z). IfcRelDefinesByType. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/beta1/html/ifckernel/lexical/ifcreldefinesbytype.htm

buildingSMART. (2016aa). IfcResourceObjectSelect. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/Add2/html/schema/ifcexternalreferenceresource/lexical/ifcresourceobjectselect.htm

buildingSMART. (2016ab). IfcRoleEnum. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/Add2/html/schema/ifcactorresource/lexical/ifcroleenum.htm

buildingSMART. (2016ac). IfcRoot. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifckernel/lexical/ifcroot.htm

buildingSMART. (2016ad). IfcSimpleValue. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcmeasureresource/lexical/ifcsimplevalue.htm

buildingSMART. (2016ae). IfcSite. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcproductextension/lexical/ifcsite.htm

buildingSMART. (2016af). IFCSpatialElement. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/lexical/ifcspatialelement.htm

buildingSMART. (2016ag). IfcSpatialElementType. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/lexical/ifcspatialelementtype.htm buildingSMART. (2016ah). IfcSpatialStructureElement. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcproductextension/lexical/ifcspatialstructureelement.htm

buildingSMART. (2016ai). IfcSpatialZoneTypeEnum. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/Add2/html/schema/ifcproductextension/lexical/ifcspatialzonetypeenum.htm

buildingSMART. (2016aj). IfcZone. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/lexical/ifczone.htm

buildingSMART. (2016ak). IfcZone. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/lexical/ifczone.htm

buildingSMART. (2016al). Pset_BuildingStoreyCommon. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/pset/pset_buildingstoreycommon.htm

buildingSMART. (2016am). Pset_SiteCommon . Retrieved from http://www.buildingsmart-tech.org/ifc/IFC2x4/beta2/html/psd/IfcProductExtension/Pset_SiteCommon.xml

buildingSMART. (2016an). Pset_SpaceCommon . Retrieved from
http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/psd/Pset_SpaceCommon.xml

buildingSMART. (2016ao). Pset_SpaceOccupancyRequirements. Retrieved from
http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/psd/Pset_SpaceOccupancyRequirements.xml

buildingSMART. (2016ap). Qto_BuildingBaseQuantities . Retrieved from
http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/qset/qto_buildingbasequantities.htm

buildingSMART. (2016aq). Qto_BuildingStoreyBaseQuantities. Retrieved from
http://www.buildingsmart-tech.org/ifc/IFC2x4/rc2/html/schema/ifcproductextension/qset/qto_buildingstoreybasequantities.htm

buildingSMART. (2016ar). Qto_SiteBaseQuantities. Retrieved from
http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcproductextension/qset/qto_sitebasequantities.htm

buildingSMART. (2016as). Qto_SpaceBaseQuantities. Retrieved from
http://www.buildingsmart-

tech.org/ifc/IFC4/final/html/schema/ifcproductextension/qset/qto_spacebasequantities.htm

buildingSMART. (2016at). Spatial Decomposition. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/templates/spatial-decomposition.htm

Cheon, Y. & Leavens, G. T. (2002). A simple and practical approach to unit testing: The JML and JUnit way. *ECOOP 2002—Object-oriented programming* (pp. 231-255) New York: Springer Publishing Company.

Cherry, E. (1999). *Programming for design: From theory to practice*. New York: John Wiley.

Clayton, M. J., Johnson, R. E., Song, Y., & Al-Qawasmi, J. (1998). *A study of information content of as-built drawings for USAA*. Texas A&M University, College Station, Texas: CRS Center.

Coates, P., Arayici, Y., Koskela, L., Kagioglou, M., Usher, C., & O'Reilly, K. (2010). *The limitations of BIM in the architectural process*. First International Conference on Sustainable Urbanization, Hong Kong, China.

Cuff, D. (1991). *Architecture : The story of practice*. Cambridge, Mass: MIT Press.

Dewsbury, R. (2007). *Google web toolkit applications*. Upper Saddle River, NJ: Pearson Education.

D'souza, D. F. & Wills, A. C. (1998). *Objects, components, and frameworks with UML: The catalysis approach*. Boston, Mass: Addison-Wesley.

Duell, R., Hathorn, T., & Hathorn, T. R. (2014). *Autodesk revit architecture 2015 essentials: Autodesk official press*. Hoboken, NJ: John Wiley & Sons.

Duerk, D. P. (1993). *Architectural programming: Information management for design*. New York: Van Nostrand Reinhold.

East, B. (2016). COBie MVD. Retrieved from http://docs.buildingsmartalliance.org/MVD_COBIE/

East, W. (2007). *Construction operations building information Exchange (COBIE)*. (No. ERDC/CERL TR-07-30). US Army Corps of Engineers.

East, W. (2012). *Building programming information exchange (BPie)*. National Institute of Building Sciences.

East, W. & Carrasquillo-Mangual, M. (2012). *The COBie guide: A commentary to the NBIMSUS COBie standard*. buildingSMART alliance.

East, W. & Nisbet, N. (2009). *Spatial compliance information exchange (SCie)*. buildingSMART alliance.

Eastman. (2008). *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers, and contractors*. Hoboken, NJ: John Wiley & Sons.

Eastman, C., Jeong, Y., Sacks, R., & Kaner, I. (2010). Exchange model and exchange object concepts for implementation of national BIM standards. *Journal of Computing in Civil Engineering, 24*(1), 25-34.

Eastman, C. M. (1999). *Building product models : Computer environments supporting design and construction*. Boca Raton, FL: CRC Press.

Eastman, C. M. & Siabiris, A. (1995). A generic building product model incorporating building type information. *Automation in Construction, 3*(4), 283-304.

Erhan, H. (2003). *Interactive computational support for modeling and generating building design requirements* (Doctoral dissertation, Carnegie Mellon University) Retrieved from ProQuest Dissertation and Theses database. (UMI No. 3350504).

Ewing, R., Handy, S., Brownson, R. C., Clemente, O., & Winston, E. (2006). Identifying and measuring urban design qualities related to walkability. *Journal of Physical Activity & Health, 3*, 223.

Ewing, R. & Handy, S. (2009). Measuring the unmeasurable: Urban design qualities related to walkability. *Journal of Urban Design, 14*(1), 65-84.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext Transfer Protocol--HTTP/1.1*. Internet Engineering Task Force. Retrieved from: http://ietf.org/rfc/rfc2068.txt

GAO. (2008). *The nation's long-term fiscal outlook april 2008 update*. (No. GAO-08-783R). United States Government Accountability Office.

Gielingh, W. (1988). General AEC referenoe model (GARM). *ISO TC 184/SC4/WG1 Document N329*.

Groat, L. N. & Wang, D. (2002). *Architectural research methods*. New York: J. Wiley.

Gruber, T. (2009). A translation approach to portable ontology specifications. *Knowledge Acquisition 5*, 199–220.

Hamill, P. (2004). *Unit test frameworks: Tools for high-quality software development* Sebastopol, CA: O'Reilly Media, Inc.

Hershberger, R. G. (1999). *Architectural programming and predesign manager*. New York: McGraw-Hill.

Hietanen, J. & Final, S. (2006). *IFC model view definition format*. Technical Report. International Alliance for Interoperability.

Hirschheim, R., Klein, H. K., & Lyytinen, K. (1995). *Information systems development and data modeling: Conceptual and philosophical foundations*. Cambridge, England: Cambridge University Press.

Howard, R. & Björk, B. C. (2008). Building information modelling-experts' views on standardisation and industry deployment. *Advanced Engineering Informatics, 22*(2), 271-280.

International Alliance for Interoperability- UK Chapter (2004). *Information requirements specification – [AR-5] early design*. London, England

ISO/IEC. (2007). *ISO 10303-28:2007 Industrial automation systems and integration -- product data representation and exchange -- part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas*. Geneva, Switzerland: ISO/IEC.

ISO/IEC. (2013). *ISO 16739:2013 Industry Foundation Classes (IFC) for data sharing in the construction and facility management industry*. Geneva, Switzerland: ISO/IEC.

Jernigan, F. E. (2008). *Big BIM, little bim: The practical approach to building information modeling : Integrated practice done the right way!* (2nd ed.). Salisbury, MD: 4Site Press.

Kiviniemi, A. (2005). *Requirements management interface to building project models* (Doctoral dissertation, Stanford University). Retrieved from ProQuest Dissertation and Theses database. (UMI No. 3162341).

Kroenke, D. & Auer, D. J. (2007). *Database concepts* (3rd ed.). Upper Saddle River, NJ: Pearson Prentice Hall.

Lalmas, M. (2009). *XML retrieval*. San Rafael, CA: Morgan & Claypool Publishers.

Lapierre, A. & Cote, P. (2007). Using open web services for urban data management: A testbed resulting from an OGC initiative for offering standard CAD/GIS/BIM services. Paper presented at the *Urban and Regional Data Management. Annual Symposium of the Urban Data Management Society,* 381-393. London, England: Taylor & Francis.

Bazjanvac, V. (2002). Early lessons from deployment of IFC compatible software. *Fourth Euro. Conf. Product Process Modeling. Portoroz, Slovacia.*

Lawson, B. (1997). *How designers think: The design process demystified*. London, England: Architectural Press.

Lee, K., Chin, S., & Kim, J. (2003). A core system for design information management using industry foundation classes. *Computer-Aided Civil and Infrastructure Engineering, 18*(4), 286-298.

Lenzerini, M. (2002). Data integration: A theoretical perspective. Paper presented at the *Symposium on Principles of Database Systems: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 3*(05) 233-246.

LePatner, B. B. (2008). *Broken buildings, busted budgets: How to fix america's trillion-dollar construction industry*. University of Chicago Press.

Liebich, T. (2009). *IFC2 Edition 3 Model implementation guide*. buildingSMART International.

Liebich, T. (2010). Unveiling IFC2x4-the next generation of OPENBIM. Paper presented at the *Proceedings of the 2010 CIB W78 Conference,* 8.

Liebich, T. & Wix, J. (1999). Highlights of the development process of industry foundation classes. Paper presented at the *Proceedings of the 1999 CIB W78 Conference,* 18.

MacKenzie, S. H. & Rendek, A. (2015). *ArchiCAD 19-the definitive guide*. Birmingham, UK: Packt Publishing.

Malleson, A., Mordue, S., & Hamil, S. (2012). *The IFC/COBie report*. Open BIM Network.

National Institute of Building Sciences. (2007). *United states national building information modeling standard : Transforming the building supply chain through* *open and interoperable information exchanges*. Washington, DC: National Institute of Building Sciences.

Open Design Alliance. (2013). *Open design specification for .dwg files version 5.3*. Retrieved from: https://www.opendesign.com

Oracle. (2009). *Java data objects 2.3*. Retrieve from: https://db.apache.org/jdo/

Ouksel, A. M. & Sheth, A. (1999). Semantic interoperability in global information systems. *ACM Sigmod Record, 28*(1), 5-12.

Page-Jones, M. (2000). *Fundamentals of object-oriented design in UML.* New York, NY: Addison-Wesley.

Palmer, M. A. (1981). *The architect's guide to facility programming*. New York, NY: Architectural Record Books.

Peak, R. S., Lubell, J., Srinivasan, V., & Waterbury, S. C. (2004). STEP, XML, and UML: Complementary technologies. *Journal of Computing and Information Science in Engineering, 4*(4), 379-390.

Peña, W. & Parshall, S. (2001). *Problem seeking: An architectural programming primer* (4th ed.). New York: Wiley.

Pontieri, L., Ursino, D., & Zumpano, E. (2003). An approach for the extensional integration of data sources with heterogeneous representation formats. *Data & Knowledge Engineering, 45*(3), 291-331. doi:10.1016/S0169-023X(02)00192-1

Pratt, M. J. (2001). Introduction to ISO 10303—the STEP standard for product data exchange. *Journal of Computing and Information Science in Engineering, 1*(1), 102-103.

Preiser, W. F. E. (1975). Programming for habitability: Symposium proceedings, september 22-24, 1974, allerton house, university of illinois. Paper presented at the *Monograph Series - University of Illinois at Urbana-Champaign, Dept. of Architecture,* iv, 119.

Preiser, W. F. E. (1993). *Professional practice in facility programming*. New York: Van Nostrand Reinhold.

QuickBase. (2016). About QuickBase. Retrieved from http://quickbase.intuit.com/about-us

Richens, P. (1978). *The OXSYS system for the design of buildings*. Paper presented at the CAD78: 3rd Inter. Conf. on Computers in Engineering and Building Design, Sussex, UK.

Roddis, W., Matamoros, A., & Graham, P. (2006). Interoperability in building construction using exchange standards. *Intelligent Computing in Engineering and Architecture*, 576-596.

Russell, C. (2010). *JavaTM data objects 3.0*. (No. JSR 243). Sun Microsystems.

Sanoff, H. (1977). *Methods of architectural programming*. Stroudsburg, PA: Dowden, Hutchinson & Ross.

Schenck, D. A. & Wilson, P. R. (1994). *Information modeling the EXPRESS way* Oxford University Press, USA.

Smith, D. (2009). In Tardif M. (Ed.), *Building information modeling: A strategic implementation guide for architects, engineers, constructors, and real estate asset managers*. Hoboken, NJ: Wiley.

Smith, B. & Wellington, J. (1986). *Initial graphics exchange specification (IGES)*. (No. PB86-199759). National Institute of Standards and Technology.

Stroustrup, B. (2014). *Programming: Principles and practice using C++ 2nd ed*. Upper Saddle River, NJ: Addison-Wesley

Succar, B. (2009). Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction, 18*(3), 357-375.

Vanlande, R., Nicolle, C., & Cruz, C. (2008). IFC and building lifecycle management. *Automation in Construction, 18*(1), 70-78.

Vlissides, J., Helm, R., Johnson, R., & Gamma, E. (1995). *Design patterns: Elements of reusable object-oriented software*. Boston, MA: Addison-Wesley.

Weise, M., Katranuschkov, P., & Scherer, R. J. (2003). Generalised model subset definition schema. *CIB REPORT, 284*, 440.

Zahariev, A. (2009). *Google app engine*. Helsinki, Finland: Helsinki University of Technology.

APPENDIX 1

DETAILED SCHEMATIC DESIGN PROGRAM for an ART SCHOOL

(sample of some typical instruction spaces)

SPACE NAME: 3D Foundations - Discussion Room

DESCRIPTION

This classroom should be adjacent to the 3D Foundations: Making zone with butcher block tables for "light fabrication" activities. Students need to go back and forth between the working tables, the main fabrication facility and this classroom for foundations classes. Within the classroom the class will have discussions and lecture activities. On occasion, it would be beneficial to move the classroom tables to the side of the room to have "closed" in-class critiques. Darkness is required when projecting. Storage can be just outside the classroom in the working/critique zone.

QUALITIES: Clean, Closed

QUANTITY: 3

NUMBER OF OCCUPANTS: 18 + 1

SIMILAR TO:

CATEGORY: Classroom

AMENITIES:

FURNITURE
• (9) 24" X 60" OR (6) 24" X 72" FLIP-TOP TABLES ON CASTERS W/BOARD COVER.
• (19) CHAIRS
• (1) INSTRUCTOR'S TABLE FULLY INTEGRATED WITH STUDENT DISCUSSION TABLES

EQUIPMENT
• (1) E001: CEILING MTD. PROJECTOR*
• (1) E002: PROJECTION SCREEN*
• *(1) ALTERNATE: FLAT SCREEN TV
• (2) E004: SPEAKER
• (1) E110: INSTRUCTOR'S COMPUTER

STORAGE
• (10) SELF-ACCESS TOOL & EQUIPMENT STORAGE
• (10) COAT HOOKS
• STORAGE FOR STUDENTS BELONGINGS TO KEEP WORK SURFACES CLEAR
• (1) PAD-LOCKED SHORT TERM INSTRUCTOR'S LOCKER
• (6) COIN-TYPE LOCKERS FOR SMALL STUDENTS' SMALL VALUABLES

LIGHTING & CEILING
• DAY-LIGHT BALANCED WALL-WASH LIGHTING (5000-5500 K) AT TACKABLE SURFACES
• DIRECT / INDIRECT GENERAL LIGHTING
• CEILING OPEN-TO DECK IS PREFERRED

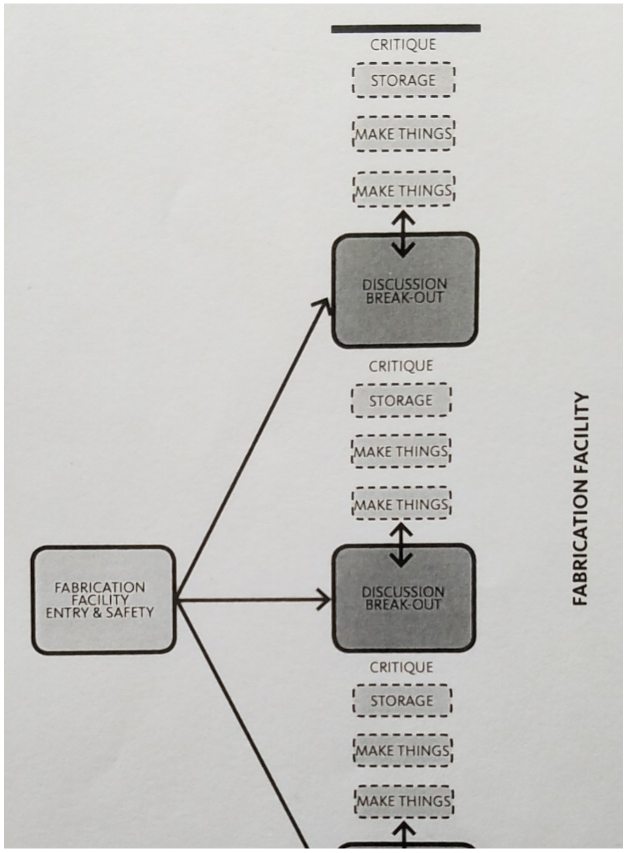
POWER & DATA
• WIRELESS CONNECTIVITY THROUGHOUT
• POWER ACCESSIBLE FROM CENTER OF ROOM
• HARD-WIRE POWER & DATA CONNECTIONS AT INSTRUCTOR'S TABLE

FINISHES & SURFACES
TACKABLE WALL SURFACE
• WRITABLE SURFACE AT TEACHING AREAS (MARKER BOARDS OR "IDEA PAINT"
• BLOCKING AT SELECT WALLS FOR MOUNTING PROJECTS AND EQUIPMENT
• HARD-SURFACE FLOORING

NOTES
• LOCATE NEAR FOUNDATIONS EQUIPMENT

LOCATION: Within Fabrication facility

DIAGRAM: *Adjacency*



***SPACE NAME**: 3D Foundations – Making Space*

DESCRIPTION

This working area should be directly adjacent to the 3D Foundations Classroom. During a class, students and the teacher will use this area for hands-on light fabrication such as gluing, cutting, models etc. The "discussion" & "making" zones work as one space for a class but are distinct enough so that the discussions can be focused and quiet while the fabricating area can get messy and students can collaborate. Within the making area there should be storage for 3D objects such that the objects are visible to other classes and students. When a foundations class is not in the making zone, people working in the fabrication facility could use the making zone as lay-out or light fabrication space.

QUALITIES: Messy, Semi-open

QUANTITY: 2·3

NUMBER OF OCCUPANTS: 18 + 1

SIMILAR TO:

CATEGORY: Fabrication

AMENITIES:

FURNITURE
• (18) 20" X 32" STATIONS (BUTCHER BLOCK TYPE), ADJUSTABLE HEIGHTS
• (19) STOOLS
• (1) 20" X 32" INTEGRATEABLE INSTRUCTORS WORK/DEMONSTRATION SURFACE
• (1) SINK

STORAGE
• (7) METAL SHELVES 36"W x 24"D x 72"H
• (1) PAD-LOCKED SHORT·TERM INSTRUCTORS LOCKER
• (10) SELF·ACCESS TOOL & EQUIPMENT STORAGE

LIGHTING & CEILING
• DAY-LIGHT BALANCED WALL-WASH LIGHTING (5000-5SOO K) TACKABLE SURFACES
• DIRECT /INDIRECT GENERAL LIGHTING
• CEILING OPEN-TO DECK IS PREFERRED

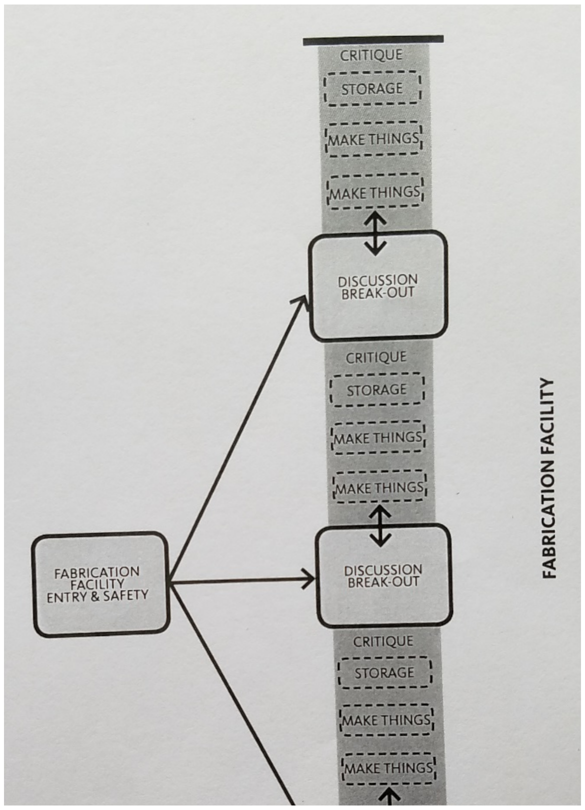
POWER & DATA
• WIRELESS CONNECTIVITY THROUGHOUT
• POWER ACCESSIBLE FROM CENTER OF ROOM

FINISHES & SURFACES
• TACKABLE WALL SURFACE
• BLOCKING AT SELECT WALLS FOR MOUNTING PROJECTS AND EQUIPMENT
• HARD-SURFACE FLOORING

NOTES
• EXHAUST/VENTILATION REQUIREMENTS
• LOCATE NEAR FOUNDATIONS EQUIPMENT

LOCATION: Within Fabrication facility

DIAGRAM: *Adjacency*


CRITIQUE
STORAGE
MAKE THINGS
MAKE THINGS
DISCUSSION BREAK-OUT
CRITIQUE
STORAGE
MAKE THINGS
MAKE THINGS
DISCUSSION BREAK-OUT
FABRICATION FACILITY ENTRY & SAFETY
CRITIQUE
STORAGE
MAKE THINGS
MAKE THINGS
FABRICATION FACILITY

SPACE NAME: Life Room

DESCRIPTION

This classroom is intended for painting, drawing and anatomical modeling with clay. Students work from a live model therefore windows need black-out shades and the entry to the room should be discreet and secure from the corridor and painting storage. The model would benefit from a separate changing area with storage for sheets and belongings. An area to store furniture while not in use is needed. Storage is also required for paintings and clay projects. The room should have an art sink with storage underneath for supplies and clay. This room is a priority for natural light.

QUALITIES: Dirty, Closed

QUANTITY: 1

NUMBER OF OCCUPANTS: 18 + 1

SIMILAR TO:

CATEGORY: Classroom

AMENITIES:

FURNITURE
• (18) EASELS
• (18) DRAWING HORSES
• (18) 20" X 30" MOVEABLE PALETTE TABLES

• (19) STOOLS
• (1) 8'x4' MODEL PLATFORM ON LOCKING CASTERS
• (3) 4' W X 6' H WORK APPARATUSES
• (1) INSTRUCTOR'S TABLE

EQUIPMENT
• (1) E001: CEILING MTD. PROJECTOR*
• (1) E002: PROJECTION SCREEN*
• *(1) ALTERNATE FLAT SCREEN TV
• (2) E004: SPEAKER
• (3) E110: COMPUTER (2 FOR STUDENTS, 1 AT INSTRUCTOR'S TABLE)
• (1) E006: SCANNER
• (1) SINK

STORAGE
• METAL PAINTING STORAGE SHELVES
• AREA FOR EASELS, DRAWING HORSES AND TABLES WHILE NOT IN USE.
• AREA FOR CLAY-PROJECT RACKS ON CASTERS
• LOCKING STORAGE FOR MODEL SUPPLIES AND BELONGINGS
• CUBBY FOR (18) 22"x30"x1/2" THICK DRAWING BOARDS
• (10) COAT HOOKS
• STORAGE FOR STUDENT BELONGINGS TO KEEP WORK SURFACES CLEAR
• (1) PAD-LOCKED SHORT-TERM INSTRUCTOR'S LOCKER

LIGHTING & CEILING
• DAY-LIGHT BALANCED WALL-WASH LIGHTING (5000-5500 K) AT TACKABLE SURFACES
• DIRECT /INDIRECT GENERAL LIGHTING
• CEILING OPEN-TO DECK IS PREFERRED
• ADJUSTABLE STAGE-TYPE LIGHTING AT MODEL PLATFORM

POWER & DATA
• WIRELESS CONNECTIVITY THROUGHOUT
• HARD-WIRE POWER & DATA CONNECTIONS AT COMPUTER STATIONS
• HARD-WIRE POWER & DATA CONNECTIONS AT INSTRUCTOR'S TABLE

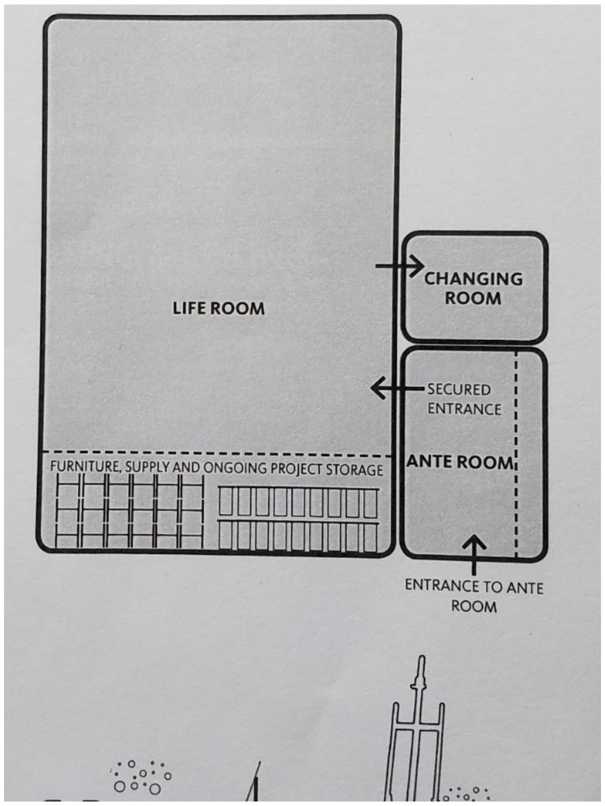
FINISHES & SURFACES
• TACKABLE WALL SURFACE

• WRITABLE SURFACE AT TEACHING AREAS (MARKER BOARDS OR "IDEA PAINT")
• BLOCKING AT SELECT WALLS FOR MOUNTING PROJECTS AND EQUIPMENT
• HARD-SURFACE FLOORING

NOTES
• BLACK-OUT SHADES AT WINDOWS
• EXHAUST / VENTILATION REQUIREMENTS

LOCATION: No proximity to fabrication facility required

DIAGRAM: *Space Type*



SPACE NAME: *Traditional Drawing*

DESCRIPTION

This classroom is intended for the instruction of traditional drawing. Students use a drawing horse and work from still life objects. This room is a priority for natural light. Black-out shades should be provided at windows to control lighting when required.

QUALITIES: Messy, Closed

QUANTITY: 1

NUMBER OF OCCUPANTS: 18 + 1

SIMILAR TO:

CATEGORY: Classroom

AMENITIES:

FURNITURE
• (18) DRAWING HORSES
• (4) EASELS
• (6) STOOLS
• (2) 4" X 4" X 30" OBJECT PLATFORM ON LOCKING CASTERS
• (3) 4' W X 6' H LARGE-SCALE WORK APPARATUSES
• (1) INSTRUCTOR'S TABLE

EQUIPMENT
• (1) E001: CEILING MTD. PROJECTOR*
• (1) E002: PROJECTION SCREEN*
• *(1) ALTERNATE FLAT SCREEN TV
• (2) E004: SPEAKER
• (2) E110: COMPUTER (1 FOR STUDENTS, 1 AT

INSTRUCTOR'S TABLE)
• (1) E006: SCANNER
• (1) SINK

STORAGE
• CUBBY FOR (18) 22"x30"x1/2" THICK DRAWING BOARDS
• (10) COAT HOOKS
• STORAGE FOR STUDENT BELONGINGS TO KEEP WORK SURFACES CLEAR
• PAD-LOCKED INSTRUCTOR STORAGE

LIGHTING & CEILING
• DAY-LIGHT BALANCED WALL-WASH LIGHTING (5000-5500 K) AT TACKABLE SURFACES
• DIRECT / INDIRECT GENERAL LIGHTING
• CEILING OPEN-TO DECK IS PREFERRED
• ADJUSTABLE STAGE-TYPE LIGHTING AT PLATFORMS

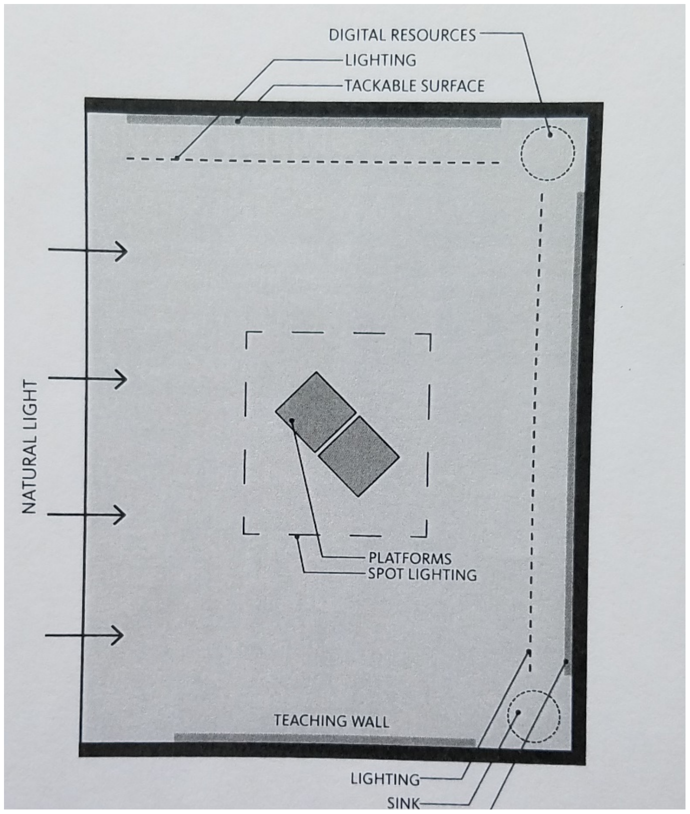
POWER & DATA
• WIRELESS CONNECTIVITY THROUGHOUT
• HARD-WIRE POWER & DATA CONNECTIONS AT COMPUTER STATIONS
• HARD-WIRE POWER & DATA CONNECTIONS AT INSTRUCTOR'S TABLE

FINISHES & SURFACES
• TACKABLE WALL SURFACE
• WRITABLE SURFACE AT TEACHING AREAS (MARKER BOARDS OR "IDEA PAINT")
• HARD-SURFACE FLOORING
• BLOCKING AT SELECT WALLS FOR MOUNTING PROJECTS AND EQUIPMENT

NOTES
• BLACK-OUT SHADES AT WINDOWS
• EXHAUST /VENTILATION REQUIREMENTS

LOCATION: No proximity to fabrication facility required

DIAGRAM: *Space Type*



SPACE NAME: Traditional Drawing

DESCRIPTION

This classroom is intended for the instruction of graphic design and other digital design classes. There should be computer stations for all students with lay-out space adjacent to the computer stations. There should be areas for students to meet and collaborate adjacent to computer stations.

QUALITIES: Clean, Closed

QUANTITY: 1-2 (DEPENDING ON QUANTITY OF INTEGRATED DESIGN STUDIOS

NUMBER OF OCCUPANTS: 18 +1

SIMILAR TO:

CATEGORY: Classroom

AMENITIES:

FURNITURE
- (18) COMPUTER WORKSTATIONS WITH MEETING SPACE ADJACENT FOR GROUPS OF 4 STUDENTS
- (19) CHAIRS
- (1) INSTRUCTOR'S TABLE

EQUIPMENT
- (1) E001: CEILING MTD. PROJECTOR*
- (1) E002: PROJECTION SCREEN*
- *(1) ALTERNATE FLAT SCREEN TV
- (2) E004: SPEAKER

• (19) EII0: COMPUTER (18 FOR STUDENTS, 1 AT INSTRUCTOR'S TABLE).
• (1) E006: SCANNER

STORAGE
• (10) COAT HOOKS
• STORAGE FOR STUDENT BELONGINGS TO KEEP WORK SURFACES CLEAR
• (1) PAD-LOCKED SHORT-TERM INSTRUCTOR'S LOCKER

LIGHTING & CEILING
• DAY-LIGHT BALANCED WALL-WASH LIGHTING (5000-5500 K) AT TACKABLE SURFACES

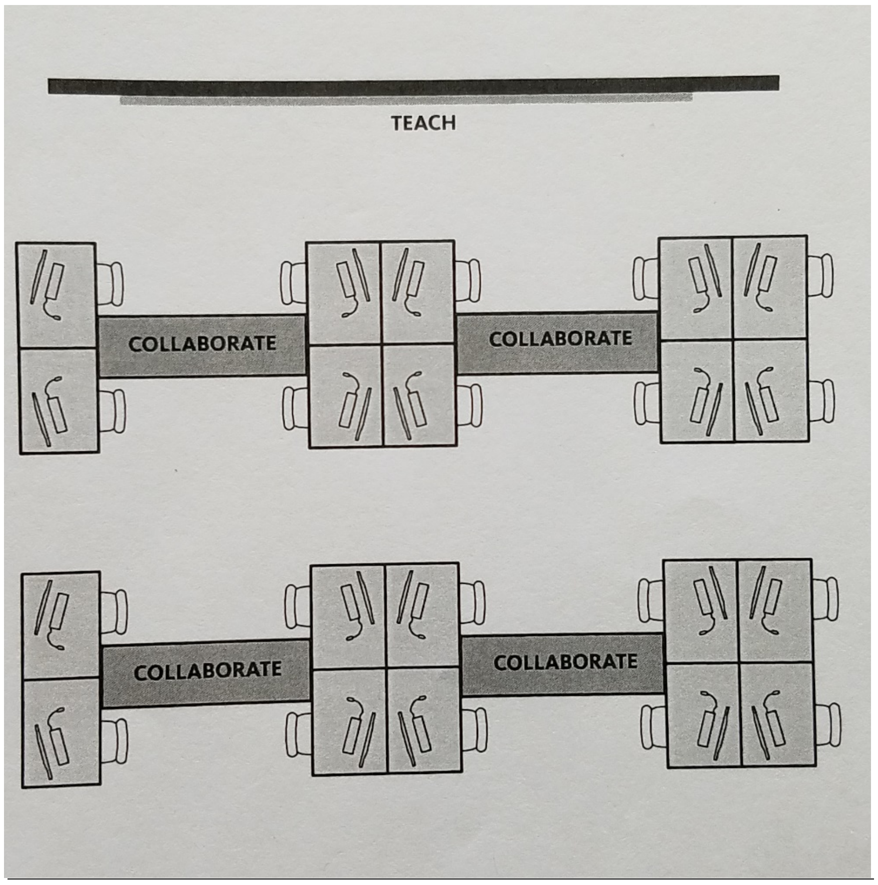
POWER & DATA
• WIRELESS CONNECTIVITY THROUGHOUT
• HARD-WIRE POWER & DATA CONNECTIONS AT COMPUTER STATIONS
• HARDWIRE POWER & DATA CONNECTIONS AT INSTRUCTOR'S TABLE

FINISHES & SURFACES
• TACKABLE WALL SURFACE
• WRITABLE SURFACE AT TEACHING AREAS (MARKER BOARDS OR "IDEA PAINT")
• HARD-SURFACE FLOORING

NOTES
• BLACK-OUT SHADES AT WINDOWS

LOCATION: *No proximity to fabrication facility required*

DIAGRAM: *Space Type*


TEACH
COLLABORATE
COLLABORATE
COLLABORATE
COLLABORATE

APPENDIX 2

```
FILE_DESCRIPTION (
        ('TestProject Long'),
        '2;1');
FILE_NAME (
        'TestProject.ifc',
        '2015/11/04T20:21:10',
        ('test'),
        ('test@test.com'),
        'UFPOR APP 0.0.1',
        'UFPOR DEMO beta',
        'test.com');
FILE_SCHEMA (('IFC4RC4'));
ENDSEC;
DATA;
#1= IFCRELDEFINESBYPROPERTIES(0ZVvjO83j4bxxwT$dbeXpg, *, *, *, (#3), #4);
#2= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 200.0, *);
#3= IFCPROJECT ('340ryhzQ15d8rqO7t8Yu1r', $, TestProject, $, $, TestProject Long, $,
$, #6);
#4= IFCELEMENTQUANTITY(3w8fL5RwDAmfsGxPvt1i4g, *, *, *, *, (#12));
#5= IFCSIUNIT(*, .VOLUMEUNIT, $, .CUBIC_METRE);
#6= IFCUNITASSIGNMENT((#9, #10, #5));
#7= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#11, #2), LOGICALAND, REQUIREMENT,
*);
#8= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #7, (#12));
#9= IFCSIUNIT(*, .AREAUNIT, $, .SQUARE_METRE);
#10= IFCSIUNIT(*, .LENGTHUNIT, $, .METRE);
#11= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 400.0, *);
#12= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);
```

```
ENDSEC;
END-ISO-10303-21;
```

APPENDIX 3

```
    ENDSEC;
    END-ISO-10303-21;
    ISO-10303-21;
    HEADER;
    FILE_DESCRIPTION (
        ('TestProject Long'),
        '2;1');
    FILE_NAME (
        'TestProject.ifc',
        '2015/11/05T14:11:44',
        ('test'),
        ('test@test.com'),
        'UFPOR APP 0.0.1',
        'UFPOR DEMO beta',
        'test.com');
    FILE_SCHEMA (('IFC4RC4'));
    ENDSEC;
    DATA;
    #1= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#44, #53), LOGICALAND,
REQUIREMENT, *);
    #2= IFCQUANTITYLENGTH(FinishFloorHeight, *, *, 0.0, *);
    #3= IFCQUANTITYLENGTH(NetPerimeter, *, *, 0.0, *);
    #4= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #49, (#31));
    #5= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #33, (#59));
    #6= IFCPROPERTYSINGLEVALUE('GrossPlannedArea', '*', '1400.0', *);
    #7= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#23, #22), LOGICALAND,
REQUIREMENT, *);
    #8= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);
    #9= IFCPROPERTYSINGLEVALUE('HandicapAccessible', '*', 'TRUE', *);
```

```
        #10= IFCPROPERTYSET(0p4LQrO9jEiQYpy0hCfVr0, *, *, *, (#14, #63, #29, #6, #54,
#39, #9));
        #11= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #7, (#25));
        #12= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#65, #62), LOGICALAND,
REQUIREMENT, *);
        #13= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#32, #36), LOGICALAND,
REQUIREMENT, *);
        #14= IFCPROPERTYSINGLEVALUE('ReferenceId', '*', '1234', *);
        #15= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#35, #46), LOGICALAND,
REQUIREMENT, *);
        #16= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2800.0, *);
        #17= IFCSIUNIT(*, .AREAUNIT, $, .SQUARE_METRE);
        #18= IFCRELDECLARES(1BKBkR1oXBLQDoYKJ1$UFy, *, 'null', 'null', #27, (#38));
        #19= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #42, (#55));
        #20= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #61, (#68));
        #21= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#47, #60), LOGICALAND,
REQUIREMENT, *);
        #22= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 200.0, *);
        #23= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 400.0, *);
        #24= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 2200.0, *);
        #25= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);
        #26= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #12, (#64));
        #27= IFCPROJECT ('04b4oIU$D5suhk5KYRrzoO', $, TestProject, $, $, TestProject Long,
$, $, #56);
        #28= IFCSIUNIT(*, .VOLUMEUNIT, $, .CUBIC_METRE);
        #29= IFCPROPERTYSINGLEVALUE('IsPublic', '*', 'TRUE', *);
        #30= IFCQUANTITYAREA(NetCeilingArea, *, *, 0.0, *);
        #31= IFCQUANTITYAREA(GrossWallArea, *, *, 0.0, *);
        #32= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 12.0, *);
```

```
        #33= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#69, #43), LOGICALAND,
REQUIREMENT, *);

        #34= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 1200.0, *);

        #35= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 500.0, *);

        #36= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 10.0, *);

        #37= IFCELEMENTQUANTITY(0I2U$B16j2MxoYXUo7hwhi, *, *, *, *, (#59, #8, #31, #55,
#30, #68, #45, #2, #64, #3));

        #38= IFCSPACETYPE ('3i8EHLgFP9nu3Av2oYJaVb', *, name!, description, *, (#10, #37),
*, *, *, EXTERNAL, longname);

        #39= IFCPROPERTYSINGLEVALUE('PubliclyAccessible', '*', 'FALSE', *);

        #40= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #13, (#45));

        #41= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#51, #34), LOGICALAND,
REQUIREMENT, *);

        #42= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#66, #24), LOGICALAND,
REQUIREMENT, *);

        #43= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 2400.0, *);

        #44= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 12.0, *);

        #45= IFCQUANTITYLENGTH(FinishCeilingHeight, *, *, 0.0, *);

        #46= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 400.0, *);

        #47= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2600.0, *);

        #48= IFCELEMENTQUANTITY(3NC66Q7AvCAfKNT4gwSr3W, *, *, *, *, (#25));

        #49= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#16, #58), LOGICALAND,
REQUIREMENT, *);

        #50= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #1, (#2));

        #51= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 1400.0, *);

        #52= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #41, (#30));

        #53= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 10.0, *);
```

```
#54= IFCPROPERTYSINGLEVALUE('NetPlannedArea', '*', '1200.0', *);
#55= IFCQUANTITYAREA(NetWallArea, *, *, 0.0, *);
#56= IFCUNITASSIGNMENT((#28, #71, #17));
#57= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #15, (#3));
#58= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 2400.0, *);
#59= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);
#60= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 2200.0, *);
#61= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (), LOGICALAND, REQUIREMENT, *);
#62= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 300.0, *);
#63= IFCPROPERTYSINGLEVALUE('IsExternal', '*', 'FALSE', *);
#64= IFCQUANTITYLENGTH(GrossPerimeter, *, *, 0.0, *);
#65= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 400.0, *);
#66= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2600.0, *);
#67= IFCRELDEFINESBYPROPERTIES(3iRJFZcpT8rBp3s9rKQUOB, *, *, *, (#27), #48);
#68= IFCQUANTITYLENGTH(Height, *, *, 0.0, *);
#69= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2800.0, *);
#70= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #21, (#8));
#71= IFCSIUNIT(*, .LENGTHUNIT, $, .METRE);
ENDSEC;
END-ISO-10303-21;
```

APPENDIX 4

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (
        ('TestProject Long'),
        '2;1');
FILE_NAME (
        'TestProject.ifc',
        '2015/11/05T14:31:16',
        ('test'),
        ('test@test.com'),
        'UFPOR APP 0.0.1',
        'UFPOR DEMO beta',
        'test.com');
FILE_SCHEMA (('IFC4RC4'));
ENDSEC;
DATA;
#1= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 300.0, *);
#2= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO, null,
400.0, *);
#3= IFCRELDECLARES(04IXQz60n5$R3IfFDsai00, *, 'null', 'null', #29, (#39));
#4= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #57, (#12));
#5= IFCSIUNIT(*, .LENGTHUNIT, $, .METRE);
#6= IFCQUANTITYAREA(NetCeilingArea, *, *, 0.0, *);
#7= IFCQUANTITYAREA(GrossWallArea, *, *, 0.0, *);
#8= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 2400.0, *);
#9= IFCPROPERTYSINGLEVALUE('IsPublic', '*', 'TRUE', *);
#10= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2800.0, *);
```

```
        #11= IFCRELDEFINESBYPROPERTIES(0g2b181mTF5uuwbjGrZw_c, *, *, *, (#29), #48);

        #12= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);

        #13= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #40, (#55));

        #14= IFCPROPERTYSET(0zzoOA49nFC9E75Y83OvCb, *, *, *, (#58, #27, #9, #15, #25,
#24, #68));

        #15= IFCPROPERTYSINGLEVALUE('GrossPlannedArea', '*', '1400.0', *);

        #16= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 400.0, *);

        #17= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 10.0, *);

        #18= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#30, #8), LOGICALAND,
REQUIREMENT, *);

        #19= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#56, #43), LOGICALAND,
REQUIREMENT, *);

        #20= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);

        #21= IFCQUANTITYLENGTH(Height, *, *, 0.0, *);

        #22= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #64, (#72));

        #23= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 2400.0, *);

        #24= IFCPROPERTYSINGLEVALUE('PubliclyAccessible', '*', 'FALSE', *);

        #25= IFCPROPERTYSINGLEVALUE('NetPlannedArea', '*', '1200.0', *);

        #26= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #67, (#21));

        #27= IFCPROPERTYSINGLEVALUE('IsExternal', '*', 'FALSE', *);

        #28= IFCSIUNIT(*, .AREAUNIT, $, .SQUARE_METRE);

        #29= IFCPROJECT ('06UCsZcYj03u4CthbWZeds', $, TestProject, $, $, TestProject Long,
$, $, #60);

        #30= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2800.0, *);

        #31= IFCRELDEFINESBYTYPE(null, *, 'null', 'null', #39, (#70));

        #32= IFCQUANTITYLENGTH(FinishCeilingHeight, *, *, 0.0, *);

        #33= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2600.0, *);

        #34= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#2, #1), LOGICALAND, REQUIREMENT,
*);
```

#35= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO, null, 2200.0, *);

#36= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO, null, 10.0, *);

#37= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #34, (#41));

#38= IFCELEMENTQUANTITY(1kc59Wurf2P94ZkRLMc4lu, *, *, *, *, (#20, #55, #7, #74, #6, #21, #32, #72, #41, #51));

#39= IFCSPACETYPE ('null', *, name!, description, *, (#14, #38), *, *, SPACE_COMPLEX, EXTERNAL, longname);

#40= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#62, #35), LOGICALAND, REQUIREMENT, *);

#41= IFCQUANTITYLENGTH(GrossPerimeter, *, *, 0.0, *);

#42= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#52, #36), LOGICALAND, REQUIREMENT, *);

#43= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO, null, 1200.0, *);

#44= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #73, (#7));

#45= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#66, #16), LOGICALAND, REQUIREMENT, *);

#46= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO, null, 12.0, *);

#47= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#33, #50), LOGICALAND, REQUIREMENT, *);

#48= IFCELEMENTQUANTITY(27FG_JMU1CG9Nt_yxGbhfP, *, *, *, *, (#12));

#49= IFCRELAGGREGATES(0BL$j4TGb6OPbWUBwHibvJ, *, 'null', 'null', #29, (#70));

#50= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO, null, 2200.0, *);

#51= IFCQUANTITYLENGTH(NetPerimeter, *, *, 0.0, *);

#52= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO, null, 12.0, *);

#53= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #42, (#32));

#54= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #45, (#51));

#55= IFCQUANTITYAREA(NetFloorArea, *, *, 0.0, *);

#56= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO, null, 1400.0, *);

```
        #57= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#59, #71), LOGICALAND,
REQUIREMENT, *);

        #58= IFCPROPERTYSINGLEVALUE('ReferenceId', '*', '1234', *);

        #59= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 400.0, *);

        #60= IFCUNITASSIGNMENT((#69, #5, #28));

        #61= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #19, (#6));

        #62= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 2600.0, *);

        #63= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #47, (#74));

        #64= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#46, #17), LOGICALAND,
REQUIREMENT, *);

        #65= IFCRESOURCECONSTRAINTRELATIONSHIP(null, null, #18, (#20));

        #66= IFCMETRIC ('MAX_VALUE', null, HARD, null, *, *, *, LESSTHANOREQUALTO,
null, 500.0, *);

        #67= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (), LOGICALAND, REQUIREMENT, *);

        #68= IFCPROPERTYSINGLEVALUE('HandicapAccessible', '*', 'TRUE', *);

        #69= IFCSIUNIT(*, .VOLUMEUNIT, $, .CUBIC_METRE);

        #70= IFCSPACE('2Z0l10u8z8kQd1Dt_8AUvf', $, $, $, $, $, $, $, COMPLEX, $, $);

        #71= IFCMETRIC ('MIN_VALUE', null, HARD, null, *, *, *, GREATERTHANOREQUALTO,
null, 200.0, *);

        #72= IFCQUANTITYLENGTH(FinishFloorHeight, *, *, 0.0, *);

        #73= IFCOBJECTIVE ('null', *, HARD, *, *, *, *, (#10, #23), LOGICALAND,
REQUIREMENT, *);

        #74= IFCQUANTITYAREA(NetWallArea, *, *, 0.0, *);

        ENDSEC;

END-ISO-10303-21;
```