NOTE TO USERS

REPRESENTATION AND REASONING
FOR
INTEGRATED CONCEPTUAL DESIGN OF BUILDING STRUCTURES

Rodrigo Mora

A Thesis in the Department of Building, Civil and Environmental Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Building Studies)
At Concordia University
Montreal, Quebec, Canada

Your file Votre référence
ISBN: 0-494-05719-X
Our file Notre référence
ISBN: 0-494-05719-X

ABSTRACT

Representation and Reasoning
for Integrated Conceptual Design of Building Structures

Rodrigo Mora, Ph.D.
Concordia University, 2005

During conceptual design the most salient characteristics of a design artefact are defined. A team of specialists is usually required to carry out this process. For buildings, the architect is the lead designer and his/her work is supported by engineers and other specialists. The distribution of work among specialists with different backgrounds and priorities and the lack of a shared understanding of each participant's view usually result in inefficiencies, redundancies, omissions and errors in the design. This translates in cost overruns, poor quality and lack of competitiveness of the building industry. Computer programs for performing structural analysis and detail design calculations have been in the market and used by practitioners for many years. However, there is still a lack of computer support to assist properly engineers in the conceptual design of building structures. One explanation for this lack of support lies in the fact that during conceptual design the engineer's work flow is highly dependent on the amount, quality and type of information that is exchanged with the architect. To address this problem, the interactions between architects and structural engineers during early design are studied and formalized in a model of conceptual design of building structures. In this model, two stages of structural support are identified which are differentiated by the type and refinement of the architectural design representations. The first stage corresponds to the earliest phases of the architectural design process in which the architectural representations are tentative

ambiguous and imprecise, whereas in the second stage an unambiguous but still imprecise three-dimensional computer model of the building architecture is made available to the engineer. This research project concentrates on the second stage of computer support. A methodology for architecture/structure integration is devised for this stage. For the implementation of this methodology a computer representation integrating the building architecture and the structural system is developed. Synthesis algorithms are also proposed that rely mostly on geometry and topology of the architecture and the structure to help identify structural opportunities, detect potential structural problems and respect constraints coming from the building architecture. A proof-of-concept software prototype is developed and implemented, and two test cases demonstrate the validity of the proposed approach for supporting conceptual structural design. It is expected that the results from this research project will lead to more integrated and efficient conceptual structural design practice for improved building performance.

ACKNOWLEDGMENTS

I am deeply grateful to my supervisors, Dr. Hugues Rivard and Dr. Claude Bédard for their guidance, constructive advice, thoughtful recommendations, and financial support.

I also acknowledge the Natural Science and Engineering Research Council (NSERC) of Canada, as well as the "Fonds Québécois de Recherche sur la Nature et les Technologies" (FQRNT) of the province of Quebec, for financially supporting this research.

My deepest appreciation for my wife Claudia and my daughter Maria-Gabriela for reflecting the love, beauty and joy of life that motivates me along the way, and to my mother that, even from far away, never stops worrying and taking care of me.

Finally, I want to thank all my friends in the Centre for Building Studies for always being there giving me support and counting on me for the fun times.

Dedicated
to the memory
of my father,
Josué Mora

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 2. Literature review

Chapter 4. Formal model of conceptual structural design

Chapter 5. Design assistance environment for conceptual structural design

Chapter 6. Prototype implementation

Chapter 7. Testing and validation

Appendix B. Geometric modeling and reasoning techniques

Appendix C. *Synthesis Algorithms*

LIST OF TABLES

TERMINOLOGY

Entity – An entity is defined in the dictionary as something that has separate and distinct existence and objective of conceptual reality (Merriam-Webster Dictionary 2005). In this thesis, the term entity is used to refer to a concept, that can be physical or not, necessary for describing a building in computerized form. The terms "entity" and "object" are used interchangeably.

Function – Function is defined by Rosenman and Gero (1996) as what an object is intended to do (e.g. transfer loads) in response to a purpose by a designer (e.g. to design a safe, serviceable and efficient structural system).

Heuristic thresholds – Refer to ranges of applicability or usability of particular construction technologies under given conditions, which are determined based on rules of thumb from knowledge and experience (e.g. floor span thresholds).

Integration – From the Cambridge Dictionary (2005), integration is the act or process of combining two or more entities together so that they can perform effectively as a single unit.

Integrity – The Merriam-Webster Dictionary (2005) defines integrity as the quality or state of being complete or undivided, free from flaw or error. Soundness and completeness are used as synonyms.

Intent (Design intent) – The term design intent is used in this thesis to refer to the individual vision and goals of a designer that are anticipated in a particular design.

Model – A model is an abstraction or simplification of some part of reality that can be used for realization, simulation or as a prototype for testing (Turk 2001). A model can describe a physical entity such as a building, a method, or a process.

Purely functional entity – As described by Rosenman and Gero (1996), a purely functional entity is an entity that performs an intended function as part of a design, but does not have a material and physical existence (e.g. a space).

Representation – A representation is a digital specification of a building. It describes the different entities that make up a building along with the manner in which they interact (Khemlani et al. 1998). Representations are used for constructing models. A synonym is a schema.

Reasoning – In the field of artificial intelligence reasoning is defined as the logical process that is used to derive conclusions or additional facts from known facts.

Regular (regularity) –. The Cambridge Dictionary (2005) the term regular is described as existing or happening repeatedly in a fixed pattern, with equal or similar amounts of space or time between one and the next. While irregular means having parts with different shapes and sizes. In this thesis the term regularity considers two aspects: (1) the external building form and (2) the geometry and internal layout of spaces and physical elements that make up the building.

System – The Cambridge Dictionary (2005) defines system as a set of interacting or interdependent components or devices that operate together with a particular purpose, and form a unified whole. A system is considered as a functional entity.

Tangible building entity – A tangible building entity is a functional entity that has a material and physical existence. In this thesis the terms "tangible building entity" and "physical building entity" are used interchangeably.

View – In this thesis, the term view is used to refer to the interpretation that a designer gives to a particular building component or assembly so that depending on the view, certain properties and descriptions become relevant (Rosenman and Gero 1996).

Virtual Building – In general, a virtual building is a computer model. In particular, it is a formalized digital description of an existing or planned building that is used to simulate and communicate the behavior of the real building in its expected contexts (Christianson 2000). A virtual building is created in a computer using the entities from an underlying representation. In this Thesis the terms "virtual building" and "digital building model" are used interchangeably.

TYPEFACES

Roman, 12 pt, Bold, Italic

Identifies the components of the approach proposed in this research, namely: an ***Integrated Representation***, ***Synthesis Algorithms*** and a ***Geometric Modeling Kernel***. The above components are the main modules of a proof-of-concept software prototype that has been implemented and therefore this typeface is also used to identify all the prototype modules.

Roman, 12 pt, Italic

Identifies each ***Synthesis Algorithm*** developed in this research project. Examples: the algorithms *verifyWallContinuity* and *uniformDepth*.

`Courier, 10 pt, Bold`

Identifies classes from object-oriented programming. Examples: the classes **`Column`** and **`Wall`** that belong to the ***Integrated Representation***.

`Courier, 10 pt, Italic`

Identifies the attributes and methods (or functions) from object-oriented classes. Examples: the attribute *`gravityLoadIntensity`* and the methods *`setGravityLoadIntensity`* and *`computeTotalGravityLoad`*.

`Courier, 10 pt`

Identifies the entities from the ***Geometric Modeling Kernel***. Examples: `wires`, `faces` and `edges`.

CHAPTER 1

INTRODUCTION

Design can be defined as the process of creating the complete description of a proposed artifact that satisfies some specific needs (Gero 1986). A team of specialists may be required to carry out this process, where the number of members in the team usually depends on the complexity of the artifact to be designed. For buildings in traditional work organizations, the architect is the lead designer and his/her work is supported by engineers and other specialists. Various firms of specialists participate in the design of a given building, which often turns out to be unique. Time wise, design is an iterative process that follows three stages of refinement, namely: conceptual design, preliminary design and detailed design. The subject of this research project is to provide computer support for the conceptual design of building structures. The next section provides a brief description of the problem of conceptual design. Then, the main motivation for this research project is presented, followed by its goal. Next the scope of this project and its limitations are briefly discussed. Finally, the overall layout of the thesis is presented.

1.1 Introduction to conceptual design

During conceptual design the most salient characteristics of an artefact are defined. Thus, major decisions are made regarding the building architecture, such as the internal configuration of spaces and physical elements that give shape to the building form, as well as major aspects of the supporting engineering systems, such as materials, type and

layout. These decisions have great impact on the constructability, cost, and overall performance of a building (Fenves et al. 2000).

Conceptual design is explorative in nature which means that changes should be the norm rather than the exception. During this stage, designers reflect upon fundamental design concepts, and not details, to come up with rough design alternatives which are produced relying on poor, inexact and missing information. Each alternative requires minimal resource commitment as it can be discarded easily and start anew (Lipson and Shpitalni 2000). In addition during conceptual design, the goals are defined only partially and procedures for obtaining solutions are not known completely (Cross 1989). This makes designers rely on their own knowledge and experience for tackling conceptual design problems. The above considerations therefore make providing computer support for conceptual design a challenging task.

1.2 Motivation

Timely engineering feedback to the architect is required early on during the design process. Computer programs for performing structural analysis and detail design calculations have been in the market and used by practitioners for many years. However, there is still a lack of computer programs available to assist properly engineers and architects in the conceptual design of building structures. The main reason for this lack of support lies in the fact that during conceptual design the engineer's work flow is highly dependent on the amount, quality and type of information that is exchanged with the architect. This in turn affects the quality and effectiveness of the structural engineering

feedback provided to the architect. During conceptual design the engineer synthesizes alternative structural layouts while considering multiple conflicting building design criteria coming from the different participants involved in the building design process.

The distribution of work among specialists with different backgrounds and priorities as well as the lack of a shared understanding of each participant's view usually result in inefficiencies, redundancies, omissions and errors in the design. This translates in cost overruns, poor quality and lack of competitiveness for the entire building industry (Froese 1994). Over the last three decades, advanced communication and computing technologies have emerged as key elements for improving quality, efficiency and competitiveness in the building design and construction industry. This research project proposes an innovative computer-based approach that aims at assisting engineers and architects in integrating structural solutions to early architectural designs. Therefore, it emphasizes only the structural and architectural aspects of the building design process.

This research is part of an effort that has been carried out at the Center for Building Studies at Concordia University for several years that aims at supporting the early stages of the building life cycle through computers. For example, in the area of conceptual and preliminary building design several papers have been published, such as Bédard and Gowri (1990) and Bédard and Ravi (1991). Teamwork and collaboration, for well-integrated building design and construction, have also been of great concern as seen, for example, in Rivard et al. (1995) and Mokhtar et al. (1998). This research is also a continuation of the work by Rivard and Fenves (2000a) at Carnegie Mellon University, whose work in design environments for conceptual design of building structures lays down the foundation that marks the beginning of this research project.

1.3 Goal

The goal of this research project is to improve current design practice by proposing and validating a computer-based approach that more effectively supports the integration of structural solutions to early architectural designs. This goal is part of a broader one which is to facilitate collaboration between architects and engineers during the early stages of the design process, which will hopefully result in improved building performance.

From a computer perspective, the problem of conceptual design of building structures can be treated as two closely coupled sub-problems:

a) Develop a representation of the physical system being designed - Several research and standardization efforts have been invested over the last two decades to develop standard representations for the structural system and the building architecture. However, such proposed representations are generally not geared towards supporting conceptual design.

b) Develop computational methods for solving conceptual design problems - Over the last three decades researchers have focused on using artificial intelligence (AI) techniques to search for satisfactory solutions to conceptual design problems. For example the following techniques have been proposed: expert systems, formal logic, shape grammars, case-based reasoning (CBR) systems, fuzzy logic, evolutionary algorithms and hybrid systems that combine two or more AI techniques. The main drawback of most proposed AI-based solutions is that they tend to minimize the impact of the architectural design on the structural synthesis process. In addition, some of them simplify the representation of the building architecture in order to adjust to the requirements of the selected problem-

solving technique(s). This is in sharp contrast with current trends in building design practice. As stated by Taranath (1998), until the early 1970s the engineer had considerable influence in keeping the building architectural form simple. However with the availability of computers, the structural engineer can now analyze more complex forms than before, thus relaxing the structural constraints for architects.

In this research project both a representation of the physical system and computational problem solving methods are developed. Since the structural system is an inherent part of the building architecture, the representation of physical system integrates the building architecture and the structural system. The main premise of this research project is that in order to properly support the conceptual design of building structures, and enable timely engineering feedback to the architect, computers must allow engineers and architects to perform conceptual structural design within a building architectural context without interrupting the creative workflow of the architect. Hence, a formal model of interaction between architects and engineers and a methodology for Architecture/Structure (A/S) integration during conceptual design are needed. These constitute the basis for a design assistance environment applicable to structures at the conceptual stage.

1.4 Research scope

This research project aims at supporting conceptual structural design for most types of modern buildings encountered today, such as office, apartment, institutional and mixed-use buildings made out of steel and/or concrete. However, sculptural and other symbolic buildings are not meant to be supported by the approach, such as: museums, stadiums and airports.

1.5 Limitations

The limitations of this research project are divided in two groups: first the general limitations on the scope of this research which have been identified before beginning the research, and second the specific limitations on the complexity of the designs.

1.5.1 General limitations

- ***Changes*** - Collaboration during the building design process may result in design refinements (i.e. more elaborate descriptions of components or systems, and more components or systems being added) and/or design changes (i.e. removing components or systems or modifying their descriptions). No change-enabling mechanism is considered in this research. The focus is therefore on enabling design refinement through collaboration.

- ***Modeling knowledge*** – A knowledge-based reasoning component organizes structural engineering knowledge that can be used for assisting the decision-making process and let the computer communicate competently with the engineer and the architect. In this research project however, the reasoning during conceptual design is based mostly on the geometry and topology of the physical system being designed and only little knowledge is embedded in the system.

- ***Management, analysis and evaluation of design alternatives*** – This research focuses on facilitating the generation of structural alternatives by the engineer with the help of the computer. However, no simplified analysis methods or evaluation mechanisms

are studied for assessing the feasibility and comparing those design alternatives. Should structural analysis be required, it is expected that the results from the structural synthesis process can be straightforwardly transferred to existing structural engineering applications for assessing structural behaviors.

- ***Load management*** – As a corollary of the previous limitation, loads are not considered in the decision making process. Thus, even though loads can be assigned to entities, loads are not propagated or modeled to ascertain the functionality of structural entities.

1.5.2 Specific limitations

- ***The building architecture*** – A storey is considered to be bounded at the bottom by a single floor slab and at the top by a single ceiling slab, which is not always the case. In modern architecture, stories are often defined by many floor and ceiling slabs at varying elevations from the ground; these are called multi-level stories. Buildings with multi-level stories are not considered in this research.
- ***The structural system*** - The only structural systems that are considered in this research are the ones where the structural elements that belong to the vertical support system are continuous down to the ground. Therefore, transfer structures are not accepted, where vertical load paths are interrupted at intermediate stories and re-directed to different vertical supports in the lower stories.
- ***Grouping*** - Engineers group structural entities based on similarities (e.g. function, material, geometry and position). Grouping of entities simplifies the design process, since it allows the engineer to deal with sets of entities with similar characteristics

instead of dealing with the individual entities. Entity grouping, however, is not considered in this research.

Note that throughout this thesis factors such as "knowledge", "analysis", "evaluation", "loads" and "grouping" are studied only to acknowledge their importance in decision-making during conceptual structural design. However, as a first step towards supporting conceptual design with computers, the goal with this research is to provide a basic level of assistance to competent structural engineers that have a thorough understanding of the structure function, behavior, and available structural technologies, which they use to produce the structural form.

1.6 Layout of the thesis

The remaining chapters of this thesis are organized as follows. Chapter 2 "Literature review" presents a description of the traditional process of conceptual design of building structures, followed by a critical review of state-of-the-art applications as well as fundamental research efforts in the area of conceptual design assistance for building structures and building integration. Based on the literature review, Chapter 3 "Methodology" first presents the objectives and the main premises of this research project, as well as the research method. Then, it introduces the stages that are followed to carry out this project, as well as the main components of a design assistance environment for conceptual structural design. Chapter 4 "Formal model of conceptual structural design", describes interactions between architects and engineers at different stages of refinement during conceptual structural design. This model includes a methodology for

Architecture/Structure (A/S) integration that defines a sequence of activities to be carried out by the engineer in a top-down fashion during conceptual design. Chapter 5 "Design assistance environment for conceptual structural design" presents the organization and development of the main components that are required by a design assistance environment, namely: a representation integrating the building architecture and the structural system and synthesis algorithms for facilitating the engineering reasoning process. Chapter 6 "Software prototype implementation" describes a proof-of-concept prototype that incorporates the main components of the design assistance environment described above. Chapter 7 "Testing and validation of the prototype" introduces two test cases that demonstrate improved assistance to the engineer and the architect during conceptual structural design following the methodology for Architecture/Structure (A/S) integration proposed in Chapter 4, using the design assistance environment for conceptual structural design described in Chapters 5 and 6.

After the summary, contributions and directions for future work given in Chapter 8, the appendices providing related information that may be of interest to readers. Appendix A contains a list of commercial architectural, structural and mechanical design software packages available in the market. Appendix B presents an overview and more recent advances in geometric modeling as well as background information on geometric reasoning techniques, which are at the foundation of the ideas proposed in this research project. Appendix C contains a list of problem-solving algorithms that have been implemented including their function, input arguments and the expected results. Appendix D includes C++ code from main driver program of the prototype.

CHAPTER 2

LITERATURE REVIEW

This chapter is divided in five main sections. The first section provides a description of the structural system with the underlying concepts that are used throughout this thesis. The second section describes the process of conceptual design of building structures. The reasons for including this section are twofold: (1) to clarify the meaning of conceptual design of building structures since various authors give different meanings to this process, and (2) to serve as a foundation in terms of principles and ideas upon which the thesis is developed. Given the background information provided in sections one and two, the third section presents an overview of commercial computer-aided design (CAD) software for architecture as well as software for structural design. Thus, the structural part shows the gaps in computer support, while the architectural part uncovers opportunities for structural integration. The fourth section reviews previous research efforts in supporting conceptual design of building structures and sheds light on why, despite the amount of research outputs, there is still little use of the proposed techniques in actual design practice. The fifth section discusses research at Concordia University in building design integration.

Finally, the conclusion section summarizes and emphasizes the main points of this chapter. It should be noted that it is not possible to study the process of conceptual structural design without considering the interdependency between the structure and the architecture. Therefore, throughout this chapter architectural issues are covered only to the extent that they influence the process of conceptual design of building structures.

2.1 Hierarchical description of the structural system

The structural system is defined as a physical arrangement of interrelated elements positioned in space in which the character of the whole system dominates the interrelationship of the parts (Schodek 2003). Its purpose is to resist the loads applied to the building and transmit them efficiently to the ground. The loads acting on a building are of varied nature and magnitude and are classified as either gravity (vertical) or lateral (horizontal) loads. Gravity loads act downwards and come mainly from the occupancy of the building, the self-weight of building elements (e.g. floors, walls, finishes, partitions, etc.), the self-weight of the structure and environmental forces (e.g. snow and water). Lateral loads act in a horizontal direction and come from inertial forces (associated with earthquake ground motion) and wind forces. The structural system therefore picks up the loads from their point of application and transmits them safely to the ground through a series of load paths which come from the arrangement and connectivity of its physical elements (e.g. slab elements, beams, columns, walls, etc.). Thus, loads propagate throughout the structural system through a network of gravity and lateral load paths.

The above definition of the structural system, which emphasizes its physical nature, corresponds to those found in most structural engineering textbooks and is the basis of commercial structural engineering applications. A broader description of the structural system is considered in this research project. This description makes explicit those concepts that are used particularly during conceptual design and reflect the way engineers understand structures (Lin and Stotesbury 1988). The structural system is thus described as a hierarchy of entities, where each parent entity acting as a whole gives rise to the child entities that make-it up. This description allows breaking down the structural design

problem into sub-problems which can be tackled semi-independently. A hierarchical description of the structural system has been presented by Rivard and Fenves (2000a), which is inspired from the top-down design approach proposed by Lin and Stotesbury (1988). Figure 2.1 illustrates the four levels of this hierarchical description as follows.

2.1.1 Level I – Massings

At the massing level, the structural system is broken down into independent structural volumes depending on the building size and complexity. An independent structural volume models a self-contained structural skeleton. Long or irregularly shaped buildings may have several independent structural volumes that are delimited by expansion joints. An independent structural volume may in turn be composed of one or more structural zones (Lin and Stotesbury 1988). A structural zone corresponds to a portion of the volume that groups spaces with the same or similar functionality that impose uniform structural requirements (e.g. an auditorium or a parking area).

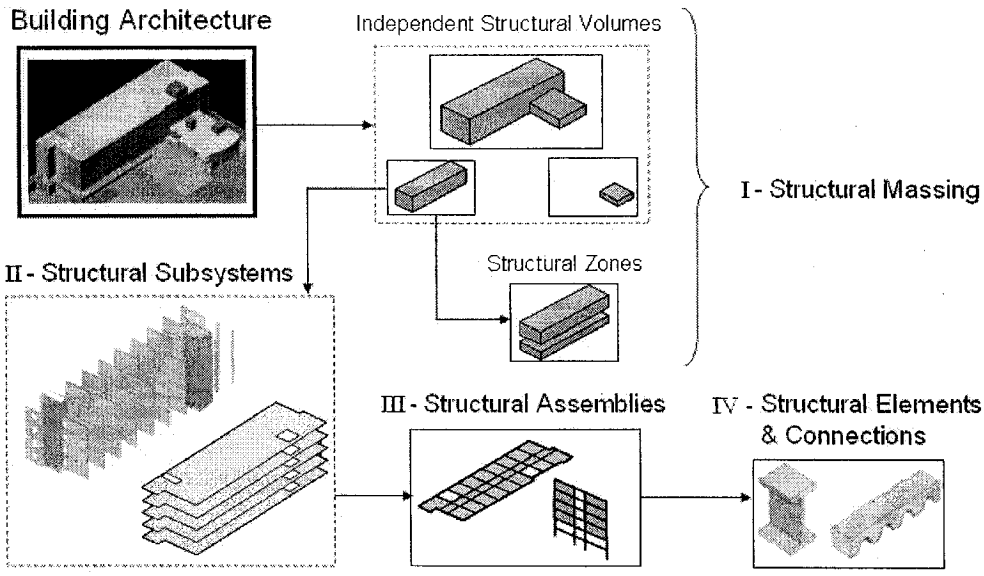


Figure 2.1 The hierarchical description of the structural system

2.1.2 Level II – Subsystems

An independent structural volume has four subsystems: the foundations, the horizontal, the vertical gravity, and the vertical lateral subsystems. The horizontal subsystem is responsible for picking up gravity loads acting on it and transferring these loads to the vertical gravity subsystem. The vertical gravity subsystem is responsible for transferring gravity loads to the foundations subsystem; likewise the vertical lateral subsystem is responsible for transferring lateral loads to the foundations subsystem which in turn is responsible for transferring all the loads from the structure to the ground. Horizontal and vertical subsystems are mutually interacting and complementary (Pillai et al. 1999). As illustrated in Figure 2.1, depending on the problem at hand, the vertical lateral and gravity subsystems may be treated together as a single vertical support subsystem. The foundations subsystem is not considered further in this research project.

2.1.3 Level III – Assemblies

Structural assemblies are arrangements of physical elements. Each structural subsystem consists of a set of structural assemblies having a particular function; therefore, they can be designed and analyzed semi-independently from the rest of the structure. For illustration purposes, the most common structural assemblies are described as follows: a floor assembly spans horizontally between vertical supports; as part of the horizontal subsystem its primary function is to resist applied gravity loads through bending of its horizontal elements. A planar frame assembly (hereinafter called frame assembly) is composed of elements that lie in a vertical plane. A frame assembly may belong to the vertical gravity subsystem (e.g. a simple gravity frame) and possibly to the vertical lateral

subsystem (e.g. a moment resistant frame). Its behavior in resisting loads as part of either of these two subsystems, or both of them, depends on the type of connectivity between its horizontal (beams) and its vertical (columns and/or walls) elements. A truss assembly is part of the horizontal subsystem and could also be part of the vertical subsystem. It is composed of straight elements assembled into triangulated patterns. A truss spans between vertical supports, therefore, it bends as a whole between supports, but its elements are subject to axial forces only (i.e. tension and compression). A column stack is part of the vertical gravity subsystem. It is composed of columns aligned, usually, from the ground to the roof of the building. A wall stack is part of the vertical gravity as well as the vertical lateral subsystems. It is composed of walls aligned on top of each other extending usually from the ground to the roof of the building. A structural assembly can be made of other structural assemblies; for example, the floor assembly can be made of trusses.

2.1.4 Level IV – Elements and connections

The physical arrangement of structural elements (e.g. beams, columns, slab elements, etc.) joined together through structural connections constitute what can be called the physical structural system (or the physical structure). For illustration purposes, the most common structural elements are described as follows: a floor assembly is usually composed of slab elements and beams. Slab elements are the first elements in the gravity load path. They pick up gravity loads and transmit them to the beams that support them. Beams are usually straight horizontal elements used primarily to resist gravity loads through bending. Primary beams are directly supported by columns, thus they are

designed as part of floor assemblies as well as frame assemblies whereas secondary beams are supported by primary beams, thus belonging to floor assemblies only. Planar frame assemblies are composed of columns, primary beams and often walls. Columns are generally vertical and resist primarily axial compressive loads. However, columns belonging to frame assemblies that are moment resistant are subjected to both axial load and bending moment. When they perform a structural function, walls are usually part of coplanar frame assemblies. Due to their planar geometry they resist both lateral and gravity loads.

2.1.5 The geometry of the structural system

In this research project the geometry of the structural system is called structural form, or structural system layout (see Figure 2.2). The structural form can be defined as the physical arrangement of the elements that make up the structural system. The structural form is obtained using available construction technologies and material(s), and includes two coupled aspects:

- ***Structural system geometry:*** describing the actual shape, length, width and cross-sectional dimensions of structural elements.
- ***Structural system topology:*** describing the connectivity among structural elements. This connectivity defines continuous paths to the ground for the loads acting on the structural system.

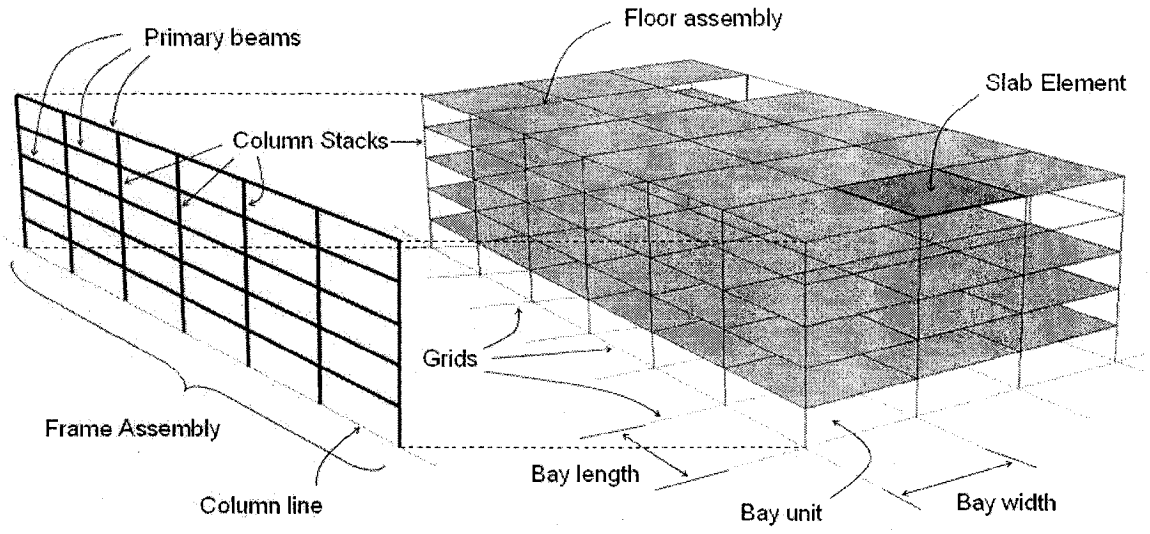


Figure 2.2 The geometry of the structural system

Even though the structural system is three-dimensional (3D), it is typically designed, in response to the building architecture and the applied loads, as a collection of two-dimensional (2D) assemblies lying primarily in the horizontal and vertical planes (Pillai et al. 1999). The elements that make up these assemblies are arranged in regular patterns within the planes of the assemblies to which they belong. As shown in Figure 2.2, the projection of a frame assembly in the horizontal floor plan is a line that is called a column line since it contains the projections of column stacks into column positions in the floor plan. Therefore, the planes of frame assemblies implicitly define column alignments (i.e. column lines) in the floor plan.

Frame assemblies are usually oriented in orthogonal directions to each other in response to the architectural functional organization of spaces. They are expected to resist lateral loads (seismic and wind loads) along their plane of action. Any two orthogonal frames usually intersect at a column stack assembly, and therefore, any column in the structural system usually belongs to two orthogonal frames. In each direction, vertical parallel

frame planes are separated by an horizontal distance that defines a structural bay, which determines the span between two consecutive columns. The combination of all the orthogonal frames defines a structural grid with structural bays running in orthogonal directions. Grid points usually define the location of columns. The rectangle formed by any two orthogonal structural bays is in turn called a bay unit which has therefore a bay width and a bay length.

Structural elements are three-dimensional solids, however for simplicity they can be represented during conceptual design as one-dimensional wires (beams and columns) or two-dimensional planar faces (walls and slab elements), i.e. a mixed-dimensional wire-frame representation. This is possible because structural elements usually have one or two salient dimensions that are sufficient for characterizing them within the structural form since the cross-section of structural elements does not affect their relative position in space.

2.2 Review of the conceptual design process of building structures

This section begins with a brief description of the architectural design process. It summarizes basic architectural concepts that will later be used by the structural engineer particularly during the early stages of the structural design process. Then, the structural design process is described with emphasis on the conceptual stage. Finally, the opportunities for architecture/structure integration are summarized.

2.2.1 Overview of the architectural design process

The primary function of a building is to house and protect people and their activities from the elements. It must provide a safe, comfortable and healthy internal environment for the well being of its occupants. This function is primarily performed through the layout of spaces that make up the building and house its occupants. The architect shapes and positions spaces with respect to each other, and connects them, through doors and circulations, such that they perform their function properly, as part of a building as a whole.

The design of the building form and its internal configuration of spaces are interdependent activities and therefore are interactively performed together. The major determinants of the building form and its internal space layout are the following: the site requirements (shape, size, flatness, orientation, surrounding buildings, site regulations, etc.), the requirements of the building program (which include the intended functionality of spaces), and aesthetic requirements (Arnold and Reitherman 1982).

Spaces in buildings are essentially defined through physical elements that enclose them, either fully or partially. Thiel (1963) calls those physical elements space establishing elements (SEE). SEEs have the purpose of protecting occupants and allowing privacy for people within a space. SEEs are mostly planar surfaces that run vertically (e.g. walls, doors, windows) and horizontally (e.g. ceiling, floor) around the perimeter of a space. SEEs can also run inside spaces, such as partition walls, sometimes defining subspaces. Other less common types of SEEs are screens (i.e. perforated surfaces) and objects such as rows of columns that may confine or divide a space. SEEs are the physical elements that will ultimately shelter the occupants from the elements and give shape to the

building. Spaces that are most explicitly established by SEEs are called primary spaces (e.g. a room), while secondary spaces are groupings of primary spaces, usually having some common functionality. For example, secondary spaces can be used to define zones in a building (e.g. a parking zone, a public zone, or a private zone).

During the architectural design process architects follow certain principles that help them fulfill the design goals, such as: proportion, scale, balance, order, unity, repetition, pattern, etc. Architects seek to conceive well proportioned building forms that give a sense of balance, order and unity while being in good scale internally and with the surrounding environment. In doing so, they look for balance between the outer form and its internal spaces which are configured in relation to each other, usually creating repetitive patterns to produce what architects call "visual rhythm" (Winters 1997). A repetitive pattern is produced when characteristic space dimensions (i.e. functional dimensions) are repeated in the direction where spaces are ordered. Since for convenience spaces are usually rectangular in shape, characteristic space dimensions repeated in orthogonal directions produce orthogonal patterns.

Architects keep record of typical functional dimensions for common spaces, which allows them to anticipate feasible patterns for every new design as required. Therefore, they are able to define in advance orthogonal patterns, called the building grids, that are used as a reference base for organizing the entire building. Other types of organizing patterns, such as radial patterns, may also be used in architectural design; however, the orthogonal pattern is the most widely used. From the cost and constructability standpoints, repetition is a desirable feature in design since it results in physical elements of the same size that can be manufactured and erected more efficiently.

2.2.1.1 Stages of the architectural design process

Architectural design is usually described as an iterative process that follows successive stages of refinement. The main phases of this process are summarized as follows.

a) Architectural programming

The outcome of the architectural programming stage is the architectural program, which is a compilation of the owner's goals, objectives and requirements, and describes the proposed functions and facilities to fulfill them. Furthermore, it establishes cost, performance, operational and quality criteria that will guide designers in later stages (Rivard et al. 1995). Consequently, the architectural program allocates spaces required to house the building activities, and compiles regulations by local authorities that need to be taken into consideration for the design. The requirements coming from the architectural program are called the programmatic requirements for the building.

b) Conceptual design (a synonym is schematic design)

Three main activities that architects perform during conceptual design have been identified as follows (Meniru et al. 2002):

1. *Site analysis* - determination of the building form and orientation in relation to the landscape and adjacent constructions while considering local regulations.
2. *Space composition (a synonym is space layout planning)* - configuration of space layouts (i.e. size, shape and relative location of spaces) in a floor plan (i.e. within stories) to satisfy functional requirements in the architectural program.

3. *Building elements (physical entities)* – select construction materials and lay out construction elements (e.g. walls, columns, doors, windows, floor slabs and roofs) and equipment (e.g. furniture, appliances, etc.).

During conceptual design architects use sketches to explore alternative layouts of spaces and building elements and sharpen the definition of the building form. Sketches are mostly used by architects for exploring ideas. However, they are often used as a means of communicating the architect's design intent to the client and the other participants in the building design process. By definition, sketches are tentative, rough and imprecise, and are meant to represent only the salient features of a design. They are also informal and vague as they represent ideas that may never be materialized. Bubble diagrams are also used with sketches for defining the topology (i.e. connectivity) of spaces during space layout planning. Depending on the complexity of the building, architects may also perform massing studies representing alternative arrangements of the main volumes that make up the overall form of the building in 3D (see Figure 2.3). Nowadays, powerful computer tools have become available that allow the architect to sketch and perform massing studies (for example Architectural Studio by Autodesk).

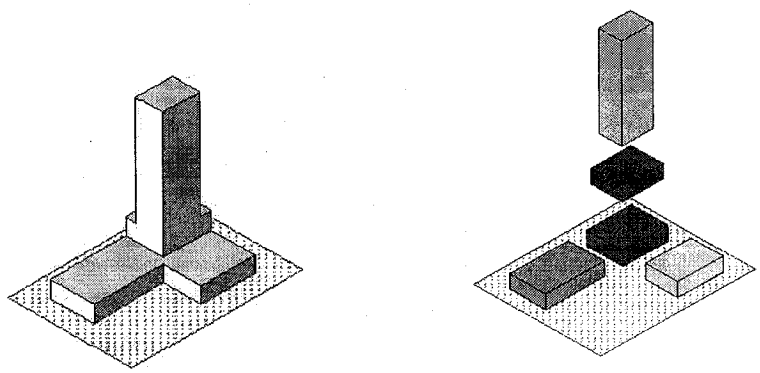


Figure 2.3 Massing studies on the building form

c) Preliminary design

During preliminary design architects refine the most promising alternative(s) and produce hard-line drawings with more accurate dimensions. Physical mockups and three-dimensional computer models are also created. This more refined architectural design information can then be passed to the engineers, for them to design engineering systems.

d) Detailed design (a synonym is design development)

Once the best alternative has been selected it is further refined during detailed design, where the goal is to provide a complete description of the building architecture. This complete description consists of blueprints and other working documents that are meant to be used for the construction of the building.

During the design process, the architect considers several priorities that are also of concern and responsibility for the engineers involved in the design of the building, for example: providing good air circulation within the building, maintaining comfortable room temperatures, ensuring occupants' safety against fire and, of course, maintaining the building shape and stability. The engineering systems (mechanical, electrical, plumbing, structural) are designed to perform those functions. Nevertheless, being the leader of the design team, the architect plays a central role in the design of engineering systems.

2.2.1.2 The role of the architect in conceptual structural design

Structural planning and conceptual structural design are often initially performed by the architect (Schaeffer 1998). As confirmed through interviews with architects and engineers, architects often place columns and in some cases define structural walls. This

task is only partially done by architects however since they rarely place beams, girders or braces for example. Nevertheless, there is a consensus among architects that the structural system is always present in the architect's thoughts from the very beginning of the architectural design process, which is not surprising since the structural system is an inherent part of the building architecture (Shaeffer 1998). Knowledge about the structural system therefore is one of the basic responsibilities of the architect (Schmitt 1988). However, most architects lack in-depth education and experience in structural design that would allow them to visualize a structural system as a whole. Architects tend to place structural elements individually, but rarely as part of structural subsystems or assemblies. Some architects may even place structural elements in space following regular patterns that adequately match the architectural functional patterns but with little regard to how the proposed partial structural system layouts may resist the loads applied to the building. Therefore, partial structural system layouts that are defined by the architect are always assessed later by the structural engineer.

In actual design practice, engineers are given architectural blueprints, which are sets of floors plans, sections and elevations. Before performing structural framing layout, the engineer carefully inspects the blueprints. While searching for direct load paths to the ground the engineer superimposes typical floors plans and compares them with sections and elevations. A three-dimensional (3D) model of the architectural design, that could be a mock-up or a digital computer model, is sometimes given to the engineer. This model, when available, facilitates the engineer's inspection of the architecture from different angles.

The architect's design intent is implicitly contained in the blueprints and models given to the engineer and translates into requirements and constraints for the structural engineer. These constraints are likely to restrict the layout of the structural system either globally or locally. Thus, the structural engineer must be aware of the intent in the architectural design in order to avoid violating architect-imposed and client-imposed requirements and constraints (Meyer and Fenves 1993). However, while this intent is apparent for the architect it is not so obvious for the engineer. Therefore, whenever possible, the architect's design intent needs to be explicitly communicated to the engineer.

During personal interviews, practicing structural engineers have explained that interaction with the architect usually starts before blueprints are produced. Architects communicate with them via telephone conversations asking specific questions regarding, materials, as well as structurally acceptable floor spans and corresponding floor depths. Given that the architect is the initiator of the building design process, it is basically up to him/her to decide when the structural engineer should be involved in the building design process. Ideally, from the structural engineering standpoint, the structural engineer should start participating as early as possible during the building design process in order to be able to provide timely structural feedback to the architect. However, this is rarely the case since most architects often prefer to develop their ideas in isolation, using "private" sketches, before discussing them with the structural engineer (Meniru et al. 2002). The following sections describe how structural engineers use the information given by the architect in order to synthesize structural systems that conform to the architect's design intent.

2.2.2 Overview of the structural design process

The design of building structures can be defined in simple terms as the act of positioning and sizing structural elements in space and specifying connections among these elements. However, this process is more involved since it actually consists of interrelated sub-processes that are carried out over various stages of refinement of the building.

2.2.2.1 The engineering design process

Bédard and Gowri (1990) describe three sub-processes of the engineering design process (Figure 2.4), which are presented hereafter in the context of the design of building structures.

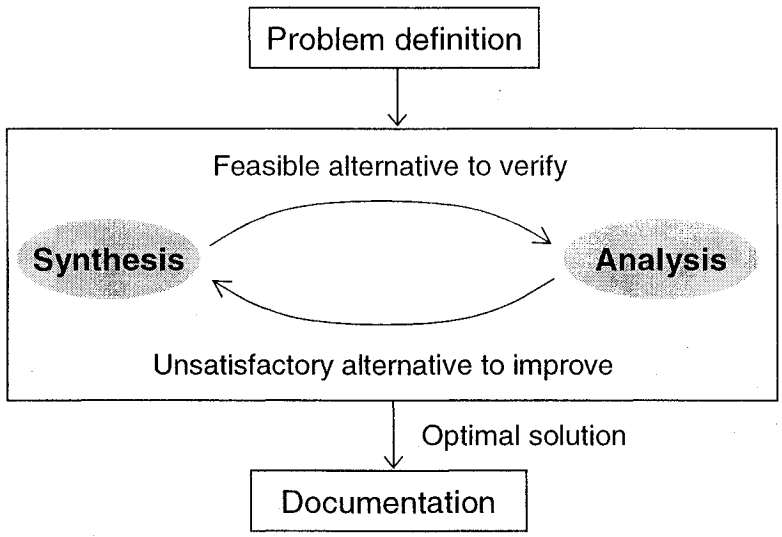


Figure 2.4 The engineering design process (Bédard and Gowri 1990)

a) Problem definition

Problem definition refers to the identification of a given need and the specification of the structural system being designed taking into account objectives, requirements and

constraints, as well as the standard structural design criteria. These become the parameters against which the design product will be evaluated. The criteria that govern the design of the structural system are: safety, serviceability and economy. Structural safety is achieved through stability, strength, and stiffness factors. Structural stability is the need to maintain the shape. It is provided by ensuring that the building is properly constrained by its supports and in equilibrium with applied vertical and horizontal loads. Strength and stiffness provide the actual resistance of the structure to imposed forces giving rise to deformations that must be within ultimate and serviceability limits to be acceptable. With these criteria in mind, the engineer seeks to design a structural system that will perform its function, usually having minimum weight and cost.

In addition to structural design criteria, the design of the structural system must take into consideration wider goals and concerns for the building that come from the client, the architect, and the building services engineers. Additional requirements come from local external factors that cannot be controlled such as environmental factors, cost and availability of materials at the building location, local building regulations, and the shape of the land. Other variables to be considered, which are related to the building itself, are the client's budget, the building type, the approximate number of stories, the approximate construction area, etc. Even the fabricator (in the case of steel structures) and the contractor are likely to impose constructability concerns.

The requirements and concerns coming from different relevant sources define the "overall building design criteria" that allow the building to be designed as a whole. The structural engineer therefore looks for structural efficiency (i.e. safety, satisfactory performance, and economy) in the light of the overall building design criteria. The final

goal in structural design is therefore to achieve the best overall building design rather than the most economical structural design (Meyer and Fenves 1993).

b) Synthesis-Analysis loop

The synthesis-analysis loop constitutes an iterative sub-process that drives the entire structural design process. The engineer first synthesizes alternative structural systems that take into consideration the overall building design criteria. Each promising alternative is then analyzed to determine its expected behavior. It is finally evaluated to determine whether it meets the structural design criteria, as well as the overall building design criteria. If some requirements are not met, the engineer goes back to the synthesis stage and either creates a new alternative or makes changes to the alternative under consideration, then perform analysis and evaluation again. The loop continues until modifications to the alternative make the current structural system satisfactory with respect to the given building design criteria.

c) Documentation

Documentation is concerned with the production of drawings and written specifications that completely describe the object being designed and are therefore sufficient for its subsequent fabrication and erection (for steel structures) or construction (for concrete structures).

2.2.2.2 Stages of structural design

From the perspective of the level of refinement and definition of the design artifact, three different stages have been identified for the engineering design process, namely:

conceptual design, preliminary design and detailed design. The three stages of structural design are briefly described as follows, with emphasis on conceptual design.

a) Conceptual design

The goal of the engineer during conceptual structural design is to devise and lay out feasible structural systems that transfer loads to the ground efficiently while matching the patterns of the building architecture and responding to the design intent of both the architect and the engineer. A feasible structural system layout provides continuous load paths to the ground in a way that makes it structurally stable and realizable as a whole. The structural elements that make up the structural form are arranged, sized and connected in such a way that acting together contribute to the intended functions of the structural assemblies and sub-systems where they belong (i.e. transferring gravity and lateral loads to the ground). Thus, the emphasis during conceptual structural design is on the synthesis of structural solutions (i.e. form and function). Consequently, during conceptual design the engineer makes decisions regarding the form of the structure by reasoning about its geometry and topology, while focusing on the functional requirement of load transfer. These decisions are based mostly on knowledge about the behaviors and experience on the applicability of available construction technologies (of subsystems, assemblies, elements and connections) and materials for a given design situation. Depending on the size and complexity of the building, and the experience of the engineer, simplified structural analysis may also be required during conceptual design. The outcome of the conceptual stage of structural design is an initial description of the structural system in terms of the layout of its members and their tentative cross-sectional properties, their connectivity, and materials.

b) Preliminary design

During the preliminary design stage a few promising design alternatives are roughly analyzed and evaluated against design criteria. If an alternative does not meet the criteria, then the design is modified, reanalyzed and reevaluated. Therefore, the preliminary design stage can be seen as a refinement stage through simplified analysis. The outcome of the preliminary design stage is identification of the most promising structural alternative, under the overall building design criteria.

c) Detailed design

During detailed design a comprehensive mathematical model of the "best" and hopefully final structural alternative, as derived from the preliminary design stage, is built in a computer. This mathematical model is used for performing thorough analysis of the structure in order to determine its behavior under different loading conditions. This usually leads to optimized cross-sectional properties of structural members. In this way, through analysis-design cycles, the level of detail in the description of the structural system becomes more complete and precise. At this stage, structural connections among elements are also specified in detail. The outcome of the detailed design is therefore a complete set of documents (including fabrication and construction drawings, specifications, schedules, etc.) that precisely describe the structural system.

In actual design practice there is no sharp distinction between these three design stages. For example according to a practicing structural engineer, during conceptual design complex structural analysis (i.e. dynamic analysis) may be performed in order to verify the stability of complex structural forms under lateral loads. During preliminary design,

optimization of each promising structural solution can also take place whenever a more accurate comparison among alternatives is required. Changes may take place at any moment during the design process and any design alternative may be discarded or modified whenever it is considered necessary. Nevertheless, it is expected that major, strategic modifications take place earlier on during the design process, while only minor adjustments take place at the detailed stage.

2.2.2.3 Structural synthesis by problem decomposition

Structural synthesis by problem decomposition, also called top-down hierarchical refinement, is an approach in which a complex synthesis problem is broken-down into simpler sub-problems. The need for problem decomposition in structural design gave rise to the structural system hierarchy as described in section 2.1. Several researchers have proposed process models that apply to conceptual structural design (Sauce et al. 1992, Sacks and Warszawski 1997, Scherer and Gehre 2000, and Rivard and Fenves 2000a). These models share several characteristics, for example all of them follow a top-down, problem decomposition, approach. However, only few of these models attempt to link decisions made on architectural space functionality to decisions concerning subsystems, assemblies and elements (Sacks and Warszawski 1997, and Rivard and Fenves 2000).

Problem decomposition is the approach for structural synthesis that is adopted in this research project. Lin and Stotesbury (1988) use synthesis by problem decomposition to develop a so-called total-system approach, which is essentially a top-down approach (i.e. in which a problem is solved starting from the most abstract concepts and ending with its most specific entities). In this research project the terms "synthesis by problem

decomposition", "hierarchical refinement" and "top-down approach" are used interchangeably. Rivard and Fenves (2000a) proposed a design model for the conceptual design of building structures, which is an extension of the total-system approach and makes use of the concepts explained in section 2.1. In this model, the structural engineer first breaks down the structural system into independent structural volumes that are assumed to behave as structural wholes. Independent structural volumes are in turn subdivided into smaller sub-volumes called structural zones. Structural zones are introduced in order to allow the definition of structural requirements that correspond to architectural functions. Independent structural volumes are also decomposed into three structural subsystems, namely the foundation, the gravity, and the lateral subsystems. Each of these subsystems is further refined into structural assemblies. Finally, structural assemblies are decomposed into structural elements and their connections.

Schodek (2003) however argues that design is not a deterministic process that moves from abstract to detail. He claims that the process is interactive and iterative since good designers typically deal with each concept in the structural hierarchy at varying levels of engagement at all times. Referring to engineering design in general, Cross (1989) states that the designer moves freely between different levels of detail and there is frequent back-tracking up and down the levels of hierarchy of the design product. The reason for this is that a decision at any particular level in the hierarchy may turn out to be sub-optimal in the light of subsequent options available at the lower hierarchical levels. Nevertheless, a top-down approach with backtracking would allow engineers to move back and forth between different levels of detail as described by Cross (1989).

In addition, the top-down approach has the advantage of enabling the engineer to model initially purely functional entities, such as structural massings, sub-systems and assemblies, that will eventually be embodied into arrangements of structural elements, whose functions contribute to those of the purely functional entities to which they belong. These entities are particularly important during early design stages because they let designers express their functional concerns, while facilitating the subsequent structural configurations (Rosenman and Gero 1996). For example, they allow overall structural concepts to become contexts for thinking about local issues of detail component interactions and ensures compatibility between overall concepts and their constituent components. They also allow relating structural concepts at different levels of the structural hierarchy to architectural concepts, which results in engineering feedback to the architect at each hierarchical level. Sacks et al. (2004) extends the top-down approach to building design in general. They argue that building design is essentially a top-down process, particularly at the conceptual design stage. This is unlike mechanical design which is oriented towards detailing complex parts and modeling their interactions.

The top-down approach also facilitates bringing analysis earlier into the design process because it allows higher-level (i.e. purely functional) entities to be analyzed before being populated with lower-level entities. Such analyses initially assume overall structural integrity of the high-level entities, which is substantiated later when the constituent low-level entities are configured (Lin and Stotesbury 1988). Bringing analysis earlier-on in the design process enables incorporating functional concerns earlier on in the design process, as well as more informed decision-making. In addition, it expedites early design iterations because the structural system descriptions are devoid of unnecessary details.

Leaving analysis for later stages may result in over-design and increases the chances of having major changes to the structural form later on in the design process (Fenves et al. 2004).

2.2.3 Integration issues

While synthesizing structural schemes, the engineer takes advantage of the organizing principles that give shape to the building architecture. Like architects, engineers strive for balance, order and unity of the structural system as part of the building as a whole, while keeping in mind other equally important principles that govern the synthesis of structural systems such as continuity, slenderness and lightness. Each level of the structural hierarchy brings to light potential structural problems and offers opportunities for the integration of the structural system to the building architecture. These problems and opportunities are described in the remainder of this section.

2.2.3.1 Building forms and independent structural volumes

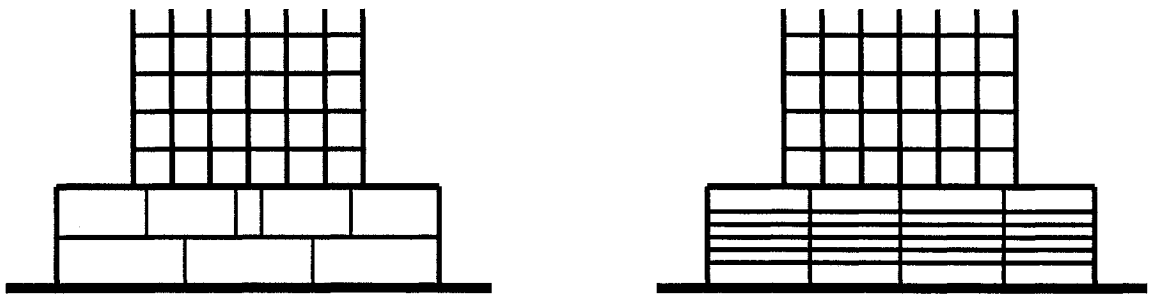
Engineers use independent structural volumes to analyze the structural properties of the building form. Depending on the irregularities of the building form and following scaling, sizing and proportioning rules the structural engineer separates the building form into independent structural volumes. In doing so, the engineer considers several characteristics as determinants of its performance, namely: size, proportions, balance, symmetry in plan and elevation, and in general the regularity of the form (Arnold and Reitherman 1982). For example, setbacks and cantilevers are form irregularities that may cause abrupt changes in strength and stiffness, as well as overall unbalance, on the

building. Lin and Stotesbury (1988) use the term total structural system to describe the overall building form in structural engineering terms. They present simplified analysis that can be applied to the building form (the total structural system) in order to verify the structural stability of the form. For example, they use the building aspect ratio (the height over the depth as measured in the direction of the load action) to determine the overturning resistance of buildings when subjected to horizontal forces. In the horizontal plan the building proportion is measured using the plan aspect ratio (relation between depth and width of the form). Symmetry is a convenient feature in structural design; this applies to the overall building form as well as the local building element layout. For example, a poorly proportioned floor plan (i.e. having a big plan aspect ratio) combined with asymmetrically placed lateral load resisting elements is likely to develop excessive deformations in the building that affect its serviceability and possibly its safety.

2.2.3.2 Space composition and structural zones

There is a natural interdependency between the overall organization of the structural system and the functional organization of spaces, i.e. space composition (Holgate 1986), which involves the functionality of spaces and their layout. Just by knowing the functionality of the spaces and the geographic location of a building, experienced engineers are able to define feasible type(s) of subsystems and assemblies that are likely to satisfy the functional requirements of the spaces, as well as load patterns expected and tentative floor spans among others. Structural zones are used in structural design to describe a functional space or a group of spaces that have similar structural characteristics.

Buildings may have only one or many structural zones, depending on the variations in programmatic requirements (cf. section 2.2.1.1 a) and the effect of this variation on the structural system, as shown in Figure 2.5. The relative location of structural zones is often a source of conflict between architects and engineers since architects may prefer to place large open spaces at lower levels, requiring a complete break of the structural system from closely spaced columns or bearing walls above to a few tall and widely spaced columns below (Holgate 1986). Each structural zone is likely to be designed differently (Lin and Stotesbury 1988) however a source of complexity comes from the need for achieving structural continuity and compatibility at the interface between structural zones.



Banquet and conference rooms at lower levels

Parking structure at lower levels

Figure 2.5 Variations in building programmatic requirements
(Arnold and Reitherman 1982)

Another important aspect of space functionality that relates to structural design is the layout of equipment within a space (cf. section 2.2.1.1 b) since it defines how the space is internally organized to fulfill its function. Personal interviews with structural engineers have shown that equipment layout within spaces, when available, is used by the engineer for inferring functional design intent from the building architecture. For example, in an

auditorium the layout of chairs and the position of the podium are used as references by the engineer for placing columns without obstructing the view for the audience.

2.2.3.3 Integration of structural subsystems and assemblies

The type of building being designed gives a first indication on the structural subsystems and assemblies that can be suitably used for transferring loads (Pillai et al. 1999). According to their type, buildings are characterized by having typical patterns of space establishing elements (cf. section 2.2.1) that define locations for structural supports and determine the types of subsystems and assemblies that are applicable to each type of building. On the one hand, apartment buildings are characterized by having permanent walls located regularly and defining small floor spans (e.g. from 3 to 7.5 m). Therefore, it is likely that these permanent walls are used for lateral and possibly gravity support. In this case, the vertical lateral subsystem will likely consist of shear walls that can also be used to carry gravity loads. The horizontal subsystem will likely consist of flat plate assemblies that can be used directly as ceilings for the floor level below. On the other hand, office buildings are characterized by having service cores and flexible spaces. The distance between the building façade and its cores is generally no greater than 12 m, because it represents an acceptable maximum distance from a natural light source. For clear spans, structural economy also tends to limit this distance. In order to span large distances, the horizontal subsystem will likely consist of secondary beams spanning over primary beams. Lateral support may be provided by rigid frames or braced frames. As another example of the relation between the type of the building and the subsystems and assemblies that are likely used, one storey factories and warehouses are characterized by

having large open spaces demanding long-spanning structures which are usually made out of steel. Lateral support is usually provided by bracings, gravity support by columns and long-spanning roofs are supported by large steel trusses.

Several types of subsystems and assemblies can be applied for each type of building. The evaluation of the most suitable one is based on architectural value (i.e. how they respect the architectural requirements) and structural efficiency (cf. 2.2.2.1 a). For example, the structure self-weight is also a key factor when selecting structural subsystem and assemblies. For a reinforced concrete building a one level flat slab is lighter than a three-level floor assembly with secondary beams and primary beams. The choice of the type of floor assembly depends, among other factors, on the span given by the structural bay dimensions, which in turn depend on the building architecture. Each floor assembly type usually has a span range of applicability. Within this range, longer spans will inevitably require heavier spanning members. Since the self-weight of the floor assembly accounts for as much as 50% of the total load that must be carried by the structure, a fundamental conceptual design decision must be made concerning bay dimensions and type of floor assembly to be used (Luth et al. 1991).

Analysis can also be performed for subsystems and assemblies as required for evaluating structural alternatives and providing feedback to the architect. For example, when analyzing a lateral-load resisting subsystem consisting of a group of wall stack assemblies distributed over the building plan, the engineer considers floor assemblies as rigid (non deformable) diaphragms that transmit lateral loads equally to the walls. Floor assemblies are therefore modeled as rigid entities that contribute to the overall performance of the lateral-load resisting subsystem. Later on, when the engineer actually

designs each floor assembly, beams and slab elements are configured, sized and connected together in such a way that the assembly performs its intended function as a rigid diaphragm.

2.2.3.4 Space establishing elements and structural elements

Before laying out structural assemblies and elements, structural engineers check vertical continuity of architectural SEEs to verify whether they can be used to transfer loads down to the ground (Meyer 1995). Considering the walls from the building architecture, the engineer selects the walls (i.e. considering factors such as wall orientation, location and extension) that can become part of the lateral and possibly gravity subsystem and assigns them a structural function so that they make up wall stack assemblies. For example, the walls that define the building core(s) (that contain the elevators, stairwells and service shafts) are ideal for becoming structural since they usually run from the basement to the upper floors (Pillai et al. 1999). Building envelope walls are also good candidates for performing a structural function because they make up continuous vertical planes around the perimeter of the building. However, modern building envelopes, usually with large setbacks and cantilevers, break the structural continuity and make the structural integration with the building envelope more difficult. From the structural engineer's standpoint, symmetric locations of structurally relevant SEEs in the floor plan are always preferred. As for the columns placed by the architect, the structural engineer assesses if those columns could properly integrate within frame assemblies.

The task of selecting architectural elements that could perform a structural function is simplified by the fact that spaces and related SEEs are usually organized into repetitive

patterns which derived from the architectural functional dimensions and the building grids (cf. section 2.2.1). This suggests that the structural system should use the same grids as reference base for laying out the structural system, which is what usually happens since in actual design practice the patterns formed by the structural system layout are intimately related to those of the architectural functional organization; each affects the other (Schodek 2003).

In addition to the potential support paths defined by SEEs, architects place constraints that limit the positioning of structural members within a space and the dimensions of structural members. For example, large open spaces require widely spaced columns and sometimes they are required to be column-free. Considering the structural elements of the horizontal subsystem, usually the location of the vertical supports, as well as the type of the vertical support subsystem suggests a most suitable arrangement for the elements of the horizontal subsystem. However, neither one dictates the design of the other, but they are interactively designed together (Schodek 2003). In addition to the free-span between vertical supports, one of the factors that influence the definition of floor assemblies is its overall depth. This depth has to be in agreement with the clear floor to ceiling height as specified by the architect while giving room for mechanical ducts and pipes.

As explained by Arnold and Reitherman (1982), another important aspect regarding space establishing elements and the structural system relates to the effects of the structural system behavior in the space establishing elements that are not intended to perform a structural function and vice-versa (i.e. the effects of non-structural space establishing elements in the structural system behavior). Ideally, the architect should ask for the engineer's advice when positioning and selecting non-structural partition walls

and façade elements with deformation capacities compatible with the designed structure (Bachmann 2003). Even though, this aspect is relevant to conceptual design it is not considered further in this research project since it relates to particular construction technologies and materials. The following subsection explores the impact that the patterns and regularity in the organization of physical elements has on the integration between the architecture and the structure. Next a parameter called the degree-of-fit that attempts to qualify this integration is presented.

a) Patterns and regularity

Engineers seek to configure structural systems in repetitive geometric patterns which are invariably dependent on the architectural programmatic requirements and corresponding functional dimensions of spaces (cf. section 2.2.1.1). In the context of the architecture and focusing on structural efficiency and performance, engineers seek to configure structural systems that are elegant and aesthetically pleasant while following constructability requirements and general synthesis principles, mainly: simplicity, continuity, symmetry and repetition. The term regularity is used often to refer to all the above principles (Arnold and Reitherman 1982). Irregular buildings at the SEE level are characterized by having structurally relevant SEEs with dissimilar geometric shapes and located unevenly at interrupted vertical and horizontal patterns. These buildings pose great structural problems since unwanted and difficult-to-predict behaviors are likely produced that require expensive and inefficient structural solutions (Arnold and Reitherman 1982).

Building can be categorized as single-cell or multiple-cell depending on the patterns generated by spaces and space establishing elements (Schodek 2003). Single-cell buildings are characterized by having only one or few mostly large spaces (e.g. factories and retail stores), and multiple-cell buildings are characterized by having smaller and repetitive spaces (e.g. apartments and hotels). Some buildings may be both single-cell and multiple-cell by having a generalized pattern of small repetitive spaces that is interrupted by spaces or groups of spaces that are different in scale from each other (e.g. an auditorium and a gym in an educational building). This is illustrated in Figure 2.5.

In addition to the varying architectural programmatic requirements (cf. section 2.2.3.2) a variety of factors affect the patterns for good integration between the building architecture and the structural system. For example, some types of buildings pose greater structural challenges than others. Tall buildings and symbolic buildings such as airport terminals and museums often demand innovative structural solutions. Also, the larger the spans between vertical supports, the lesser structural alternatives are available (Schodek 2003). For example, to span large distances high-efficiency structures are required, such as space trusses. By contrast, if spans are shorter and plenty of support is available, then structural possibilities are plentiful.

Considering training and personal factors, architects and engineers have different design priorities and functional concerns. Therefore it is unreasonable to expect a perfect match between the architectural functional organization of spaces and the structural system layout. This is caused in part by the tighter rules applied to the structural system layout as opposed to the more flexible design rules followed by architects. A consequence of this is that synthesis principles do not mean exactly the same for architects and engineers. In

structural engineering, concepts like scale, balance, order and repetition are given a more strict meaning than in architecture. For example, the principle of repetition usually means in structural engineering equally sized structural elements repeated over and over. For architects this is regular repetition. Architects also consider irregular repetition since it also produces pleasant aesthetic effects (Winters 1997). Irregular repetition produces irregular structural bays. Therefore, the presence of irregular repetition in architectural design is likely to become a source of conflict between architects and engineers since a key objective of conceptual structural design is to maximize regular repetition (Luth et al. 1991). The principle of balance is another example where different meanings are often given by architects and engineers; balance in engineering comes with symmetry, for architects this is formal balance. Architects often design informally balanced buildings that are asymmetrical (Winters 1997).

Another important concept in engineering synthesis that relates to regularity and repetition is the concept of grouping. Engineers group structural entities based on similarities (e.g. function, material, geometry and position). Grouping structural elements leads to more efficient structures. The benefits of grouping are better design organization, uniformity, reduced design, fabrication and construction time, and ultimately cost savings. Grouping actually simplifies the synthesis process, since it allows the engineer to deal with sets of entities with similar characteristics instead of dealing with the individual entities. To account for the common patterns in architecture and structural engineering Schodek (2003) defines a term called degree-of-fit that is described in the next subsection.

b) Degree-of-fit (DoF) between architecture and structure

The term degree-of-fit (DoF) is used by Schodek (2003) to describe the type of match between so-called "critical functional dimensions" and the pattern formed by the vertical support system. These critical functional dimensions determine in the floor plan basic functional modules, which essentially correspond to the architectural grids. Schodek describes two possible fits: (1) a one-on-one fit where the structural bay dimensions correspond directly to the functional dimensions, and (2) a loose fit where the structural system may span over multiples of the functional dimensions. In simple terms, the task of the engineer is therefore to find the best structural system possible that matches the architectural grids, following structural considerations. Thus, depending on the structural technologies applicable according mostly to the available floor spans, the structural grid should map or be a subset of the architectural grid, thus producing a single global grid for the building. However, those DoF are not sufficient to describe the actual degree of integration between the architectural design and the structural system layout; the following degrees-of-fit are considered more convenient:

1. One-on-one (or map) fit: as per Schodek;
2. Tight (or subset) fit: equivalent to the "loose" fit as per Schodek;
3. Loose fit: is a tight fit globally but with mostly local misfits;
4. Misfit: there is little or no match between patterns.

DoF number 1 and 2 can both be considered tight fits or good fits since in both cases the structural system layout adequately matches the architectural patterns. DoF number 3 (loose fit) is considered to represent most buildings that exhibit a generalized uniform organizational pattern but either the architect or the engineer configures elements locally

outside the pattern, thus producing local "misfits". This is a normal situation since the organizational patterns are supposed to act as reference base rather than a straight jacket that impair design creativity. For local misfits a special local framing is usually required to integrate the structural elements that do not fit with the rest of the structural system (Schodek 2003). DoF number 4 (i.e. misfit), is expected to happen in rare situations and could be a sign of a poorly integrated design.

The fact that Schodek defines DoF based on critical architectural functional dimensions, which correspond to architectural grids, instead of basing it on the actual patterns of space establishing elements (SEEs) makes the DoF a poor indicator for the integration between the structural system and the building architecture. Architectural grids are used by architects as a reference-base for organizing spaces and placing SEEs. However, there is no one-on-one mapping between grid lines and their intersection points, and vertical SEEs. Therefore, the DoF indicator should be based on the actual patterns of SEEs as placed by the architect rather than architectural modular grids. This "modified" DoF can then be used to determine the magnitude of the spanning dimension and the load transferring direction for the horizontal subsystem. Figure 2.6, taken from Schodek (2003), shows a hypothetical building plan to illustrate how patterns based on modular functional dimensions are a good reference but are not sufficient to convey the intent of the architect. Figure 2.6 a, shows a pattern formed by a basic functional module, which is used by Schodek as a basis for defining the DoF. Figures 2.6 b, c and d, show patterns formed by walls (i.e. vertical SEEs) which are subsets of the first pattern (shown in Figure 2.6 a). In Figure 2.6 c, one of the spaces is intended to be column-free. If the engineer would like to place a line of columns crossing the space as indicated by the

discontinuous line, it becomes a one-on-one fit based on Schodek's definition. However, it shows poor integration. By contrast, using the patterns defined by the vertical SEEs takes into consideration the architect's design intent since vertical SEEs define actual spaces and not space modules. Furthermore, wall patterns are not always subsets of the pattern formed by the critical functional dimensions.

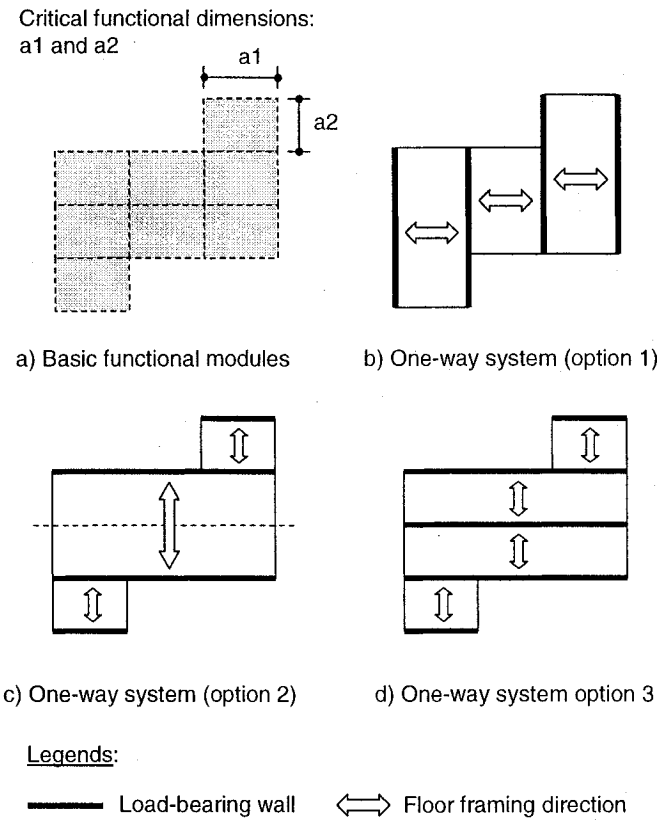


Figure 2.6 Patterns formed by critical functional dimensions versus patterns formed by vertical SEEs (adapted from Schodek 2003)

Therefore the actual patterns formed by vertical SEEs should be used as reference base for configuring the structural system. In practice, both the architectural grids and the vertical SEEs are used by engineers as reference bases for configuring the structural system. For example, as seen from Figure 2.6, the critical functional dimensions can be used by the engineer at the outset for having an idea of the minimum clear spans.

However, as indicated by the doubled-headed arrows in Figure 2.6, the load transfer direction for floor assemblies is given by the availability of vertical supports (walls and columns). Both references should be respected by the engineer. However, priority should be given to the patterns formed by vertical SEEs. Chapter 4 proposes a model that considers both architectural functional and physical concerns when synthesizing structural solutions. The following subsections are examples of DoF for actual buildings.

c) Examples of degree-of-fit

Example 1

Apartment buildings have permanent walls to separate apartments, which usually become structural. Figures 2.7 through 2.10 illustrate this case. The result is a tight structural fit. However, Figure 2.9 shows that, as expected, local misfits take place. Therefore, overall this case could be classified as a loose fit (i.e. a tight global fit with local misfits).

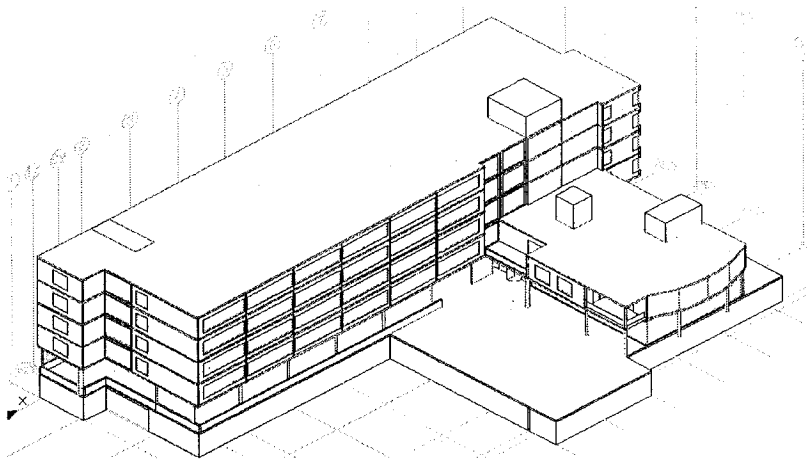


Figure 2.7 3D model of an apartment building

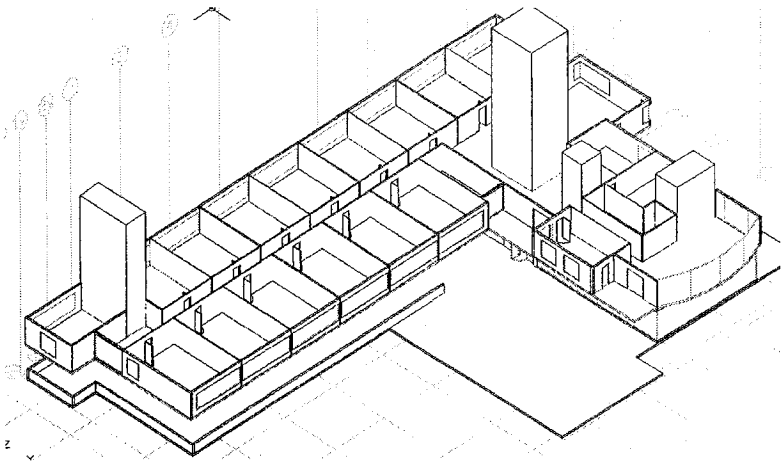


Figure 2.8 Patterns formed by SEEs

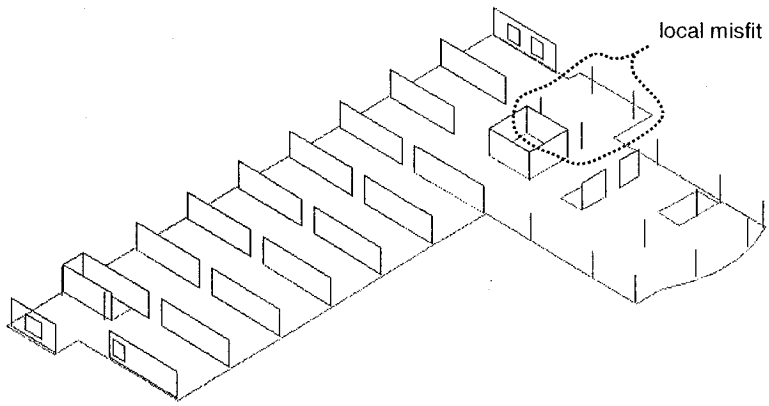


Figure 2.9 Walls and columns defined by the architect are structurally relevant

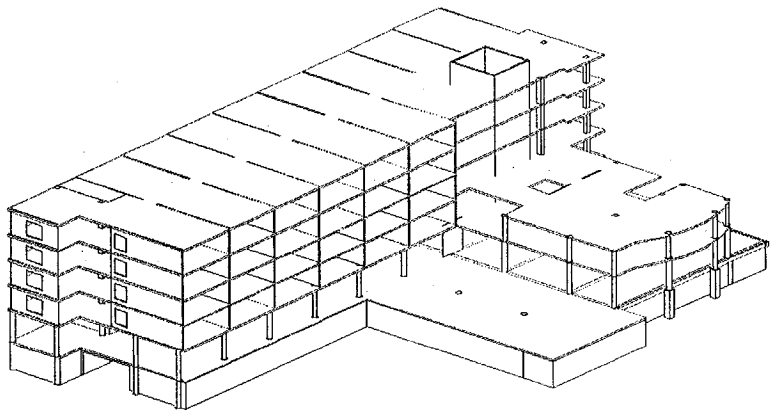


Figure 2.10 The structural system as defined from the architecture

Example 2

Office buildings usually have flexible open spaces where cubicles are organized using removable partitions. Space flexibility means that partitions can be moved whenever required and produce different cubicle organizations within the floor. In an ideal situation there would be no columns in open spaces, thus imposing no limitation on the layout of working cubicles. However, structural systems and materials have limited capabilities that impose limits on floor spans. Therefore, columns are sometimes inevitably required within open spaces. This is illustrated in Figures 2.11 and 2.12. Figure 2.11 shows a typical floor plan of an office building with considerable misfits among the structural patterns and the partitions. However, once non-permanent partitions are removed the structural pattern becomes apparent (Figure 2.12). In Figure 2.12 only permanent walls (core walls) are left from the architecture for performing structural functions. In general, the building presents a regular global pattern. However, due to the geometry of the building, local misfits are found with core walls.

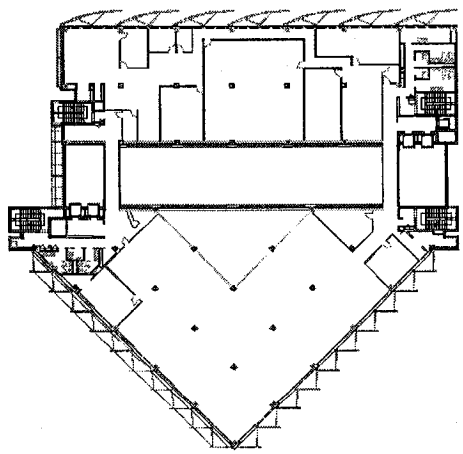


Figure 2.11 Floor plan of an office building (from Autodesk®)

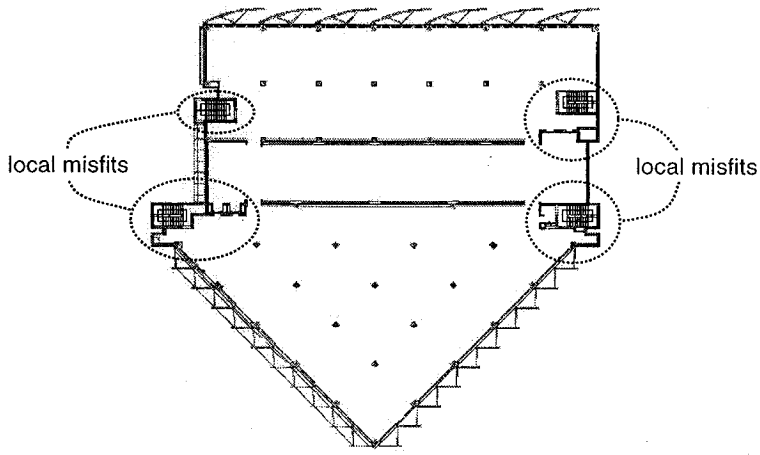


Figure 2.12 Floor plan after removing non permanent partitions

2.2.4 Summary

The purpose of this section has been to outline the main factors that should be taken into consideration by an application to assist conceptual structural design. Conceptual structural design is a process that involves visualizing patterns and synthesizing entities at various levels of detail. This process is made simpler through problem decomposition that allows a complex synthesis problem to be decomposed into simpler sub-problems.

The processes of conceptual design of building structures and the architectural design process are interdependent. This interdependence may become a curse or a blessing for both parties depending mostly on the degree of awareness that each party has of the design priorities and concerns of the other party. Architects and engineers think differently, however they also have much knowledge in common that become relevant during conceptual design. Therefore, the main challenge of developing a computer application that aims at supporting conceptual design of building structures is being able to make use of the common knowledge between both parties, while acknowledging individual priorities and concerns.

2.3 Overview of commercial architectural CAD and structural engineering software

This section presents new trends in architectural design and structural engineering software. On the one hand, it describes first the features of architectural design software that have an impact on the conceptual design of building structures. On the other hand, it emphasizes structural engineering applications that attempt to support the conceptual stage of structural design. The Appendix A presents a list of all software packages mentioned in this section.

2.3.1 Commercial architectural CAD software

Architectural CAD (computer aided design/drafting) software has evolved from purely 2D drafting software, in which the aim was generating drawings that describe buildings, to "digital building" or "virtual building" modeling software that models buildings in three-dimensions (drawings are generated automatically from the 3D model). Modern CAD packages share many important characteristics that reflect the common concerns of CAD developers towards better support for the entire building life-cycle. In this overview the following main characteristics were identified of the latest architectural CAD packages that could facilitate the support by the structural engineer: (1) there is a trend towards incorporating the digital or "virtual" building paradigm for a unique and consistent digital representation of the building throughout its life-cycle; (2) efforts are directed towards sharing and exchanging data; (3) CAD software offer improved parametric capabilities for facilitating design changes; (4) there is still poor support for abstraction/decomposition of building systems; and (5) there is also poor support for the

early stages of design. These characteristics are described below in this section, thus showing new trends and developments in architectural CAD may lead to novel paths for integration with structural engineering software during conceptual design. The applications mentioned in this sub-section are listed in section A.1 of the Appendix A.

2.3.1.1 Incorporate the digital building model paradigm

A digital building model (DBM), also called virtual building, describes the entities (e.g. walls and spaces) that make up a building, and how these entities are related to one another (e.g. a wall encloses a space). Modern architectural CAD packages enable architects to create a digital model of the building in three-dimensions by manipulating building entities such as walls and spaces instead of using points, lines, planes and volumes. Thus, these packages have an underlying representation (cf. section 1.5) of the building, also called building data model, specifying how building entities should be described (via attributes) and related to one another (via relationships) to guarantee the semantic validity of the digital building model at all times.

The evolution of architectural CAD programs from drawing oriented to DBM oriented has been made possible due to the advent of geometric modeling kernels (GMKs). Since all computer programs deal with some kind of data, they must have some kind of underlying data model (or data representation). GMKs rely on geometric-based data models and algorithms to handle mathematics, geometry (e.g. points, planes and volumes), and topology (e.g. the connectivity among geometric entities), which are hidden from the designer. Thus, they enable the derivation, directly from a model, of geometric information such as distances, lengths, areas, and volumes, as well as topologic

information such as adjacencies and intersections among geometric entities. For more information about GMKs please refer to the Appendix B.

Traditional general-purpose CAD programs, such as AutoCAD and MicroStation, deal primarily with geometric entities (e.g. points, lines, planes), so any building drawings created with these programs provide no information about the building itself. General-purpose 3D-modeling software products, such as Autodesk VIZ and form•Z, also have geometry-based data models, so even the 3D building models created with them fall into the same category and are used purely for visualization. Architecture-specific CAD applications such as Autodesk Architectural Desktop (ADT) and Architecture for Microstation Triforma do have an underlying building data model. However, since these tools are add-ons to AutoCAD and MicroStation their building data model is subjugated to the main geometric data model. As a consequence, they display limited capabilities for creating and editing building models and extracting building-related information. For example, ADT does not allow modeling non-tangible building entities such as spaces. More advanced architectural CAD applications such as Autodesk Revit and ArchiCAD by Graphisoft have been developed to model buildings in 3D right from the start, so their underlying data model deals primarily with building entities. Geometry is only one of the properties of these building entities. Nevertheless, these advanced applications still rely on a GMK to handle geometry and topology. However, the emphasis on building entities rather than geometry gives these applications most of the advantages of a truly DBM, such as the following: (1) since the data model of these programs focuses on the building objects and their relationships to each other, it is rich in information about the building itself, and the data can be extracted and used for various purposes, be they

documentation, visualization, or analysis (e.g. they enable the manipulation of non-tangible building entities such as spaces); and (2) a single building object is described only once in the data model, whether it appears in a plan, an elevation, or a 3D view. This leads to a model that is always consistent. And any change or update made will automatically be reflected in all the views.

The aim of software developers with the digital building model goes beyond architectural design. They want to be able to provide DBM-based support for the architecture, engineering, construction and facility management (AEC and FM) industries. However, defining a single representation or several interrelated domain-specific representations to describe a building consistently over its entire life-span is a challenging task. Considerable efforts are being made by software developers in this direction with still little impact in actual design, construction and facility management practices.

2.3.1.2 International efforts in sharing and exchanging data

To overcome the problem of having to devise a digital building model (DBM) that will be used by all disciplines throughout the building life-cycle, CAD software developers have supported the development of information exchange standards, most notably the Industry Foundation Classes (IFC) by the International Alliance for Interoperability (IAI 2004). IFC is a non-proprietary building representation providing a standard format for describing buildings that has been specifically developed as a means to exchange DBM-based data among DBM-based applications in the AEC and FM industries. Because the IFC is an open data exchange format that captures building information, it can be used by the commercial DBM-based applications to exchange data with each other. Therefore,

since it is not meant to work with one particular application, it not only provides descriptions for entities that are commonly used and shared between multiple building construction and facilities management applications (e.g. a wall), but it also provides the mechanisms for creating new entities. In addition, it does not provide pre-determined relationships between entities (e.g. a wall encloses a space) but only the mechanisms for generating such relationships. An application that wants to exchange data via IFC has to become "IFC-compliant," which means that it is capable of importing and exporting IFC files. Applications get the IFC-compliant tag by going through a certification process. The IFC model specification is posted publicly and accessible to anyone, so developers can work with it and build the necessary IFC import and export capabilities into their applications. What this essentially means is that the data has to be mapped between their internal representation and the IFC representation. While Architectural Desktop by Autodesk and ArchiCAD by Graphisoft are IFC compliant, Revit by Autodesk is not.

2.3.1.3 Attempts at incorporating parametric relations

Parametric relations among entities simplify model construction and maintain the integrity of the digital building model when changes take place. Some examples of parametric relations are as follows: wall A and wall B must be parallel, partition wall C must be in the middle of space S1, the minimum dimensions of bathroom B1 must be n1 in the X direction and n2 in the Y direction, roof R must always be connected to walls A and B. Thus, when a dimension is modified or an entity moved its related entities will be modified accordingly so that the parametric relationship is maintained. Many complex mathematical relations, not only distances, can be parameterized; for instance, the light

intensity and its location as a function of the dimensions of the room and the number of joists in the roof. Parametric relations are a way of capturing the intent of the architect because they reflect the architect's intentions that need to be preserved when changes in the DBM take place. However, since these relations are explicitly defined within the model, if they are not implemented properly they may overcrowd the model and become a source of pitfalls that may potentially deter the efficiency and reliability of the application. In addition, they interfere with the architect's workflow because they require him/her to explicitly define associations among entities. Most DBM-based architectural packages are not parametric except for Revit by Autodesk that offers good parametric capabilities.

2.3.1.4 Poor support for system abstraction and decomposition

Abstraction and decomposition are operations that allow designers to simplify the design problem by focusing only on the relevant design aspects at each stage of the design process. Abstraction uses *type-of* relationships for specialization of building concepts, for example, a curtain wall is a *type-of* envelope wall. Decomposition uses *part-of* relationships to specify what components are parts of a system. The advantages of abstraction and problem decomposition in design are two-fold: (1) allow design refinement, and (2) enable each designer to have his/her own view of the model. A good example of abstraction/decomposition in architectural design is demonstrated in the complexity of the building envelope that protects the building interior from the elements. The building envelope is an assemblage of components that are carefully placed in specific positions so that the envelope, as a whole, performs its function properly. Typical

components of the envelope are: exterior finish, thermal barrier, vapor barrier, air barrier, structural support and interior finish.

Considering design refinement through decomposition, at the conceptual level an envelope wall can be created by the architect, using tentative dimensions and properties. Windows and doors having also tentative dimensions and properties may be associated to the wall while floor slabs and roofs may be parametrically related to the wall. At this point, envelope components and details are omitted. In subsequent design stages, however, envelope components must be defined, laid out and connected together within the layout provided by already defined components. Edges, corners, protrusions and connections are also detailed. All these components are *part-of* the building envelope assemblage that are defined as the building envelope is refined during the design process.

Considering the designer's own view of the model through abstraction, each component of the building envelope can be classified as belonging to one or many building envelope views. Then, when the structural engineer is required to design the structural support for the envelope, abstraction mechanisms allow him/her to view or emphasize only the components classified as belonging to the structural view (i.e. a *type-of* view). Similarly, a thermal view (i.e. another *type-of* view) shows or emphasizes the envelope components that are required for specifying the thermal properties of the envelope. Thus, each view represents an abstraction of the building envelope.

Commercial CAD systems are still far from achieving the required levels of abstraction/decomposition for allowing refinement of design concepts and providing multiple views for all design participants.

2.3.1.5 Poor support for the early stages of design

Commercial architectural software applications are devoting more efforts in supporting the early stages of architectural design. However, support for conceptual design poses greater difficulties. Usually regarded as the most creative phase of architectural design, this part of the process has traditionally been performed by hand. Nowadays, conceptual design applications are still not being widely accepted by architects that prefer the traditional tools. One of the many reasons that designers still hold onto the traditional ways in which design was practiced is because they have control over all the tools. They can adjust a parallel bar or T-square, adjust the angles of an adjustable triangle, use an architectural scale to design and layout physical spaces, use an engineering scale for site design; and most importantly have the tools on their desks that are related to the job at hand (Brown and Charles 1995).

Nevertheless, innovative conceptual design tools are in the market that resemble traditional architectural design tools. Architectural Studio by Autodesk and SketchUp by @Last software support freehand sketching and massing studies (cf. section 2.2.1.1 b). They re-create the architect's physical desktop electronically by having digital versions of familiar architectural tools, including pencils, markers, cutting blades, erasers, and tracing paper, and combine multiple representations including sketches, 3D models and renderings, CAD drawings, photographs, and other non-graphical data. Using these applications the architect creates geometric 3D models from sketches, which can be directly transferred to Architectural Desktop to produce digital building models. The main drawbacks from these applications regarding freehand sketching are that they still require a good level precision and do not tolerate any ambiguities in the sketches.

2.3.2 Commercial Structural Engineering Software

Nowadays, structural engineering software has advanced to a point that support is available for the entire process of structural design and analysis up to manufacturing and erection for steel, and up to detailing for concrete construction. The emphasis of these software applications is more on the automation of activities and processes, rather than the integration with related disciplines. In addition to performing complex calculations, these software applications nearly automate most of the repetitive and tedious work that is required for structural analysis, such as calculating tributary loads for the gravity system, calculating and distributing wind and seismic loads for the lateral system, selecting the required individual load cases and combinations from a specified code and performing quantity take-offs.

According to the tasks it supports, structural engineering software can be categorized into four groups that attempt to cover the entire spectrum of structural engineering activities (adapted from Fenves et al. 2000): (1) structural model generation, (2) specialized packages for integrated design and analysis, (3) component design and analysis, (4) detailing and drafting. From the first group, two subgroups have been identified: (1a) software for structural model generation using the traditional template approach, and (1b) software for structural model generation using modern 3D modeling techniques. Each category is covered in the following subsections along with examples of commercial structural engineering applications. The applications mentioned in this sub-section are listed in sub-section A.2 of the Appendix A.

2.3.2.1 Structural model generation using the traditional approach

Specialized design and analysis packages usually include a module for model generation. Following the traditional approach, engineers use a graphical user interface (GUI) to generate a wire-frame 3D model of the structural system, either by building it element by element or by selecting from pre-defined templates of common structure types (see Figure 2.14). To facilitate model generation, graphic editing tools are provided (i.e. copy, move, modify, etc.) as well as grouping tools (i.e. properties can be assigned to individual members or by group).

For example, GT STRUDL by Georgia Institute of Technology includes a model wizard that provides the user with predefined templates of common structures, which are selected from a palette. The structure is defined by selecting one structure type from the palette, and specifying dimensional information, such as: number of bays, number of floors, dimension of typical bays, and height of the floor. Then the material and the initial cross-sectional details for each element are selected by the engineer from lists of sections. Using ETABS by Computer and Structures Inc., the engineer even has the choice of letting the computer select initial element cross sections automatically. The structure can then be modified for example by merging structural bays or floors. The software guarantees model validity by model error checking, e.g. warning the user when rigid-body instabilities may arise due to connectivity problems. Integration with the building architecture is done in two ways: (1) by using the same project grids, defined by the architect, as a reference base for generating the model of the structural system, and (2) by transferring the geometry of the structural model to CAD packages back and forth using neutral file formats, such as DXF, the de facto standard format.

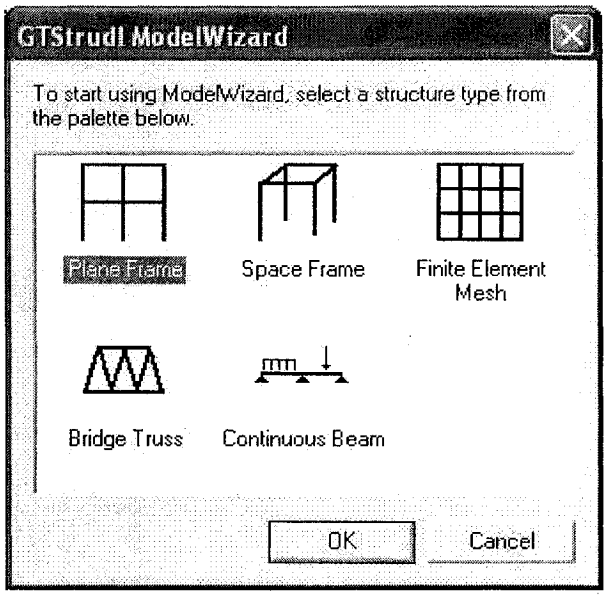


Figure 2.13 Templates of common structure types in GT-STRUDL

2.3.2.2 Structural model generation using the modeling approach

Personal interviews with practicing structural engineers have shown that, for simplicity, engineers usually prefer to model and analyze building structures in two-dimensions, thus avoiding the time consuming tasks of entering and analyzing complex three-dimensional (3D) models. However, three-dimensional modeling of structures becomes a necessity when analyzing large buildings with complex geometries under lateral loads. Nowadays, most structural engineering packages provide capabilities for creating a 3D model of the structural system (see Figure 2.15). Taking advantage of geometric modeling techniques, software applications allow the generation of detailed 3D models of a building structure using actual structural elements (physical members such as beams, columns, braces, walls and slabs) rather than analytical components (nodes and finite elements). Thus, elements are modeled having true cross sectional details instead of a wire-frame representation.

For example, RAM Modeler by RAM International helps the user in generating a detailed 3D model from pre-defined structural members and connections represented with true shapes as selected from tables. Once the model is completed, it serves as input to the RAM Steel module, which applies the different kinds of loads to the structure, automatically distributes and computes the loads on each member, performs analysis, member optimization, and design of beams, joists, girders, columns, and base plates. The structural model can also be input to the RAM Frame module, which computes all the parameters relevant to finite element analysis and performs static and dynamic analysis of the lateral load resisting system.

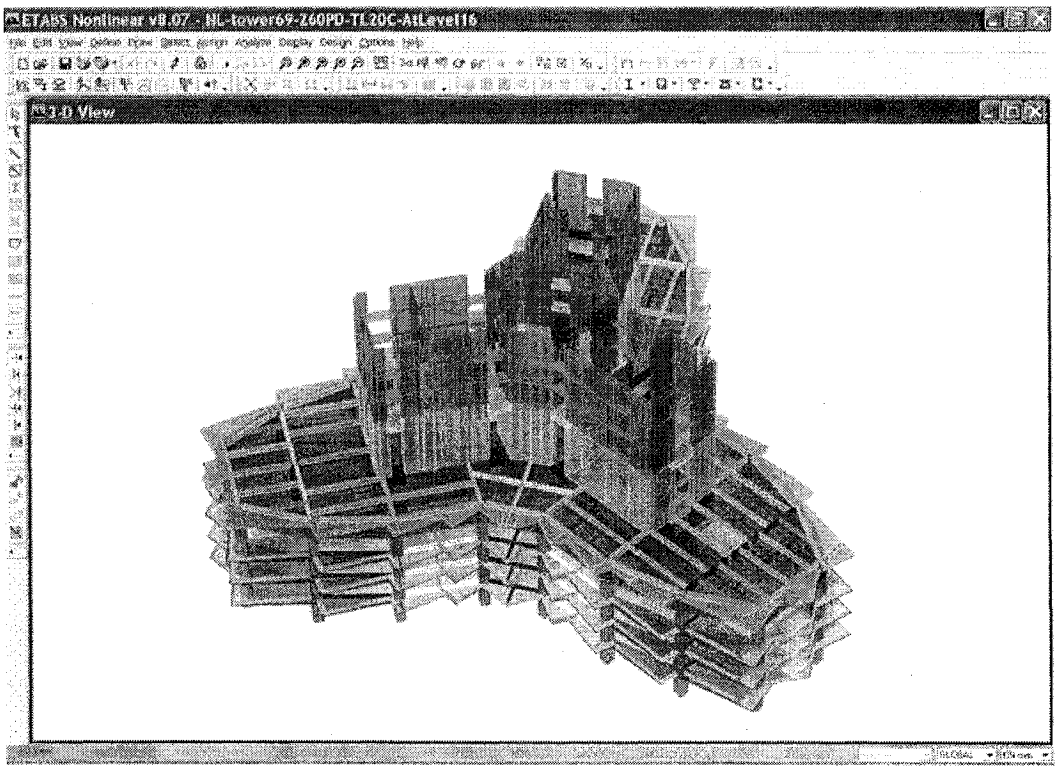


Figure 2.14 3D modeling capabilities in ETABS

More advanced modeling features are also provided by some applications in this group, such as "intelligent" structural elements that "know how" to connect to each other (see

Figure 2.16). Depending on the structural elements being positioned and their geometries, the software automatically defines the type of connection required. In addition, structural elements are parametrically associated to each other so that when a change is made in the geometry of one element, it automatically produces a change in associated elements so that the model integrity is preserved.

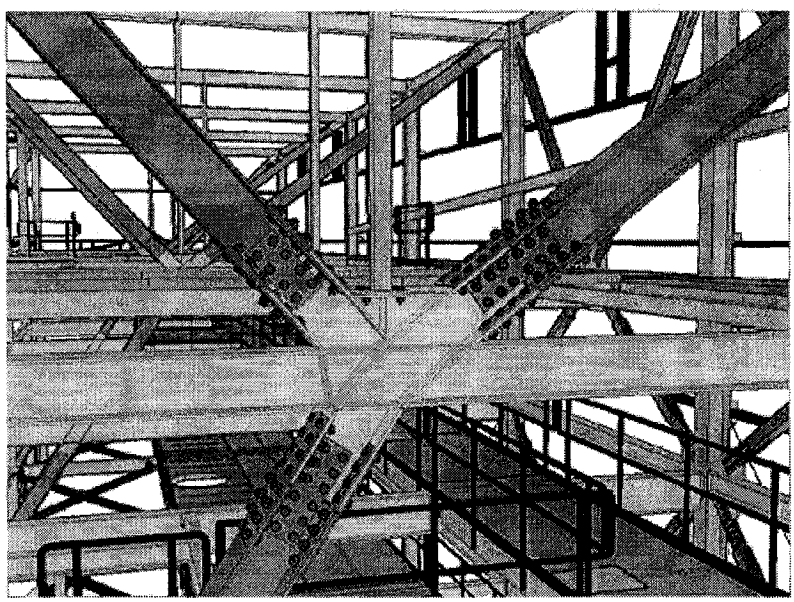


Figure 2.15 "Intelligent" structural elements that "know how" to connect to each other in Structures by Graitec

In recent years, other applications in this group have emerged that help the engineer in producing the 3D model of the structural system directly from the building architecture (see Figure 2.17). Such packages normally work within architectural CAD environments. Examples are Advance from Graitec, IdeCAD by IdeYAPI and Microstation Structural.

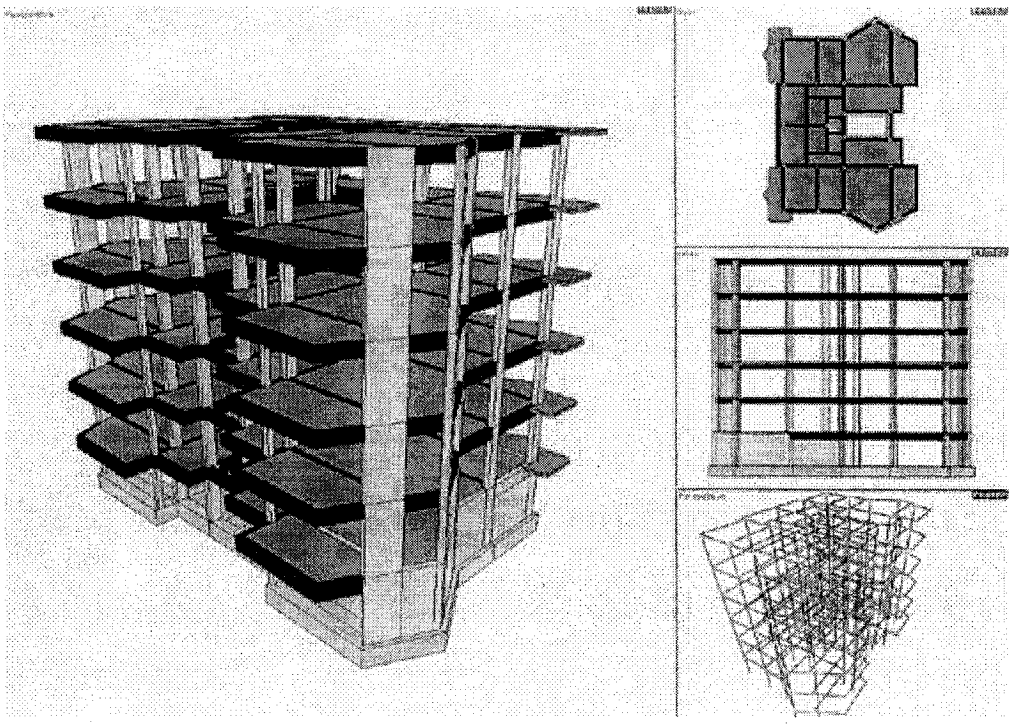


Figure 2.16 The structural system generated directly from the building architecture in IdeCAD

For example using Advance, the engineer creates the structural system model directly from an architectural model produced using Autodesk Architectural Desktop. Thus, walls and other elements from the architecture can be selected by the engineer and integrated to the structural system. Microstation Structural allows the generation of the structural model from the architectural model developed using Microstation Architectural. IdeCAD provides an integrated CAD environment where both the architect and the engineer are able to work together. Therefore, when the architect wants to create the building architecture, he/she sets the environment to the architectural mode; conversely, when the engineer wants to work on the structural system, he/she switches to the structural mode. While both modes provide shared capabilities, they also add specific capabilities according to the domain tasks to be performed. For example the structural engineer has access to entities (e.g. beam, column, footing, etc.) that are not accessible to the architect

and the architect has access to entities such as walls and windows that are not accessible to the engineer. In addition, IdeCAD allows the engineer to construct the model using various construction aids such as: axis, arrays, alignments, offsets, parallel relations, as well as parametric relationships with other structural entities. An interesting feature in IdeCAD is that when the engineer wants to work in a given storey, the application shows him/her elements in the storey below which may act as supports for the current storey. For the architect, the storey below is not shown.

In general, this last group of applications allows the engineer to construct the model of the structural system directly from the building architecture by selecting construction elements from the architecture and integrating them to the structural system. In doing so, they use a building architectural model as a reference base for constructing a model of the structural system. The resulting structural model therefore shares some construction elements, such as walls, with the architectural model. It is apparent that these applications are coming closer to providing an integrated alternative for conceptual design of building structures. However, they still have limitations, namely:

- They promote the construction of the structural system model element by element, rather than helping the engineer in making decisions and synthesizing and refining this model from functional wholes (e.g. independent structural volume, structural subsystem and assembly).
- Such applications give no assistance for engineers in reasoning while making decisions or while inspecting the building architectural model and selecting construction elements from this model. For example, they provide no tools for facilitating the engineer's search for continuous load paths to the ground.

2.3.2.3 Specialized packages for integrated analysis and design

Integrated analysis and design packages constitute the kernel of structural engineering software. For researchers in the area of computer-aided structural engineering this is the main topic for research and development. Examples of packages in this group are: SAP 2000 and ETABS by Computer and Structures Inc., STAAD.Pro by Research Engineers International, RAM Structural system by RAM International and GT-STRUDL from Georgia Institute of Technology. These packages allow designers to perform detailed analysis for predicting, with good accuracy, the behavior of complex structures under many design load combinations as required by international building codes. Besides the traditional analysis methods, they now include dynamic analysis, non-linear analysis, and finite element analysis modules for accurately modeling complex structures under more realistic loading conditions. For predicting structural behavior under dynamic loading conditions, for example, they incorporate periods of vibration and complete response spectrum analysis. Design of structural elements and assemblies is performed based on traditional materials such as steel and reinforced concrete, as well as composite construction. With steel, for example, all these packages supply complete material tables along with their physical properties and section tables. They incorporate design methods and codes from different countries (e.g. Canadian limit states design).

2.3.2.4 Component analysis and design

Despite the availability of integrated analysis and design packages, interviews with structural engineers have shown that it is not uncommon for them to use ancillary in-house developed applications and spreadsheets to design individual structural elements or

even complete assemblies. Component analysis and design packages provide assistance to designers interested in the analysis and design of specific structural components, such as retaining walls, floor slabs, continuous beams, columns, base plates, etc. These are often included as modules in specialized analysis and design applications. Examples of these packages are Floor2 by Structural Concrete Software Inc. and Beam 2D 3.1 by ORAND Systems Inc.

2.3.2.5 Detailing and drafting

These applications are used for the production of detailed fabrication and construction drawings in compliance with the applicable codes, as well as customized reports (including bills of material, and material takeoffs). Examples in this group include SDS/2 software by Design Data and Structures by Tekla. In addition to detailing, the designer can also perform design of connections, web stiffeners, joist seats, and other accessories according to relevant code specifications. Once detailed and designed, accessories and connections can be assigned to the structure, either individually or by a global assignment method. In addition, the applications check for member connectivity and proper support for beams and columns. Some of these applications can either work independently or as detailing sub-modules of a specialized analysis and design package. For steel, detailing software can download data directly into fabrication equipment. Taking advantage of 3D modeling techniques, most detailing and drafting applications provide features for generating the structural system model in a constructive bottom-up fashion.

2.3.3 Conclusions from software overview

Considering architectural CAD software, there is a still ongoing paradigm shift from drafting to digital building modeling (or virtual building modeling). This paradigm shift comes from the necessity to manage design, construction and facilities management information in a more intuitive, consistent and efficient manner. Related efforts are concentrating in sharing and exchanging digital building model related data among applications throughout the building life-cycle. CAD developers are also interested in supporting the early stages of architectural design and linking this support to downstream already supported stages. However, in general, the following limitations have been identified in these new architectural CAD software trends: there is still a lack of support for facilitating design changes, CAD applications are still not able to capture the architect's design intent, these applications offer little capabilities for abstraction and decomposition of building entities for enabling design refinement and multiple views for each design participant, support for conceptual design has not reached the level required by architects since they are still reluctant about using a technology that could "constrain" their design freedom and creativity.

Considering structural engineering applications, it is clear that software developers are interested in developing applications that support the entire design process. However, this support is still incomplete for the conceptual design phase. Despite the fact that more 3D modeling tools are emerging in the market, analysis plays a dominant role in structural engineering while synthesis is still secondary. With no exception, all the applications studied promote a constructive, bottom-up, approach for generating a model of the

structural system. The constructive approach for conceptual structural design works well for the design of small buildings. However, this bottom-up approach for model generation, element by element, becomes tedious and cumbersome for complex and larger buildings having hundreds or even thousands of structural elements and connections. The approach proposed in this research project aims at moving one step forward by taking advantage of the hierarchical organization of the structural system for structural synthesis and its integration with the building architecture.

2.4 Research in conceptual design of building structures

Over the last two decades, researchers in conceptual design have relied on strategies from the field of artificial intelligence (AI) for tackling poorly-defined problems. *"AI is the branch of computer science that is concerned with the automation of intelligent behavior"* (Luger and Stubblefield 1989). By definition, AI problems share some common characteristics: focus on heuristics and search-based rather than algorithmic solutions, deal with inexact, missing, or poorly defined information, provide satisfactory answers that are neither exact nor optimal, and deal with qualitative features rather than numerical methods. AI has been supported by an area of psychology called cognitive science where cognition is the study of how humans process information, of how people think, especially when solving problems (Giarratano and Riley 1998).

Every problem solving effort must begin with creating a representation for the problem - a problem space in which the search for the solution can take place (Simon 1996). In order to represent a design problem one has to represent, on the one hand the reasoning (i.e. problem-solving) process and, on the other hand, the physical system that is the

subject of this reasoning. Concepts or objects that are not represented cannot be reasoned about (Luth et al. 1991). In the area of conceptual design of building structures many representation schemes have been proposed for modeling the design process (i.e. the design knowledge and reasoning), as well as the information that is derived as an outcome of design (i.e. the physical system). In a generic sense, the goal of the majority of research projects that aim at supporting conceptual design of building structures is to generate feasible structural systems, which are described in terms of material(s), frame and floor system types, overall configuration and approximate member sizing. Once generated, the structural solutions are then evaluated mostly in terms of self-weight, and cost. Structural behavior is usually verified using simplified analysis methods. Feasibility (hard) constraints guide the generation process (e.g. compatibility of structural subsystems and assemblies), while desirability (soft) constrains (i.e. related to preferences by the owner and the designers) are applied during the evaluation of alternatives. The most significant research works are described in the following section.

2.4.1 Knowledge representation and reasoning

This section presents the research approaches for representing the structural engineering knowledge and reasoning during conceptual design. These approaches are grouped according to the main paradigm that supports them, namely: knowledge-based expert systems, formal logic, case-based reasoning, evolutionary approaches, generative grammars and algebras, first principles and hybrid approaches that combine two or more paradigms. It should be noted that some of the research projects discussed in this section aim at supporting the preliminary stage of structural design. However, they are classified

here as conceptual design approaches because they provide an initial description of the structural system in terms of the layout of its members and their tentative cross-sectional properties, their connectivity, and materials (cf. 2.2.2.2 a).

2.4.1.1 Knowledge-based expert systems

Knowledge based expert systems (KBES) were among the first AI paradigms to be used for supporting conceptual structural design. KBES incorporate expert heuristics in a knowledge base (i.e. generalized knowledge), which are then used by an inference engine to develop a solution to the problem, thus emulating the decision-making activities of a human expert. Representative examples of KBES to support the early stages of structural design are HI-RISE (Maher 1985), STRYPES as part of the IBDE project (Fenves et al. 1994), and Tall-D (Ravi 1998).

HI-RISE (Maher 1985) is an expert system for preliminary structural design of hi-rise, rectangular office buildings. Input is the three-dimensional location of the structural system and service core, along with loads and other functional requirements. Output is a description of feasible structural systems. The structural feasibility of an alternative is determined exclusively via domain heuristic rules and confirmed by approximate analysis using the portal method. Approximate sizing of steel and/or concrete members is also performed using heuristics, as well as first principles. Evaluation of alternatives takes place using weighing factors to emphasize the relative importance of certain design features. HI-RISE was a pioneer in the application of AI to the domain of structural engineering, as one of the first attempts to computerize the reasoning process in structural engineering. One of the main limitation of HI-RISE comes from the fact that the

structural system has to be already configured in space in order to be used as input to the program.

The above limitation was partially overcome in a later project called the integrated building design environment (IBDE) by Fenves at al. (1994). STRYPES is the conceptual structural design module of IBDE. It is based on the knowledge acquired during the development of HI-RISE. Its input is the structural grid specifying potential locations for structural systems, and is produced by an upstream module called ARCHPLAN (a knowledge-based architectural planning expert system). ARCHPLAN performs overall building configuration by laying out the floor plan with arrangements of adjacent rectangles, using the LOOS system (Flemming 1990), and then stacking floor plan layers vertically over the entire building height. A second version of IBDE introduced an intermediate module between ARCHPLAN and STRYPES called GRID to allow the user to generate structural grids that override the grids based on the limited structural heuristics of ARCHPLAN.

Tall-D (Ravi 1998), a KBES developed at Concordia University, aims at integrating design of multistory office buildings by combining heuristic knowledge mainly from the domains of architecture and structural engineering. The input for Tall-D consists on the dimensions of the land, the maximum number of floors, an available budget, and a minimum required floor area. In addition, preference constraints are introduced by the designer including the relative importance in the overall design of configuration features such as flexibility of rental areas, lateral structural system, travel distance from core, amount of daylight, external view, etc. Tall-D solves the design problem in two phases: first it performs overall building configuration (i.e. the architecture), next it performs

structural systems configuration. Initially, Tall-D presents to the user several rectangular floor plan configurations showing feasible arrangements of two rectangles of varying dimensions (i.e. the service core and the rentable floor area). It also computes the resulting number of stories, aspect ratios, and an overall evaluation and ranking of each alternative according to the user predefined preferences. The structural system is then configured using as a template the selected floor plan layout. Several lateral and gravity load-resisting schemes are incorporated, and their selection and geometric configuration depends on a variety of factors implemented with heuristic rules. For analysis the portal frame method is used, and for sizing approximate methods based on charts and codes are incorporated.

The above researchers contributed in the formalization of the structural engineering knowledge at the conceptual level and the identification of heuristic rules for reasoning during conceptual and preliminary design stages. However, these systems suffer from the inherent limitations of KBES, namely: they cover narrow domains, they are not easily extensible, they emphasize shallow, non-causal, knowledge for reasoning and they do not easily scale up into more complex geometries. The first limitation implies that integration between disciplines in design using KBES, although not impossible, may be a cumbersome task (i.e. the task of hard-coding all the experience from designers in a computer as rules of thumb). In addition, hard-coded knowledge is difficult to upgrade. Therefore, KBES cannot learn from new cases or new design experiences (i.e. their knowledge is static). Emphasizing shallow heuristic knowledge means that function and behavior of the structural system are not explicitly represented. Hence, they cannot be used for reasoning, for instance through engineering first principles. Finally, managing

geometry exclusively via heuristic rules and operations on rectangles is restrictive and forces designers to simplify spatial design alternatives. The above limitations translate in a lack of flexibility that limits the use of KBES for practical conceptual design applications.

2.4.1.2 Formal Logic

Reasoning in terms of form, function and behavior, rather than heuristics was proposed by Luth et al. (1991) and Jain et al. (1991) for solving conceptual structural design problems. They formalized the engineering reasoning using formal logic based on first-order predicate calculus. For example, a truth statement such as, "a post-and-beam frame is unstable" is included in the logic. The approach also incorporates multidisciplinary aspects of design in the form of exogenous constraints (i.e. coming from factors outside the structural domain), such as architectural and mechanical constraints. Examples of architectural constraints are: typical architectural planning module, minimum office width, no-column zones, etc. As an example of reasoning in terms of function, the system establishes the need for a structural element from the fact that the building structure carries loads from above ground levels to the ground; it is inferred that a transfer of load in the vertical direction is needed. Since the primary function of columns is to transfer applied loading in the vertical direction, the element column is then selected for performing the specified function. Using this strategy, Jain et al. (1991) developed a prototype called FFG (floor framing generator), which generates floor framing schemes for steel office buildings that are rectangular in plan and have a single service core. From an initial input consisting of basic functional requirements and floor plan geometry, FFG

first defines column locations; then it configures the floor system by defining the type of floor system, as well as beam and girder locations. Finally it performs member sizing.

The main limitation of predicate logic comes from the fact that this paradigm is driven by deductions based on logic statements. Thus, the universe of solutions is constrained to the paths defined by deductive logic. This makes solution paths rather predictable and limits the applicability of software based on the predicate logic paradigm to represent real-life situations. In addition, from the perspective of fundamental logic, the dominant forms of inference for reasoning during design synthesis are abduction and analogy (Raphael and Smith 2003) and not deduction. Therefore, at the conceptual stage, deductive logic can only be used for verification purposes of already synthesized conceptual design solutions.

2.4.1.3 Case-based reasoning

Analogical reasoning uses episodic knowledge for creating designs by comparing the design at hand with previous similar design solutions. Therefore, when faced with a new problem, designers do not start solving problems from scratch, but search for solutions from previous similar design experiences. Case-based reasoning (CBR) is an AI paradigm that accumulates previous design solutions, uses analogy to retrieve similar cases from the case library, and adapt the most similar case to the problem at hand. Four major aspects characterize CBR systems: case representation that addresses how the case information is represented, case accumulation that is responsible for storing new cases in the case library, case retrieval is responsible for finding the most similar cases stored in the case library for the problem at hand, and case adaptation transforms the selected case into a solution for the current problem. Over the last ten years several research projects

have relied on the CBR approach in an attempt to support structural design. Representative examples include: CADRE (Bailey and Smith 1994), CADREM (Kumar and Raphael 1997), and SEED-Config (Rivard and Fenves 2000b).

CADRE (Bailey and Smith 1994) aims at integrating architectural and structural design based on previous cases in which integration has already been achieved. It is one of the few CBR systems that use geometric models for case representation. In CADRE a case consists of a geometric model of both the structural system and the architectural layout of spaces, i.e. it does not incorporate any explicit architectural or structural domain-specific descriptions (e.g. stories, spaces, walls, columns, etc.). Considerable knowledge is required during adaptation to interpret a "plain" geometric model. CADRE focuses on the representation and adaptation of cases rather than their indexing and retrieval. Users decide which building they would like to adapt by selecting an appropriate case. Once a case is selected by the user, CADRE uses a knowledge-base to generate constraints for that case automatically (infers constraints from the geometric model). The user also posts additional dimensional constraints. At this point the model may become over constrained. A technique called dimensionality reduction is therefore used to identify the key parameters for adaptation. Then, CADRE performs parametric dimensional adaptation by trying to modify the geometry of the model without affecting its topology. In the event that a solution cannot be found by dimensional adaptation, a topological adaptation is performed, in which case the topology of the spaces and the structure are affected. Domain knowledge is used to decide when the type of construction needs to be changed. For example, a flat slab floor system is suitable up to spans of 8 m, but beyond that, a beam and slab system is preferable. The function of structural elements is also considered

during topological adaptation. Elements that contribute to the overall stability of the building (e.g. bracing and shear walls) are identified automatically and maintained during adaptation. An interesting feature of CADRE is that it allows the user to reason in terms of the geometry and topology of the model being created (for case adaptation). The main limitations of CADRE are related to its scalability. CADRE succeeds in performing adaptation for relatively small rectangular buildings. However, tackling dimensional and topological adaptation via constraint solving methods may become intractable for large buildings with complex geometries.

CADREM (Kumar and Raphael 1997) represents design methods instead of design solutions. Design methods are represented as a decomposition hierarchy (i.e. a method for a particular task has several sub-tasks that have their own methods and so on). A method can be viewed as a combination of rules and procedures used to arrive at a specific design solution. The approach is tested in conceptual structural design of buildings for integrating building architectural and structural layout design. Thus, given an architectural layout of spaces, CADREM uses pre-defined design methods from the case-base to generate alternative structural layouts. Note that different design methods may arrive at the same solution. The most important limitation of representing design methods is that these methods need to be extracted manually from past designs as they are not readily available anywhere.

SEED-Config (Rivard and Fenves 2000b) is a design environment for conceptual building design which incorporates case-based reasoning functionality to provide designers with initial potential solutions. An information model is used in SEED-Config to represent cases. The information model has the advantage of supporting the

hierarchical decomposition of design cases. In its current state, case representations in SEED-Config include reduced geometric descriptions for design solutions. However, the information model provides the capabilities for adding more complex geometric descriptions, including geometric models, in the future.

Case-based reasoning has the potential for becoming a useful tool for conceptual design since representing previous cases in a case-base is a necessary first step in organizing and accessing previous knowledge. However, there are still many issues to be resolved that limit the applicability of CBR systems in design, for example: how can design cases be stored without interrupting the workflow of a designer? How can they deal with conceptual design information that is imprecise and incomplete? How can they manage efficiently the geometry and topology of a design? Retrieving previous cases based on similarities with a new case at hand and taking advantage of those similarities for solving the new case can be left at the discretion of the user. By contrast, the practical application of the adaptation process remains questionable: why would a designer want a computer to tell him how to design, for example how to adapt a previous case when the source of inspiration (i.e. through analogy) has already been given to him/her (as retrieved cases)?

2.4.1.4 Evolutionary approaches

In recent years, genetic algorithms (GA) have been proposed for solving conceptual design problems. GA is a stochastic search technique that is used to find progressively better solutions to a problem largely through discovery (Rafiq et al. 2003). In design applications, the chosen design parameters are encoded so that each becomes an artificial gene within a genetic string of chromosome. Starting with an initial population of parent

designs generated at random, new designs evolve through a reproduction process, using crossover and mutation, between design generations. For each generation, a fitness function evaluates the fitness of individuals. Thus, the fittest individuals always have the highest probability of mating and passing their genes to next generations.

In conceptual structural design Grierson and Park (1993) used a GA for simultaneous optimization of member sizes and geometry of a plane frame. More recently, Grierson and Khajehpour (2002) presented a method for conceptual building design applied to office buildings relying on a multi-criteria genetic algorithm to search for Pareto-optimal design solutions. Multi-criteria mean that the design of the building is governed by multiple objective criteria that may be in conflict with each other in the sense that improvement of any one criterion may occur at the expense of one or more of the other criteria. Pareto-optimal design solutions define the best trade-offs between competing objective criteria. Three cost-revenue objective criteria are used: (1) minimize initial capital cost, (2) minimize annual operating cost, and (3) maximize annual income revenue. The three objective criteria are functions of the design parameters, which carry unit costs and are subject to constraints. The design parameters are: type of structure, type of floor system, number of stories, building footprint dimensions, bay unit dimensions, core dimensions, floor area, window ratio, window type, cladding type, number of elevators, number of staircases, annual energy consumption of the mechanical system and type of lighting. Pareto-optimal design solutions are plotted in colored graphs that show the effect of the various design parameters in the objective criteria and facilitate design evaluation.

Sisk et al. (2003) proposed a GA for conceptual design of steel office buildings, which receives as input simplified geometrical information of the building (e.g. plan dimensions and number of floors), site location, site regulations, location of cores and atria, and dimensional constraints. The design solutions are encoded into chromosomes having three parts: the first part contains x-coordinates of the columns, the second part contains y-coordinates, and the third part consists of other aspects of the design such as the environmental strategy in relation to building services systems, and the clear floor-to-ceiling height. Structural elements are sized using heuristics such as span/depth ratios. The aim of the fitness functions is to minimize cost, maximize clear spans and maximize the use of natural resources, which depends on the environmental strategy. A software prototype called BGRID was developed that produces the best design solutions for short-, medium- and long-span systems. The user can edit the design solutions, for example by moving the location of the atrium or changing the clear floor-to-ceiling height. BGRID then checks amendments and reevaluates the fitness. If the resulting structural grid does not fit the building footprint, the outermost bay is adjusted accordingly. Likewise, columns are moved to ensure that a column is at each corner of a core and atrium.

In a parallel effort, Rafiq et al. (2003) have developed a structured genetic algorithm (SGA) for conceptual design of steel and concrete office buildings. SGA is a variant of GA, in which design solutions are encoded in a hierarchy of interrelated genes. Thus, the SGA allows better exploration of design solutions by switching on and off or changing values on particular branches of the hierarchic chromosome. The solutions encoded in the chromosome include the following information: frame type, floor type, and structural grid in x and y directions. The fitness function evaluates the maximization of profit,

which is the subtraction of capital cost of the structure from total income expected. Once a design solution is found, member sizing is done using the standard methods given in codes of practice. The designer can also modify any parameter in the solution and re-compute the best solution. For example, if the GA finds a steel frame to be the best choice in a particular situation, the user can still find out what happens when concrete is used. Therefore, the GA can be used for parametric studies to see if changing the value of one design parameter will effect the selection of other parameters when seeking an optimal overall design.

Genetic algorithms have the potential of representing complex design systems in the chromosomes. However, their efficiency and effectiveness are yet to be demonstrated in the synthesis process of systems with complex geometries. In particular, it is not clear how can these algorithms be used for solving structural synthesis problems under stringent architectural geometric and topologic constraints.

2.4.1.5 First Principles

Reasoning based on first principles during the conceptual design stage was proposed by Fuyama et al. (1997). They developed a software prototype called BERT in which, given the configuration of the structural system, the prototype uses engineering first principles to select efficient member sizes through reasoning about the behavior of the structure. The member sizing approach uses approximate analysis methods, which are based on simplifying assumptions. For example, it is assumed that in the frames subjected to lateral loads the points of contraflexure are located at the mid-height of the columns and mid-span of girders. In the design process, initial member sizes are determined from

strength considerations and some of them are later overridden by flexural and shear drift considerations.

Even though approximate member sizing is iteratively and interactively performed with system configuration, obviously, the configuration task must always come first. The design process using BERT assumes a given initial configuration which is tested through approximate analysis to obtain member sizing that aims at "minimizing" member weight within the given constraints of allowable stress and member depth. If constraints are not satisfied then design parameters must be changed, such as the given building configuration. Therefore, the approach supports only the member selection task of conceptual design.

2.4.1.6 Generative design

Generative design systems aim at producing design descriptions automatically or semi-automatically. All the AI techniques described so far can be classified as generative since they produce partial or complete design descriptions semi-automatically. However, the following AI techniques are meant to be more generative in nature (i.e. giving higher priority to automatic design generation): grammars, algebras, shape annealing and spatial tiling. Grammars implement a synthesis by transformation approach based on generalizations. It is called synthesis by transformation since transformation rules are used to develop (i.e. transform) an initial design idea into a design solution. The grammar paradigm has been considered as an alternative to KBES for design synthesis. Like KBES, grammars work based on rules, however of a different type. In addition, their syntax and underlying mechanisms are essentially different.

Meyer (1995) proposed using the grammar paradigm for synthesizing the structural design of tall buildings. It incorporates architectural and engineering knowledge in grammatical rules to achieve design integration. However, unlike other approaches, it allows building design with multiple occupancies, which are stacked vertically over the building height. Initially, a grammar organizes the architectural program requirements, which allow the definition of the architectural form into various occupancy units (office, apartment, parking, etc.). Then, another grammar generates the architectural 3D model from bottom to top by stacking massing volumes corresponding to occupancy units one on top of the other. Setbacks are accepted but overhangs are not. The next step is the definition of structural grids followed by the definition of feasible "abstract" structural systems (providing feasible patterns but not yet populated with assemblies and elements) within each occupancy unit. Next, there is a vertical integration of structural grids and "abstract" systems between occupancy units. Finally, "abstract" structural systems are refined into specific structural systems (e.g. braced-frame system, framed-tube system, etc.).

Using NOODLES (Gursoz et al. 1991) geometric modeler as a substrate, Zamanian (1992) developed an algebra of operations for the definition of the building geometry into a single spatial model. Using this algebra, entities in the building are defined spatially with respect to some reference geometric entities, called superior elements. Superior elements are therefore used to represent and identify the spatial extent of any geometric element in a given spatial configuration. Thus, a given building configuration is created through a combination of user specified superior elements plus the following algebraic operations: labeling, decomposition, selection, composition and condensation.

Even though the above examples can be categorized as generative, they still require significant guidance and intervention by the designer during the design process (similar to the examples in sections 2.4.1.1 through 2.4.1.5). This is because they are aimed at assisting engineers in the design of entire building structures (i.e. globally), a highly constrained problem that is coupled with the architectural design. As a consequence, their solutions are limited to office buildings, usually of the conventional type, with flexible open spaces.

By contrast, generative techniques have also been envisaged as a means for enabling designers to explore complex 2D and 3D geometries for unconventional designs. Shea and Cagan (1998) have investigated automated approaches for generating and exploring design alternatives that could be applied at a local level (i.e. for large spaces or building zones). They have proposed shape annealing for designing novel roof trusses that builds structures using a shape grammar (Stiny 1980) and optimizes the structures using the stochastic optimization method of simulated annealing (Kirkpatrick et al. 1983). Informally, a shape grammar is a kind of grammar where the goal is generating shape configurations. The Simulated Annealing (SA) formalism, as its name implies, exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. The configuration of trusses using the shape annealing technique is quite different from the process generally used by a designer. Rather than relying on knowledge of standard truss configurations, shape annealing uses topology transformation rules to add and remove members based on principles of simple truss design. Shape annealing is a form-driven strategy where the structural form is modified

independent of the behavioral implications. The advantage of computer-based conceptual design with shape annealing is that the resulting design is a direct derivative of the allowable forms described by the shape grammar and the design goals formulated in the optimization model. Thus it is not restricted to intuition based on knowledge of standard forms, but rather can be generated as an innovative form based on evaluation of imposed design goals.

As another example of generative design at its best, Shea (2004) combines mathematical spatial tiling and shape grammars formalisms for generating structures inspired from natural and organic influences. Tiling can be described as a plane-filling arrangement of plane shapes whereas spatial tiling is a space-filling arrangement of three-dimensional shapes. Shape grammars share many common attributes to planar and spatial tiling. Both formalisms are concerned with the fundamental logic and rules for composing designs of shapes with certain spatial relations in Euclidean space. In this research, a mathematical tiling was first implemented as a simple shape grammar. It was then relaxed to enhance the generative power by transforming a simple grammar into an unrestricted grammar. Example applications are the generation of complex architectural spaces, structures and building façade patterns.

In general, generative design techniques have been criticized for attempting to perform the work of designers rather than to support it. It has been argued that designers do not require computer tools that create designs or even foster their creativity. Instead they want to be freed from tedious and time consuming design tasks in order to concentrate on what they enjoy more, which is the creation of innovative designs.

2.4.1.7 Hybrid approaches

Hybrid approaches combine AI paradigms in order to benefit from the strengths of each one in some aspects of the design process. Krishnamoorthy and Rajeev (1996) developed a hybrid AI-system that combines a KBES and a GA for generating structural frames for office buildings with articulated floor plan and continuous vertical elevations. The application first decomposes the complex plan into several rectangular zones. The zones are classified as either free or constrained zones depending on whether or not they contain elements that constrain the placement of columns within them, such as services cores and shafts. The AI-system first generates an initial population within each zone using heuristic KBES techniques. From this initial population, the GA produces several generations until an "optimized", least-cost solution converges. The optimization variables are the number and location of the column lines in the two orthogonal directions. The least-cost alternative generated through the hybrid approach is subjected to evaluation from both constructability and architectural viewpoints.

M-RAM (Soibelman and Peña-Mora 2000) is a CBR for conceptual structural design. M-RAM's main features are a distributed architecture that runs over the internet and a multi-reasoning mechanism that combines heuristic rules for case classification, CBR for retrieving previous cases, and a genetic algorithm (GA) for case adaptation. Thus, based on the initial project information introduced over the Internet, the M-RAM manager first calls the classification agent for classifying the case according its own knowledge, then it calls the CBR agent that retrieves similar cases, and finally the GA adaptation agent that uses the population retrieved from the CBR agent to retrieve the URL of the best matching case along with its structural system. In M-RAM nineteen properties describe a

case: height from street to roof, number of stories, number of levels below ground, building use, frame material, typical floor live load, maximum lateral deflection, design acceleration, typical floor beam depth, etc. The outcome of each case is given as braced frame, moment-resisting frame, shear wall, core and outrigger, tubular or hybrid. Therefore, based on the general structural requirements of a new case, a complete best matching case is retrieved. However, this approach does not consider building geometry or multidisciplinary design in its cases.

2.4.2 Physical system representation

The task of developing a computer representation that describes an artifact to be designed is interdependent with that of devising problem solving techniques for solving the design problem. A design representation for supporting conceptual building design must satisfy the following requirements (Rivard and Fenves 2000b): (1) integrate multiple functional views for the different participants involved in the design process, (2) support design refinement/evolution, and (3) favour the exploration of alternative design solutions (e.g. enable iterations and changes in the design). Several computer representations have been developed for supporting building design, however most of these representations are not specifically tailored towards supporting conceptual design. Such representations vary in scope, as well as on the emphasis they place on a set of specific design requirements. The scope of the representation usually dictates whether a semantically explicit representation (i.e. exhaustively incorporating the concepts that are relevant to the domains) is more convenient than a syntactic representation (i.e. generically defining the data structures and the rules for organizing the information).

2.4.2.1 Semantically explicit representations

Several semantically explicit representations have been proposed for supporting structural design. Most of these representations model structural entities exclusively while including some strictly necessary architectural entities, such as storeys and walls. Some relevant examples are Law et al. (1990), Biedermann and Grierson (1995), and Fuyama et al. (1997). Only few representations provide more detailed descriptions of the building architecture and its relation to the structure: these are from Björk (1992), Dias (1996), Khemlani et al. (1998), and Sacks and Warsawski (1997).

Bjork, Dias and Khemlani et al. tackle the problem of formalizing the relationship between building spaces and their boundaries. All of them assume that structural elements always lie along the boundaries of spaces, which is a limiting assumption since structural elements also often lie within spaces. In addition, none of these representations consider the role of high level structural entities (e.g. structural massings, sub-systems and assemblies) in specifying and laying out lower level structural elements. As a consequence, they support neither abstraction nor decomposition of the structural system for design refinement.

Sacks and Warsawski (1997) proposed a representation that incorporates product and process considerations, and therefore is called project model (i.e. a product model plus a process model). It relates three main hierarchies that represent: spaces, functional systems, and construction processes. The representation describes various types of spaces and decomposes functional systems into assemblies and elements. Spaces contain assemblies and elements that are installed by construction activities. This representation supports design refinement through functional systems, assemblies and elements. It also

supports multiple views of the design since a functional system can be structural, lighting, mechanical, etc. The main limitation of this representation for conceptual structural design comes from its correspondence between the decomposition of spaces and construction activities. Hence, the representation does not accept overlapping space aggregations, which are useful for providing multiple design views. For example, architectural concerns may require the distinction between private and public zones, which may be described as a single zone for structural purposes.

2.4.2.2 Syntactic representations

Turk (2001) argues that representation models are subjective (i.e. they respond to their developer's partial understanding of reality) and hinder creativity (i.e., they force designers to use predefined semantic structures). Thus, he advocates for generic (i.e. syntactic) models because they impose no semantic structures on designers. Rivard and Fenves (2000a) developed a syntactic representation for conceptual building design. The representation supports hierarchical decomposition of the design problem, as well as several reasoning mechanisms, such as heuristics and case-based reasoning. The representation is built upon three layered models: an underlying object-oriented data model, a generic information model built on top of the data model, and a conceptual model incorporating the semantics for a particular design situation. Similarly, Datta and Woodbury (2004) developed a syntactic construct, called Feature Nodes that aims at capturing the rationale underlying the design process. The Feature Nodes construct emphasizes the support for design exploration and therefore it captures the exploration history as a collection of ancestor and progeny feature nodes. The above generic

representations exhibit good capabilities for supporting conceptual design, however they overlook the role of spatial information in conceptual building design.

2.4.2.3 Standardization efforts

Current standards for information exchange in the design and construction industry, the Industry Foundation Classes (IAI 2004) and CIMSteel standard (SCI 2004), are based on an integrated representation model that incorporates all the entities, attributes and relationships relevant to the domains involved in the building design, fabrication, construction and operation. Thus they attempt to incorporate all functional views of the building. The Industry Foundation Classes (IFC) use object-based data modeling principles to provide complete descriptions of real world building objects, thus enabling design and construction related applications to use such objects in a consistent and interoperable manner. IFCs provide a layered architecture with a base syntactic layer that describes entities generically, upon which a semantically rich layer is built including entities from different building domains. In the structural engineering domain, there are four completed IFC projects that integrate structural engineering information with the IFC object model, namely: steel frame constructions (ST-1), reinforced concrete structures and foundation structures (ST-2), pre-cast concrete construction (ST-3), and structural analysis model and steel constructions (ST-4). Another project is currently underway: structural timber model (ST-5).

For the IFC ST-1 project, a liaison was made between the IAI and the developers of a structural steel standard developed in the UK by the Steel Construction Institute (SCI 2004) and the CAE group at the University of Leeds. The name of the published standard

is CIMsteel Integration Standard version 2.0 (CIS/2). CIS/2 provides a semantically explicit representation for describing steel structures. CIS/2 developers have produced formal specifications for a standard data exchange format so that software applications supporting structural steel work can map their own representations into that neutral format and exchange information easily, thus making applications mutually compatible.

The above standardization efforts are significant contributions to the integration of information in the design of building structures. However, the following limitations have been identified in relation to the support for conceptual building design: their component descriptions are limited to tangible building entities (e.g. doors, walls and beams) and are not able to model purely functional building entities (e.g. structural subsystems and assemblies), which are particularly important during early design stages (Rosenman and Gero 1996). In addition, their component descriptions are too detailed since they are more geared towards detailed design and construction (or fabrication and erection for steel structures). For example, they describe reinforcing bars for concrete and provide precise specifications for steel connections. Detailed information results in unnecessary model complexity, and in adverse effects on design exploration during early design stages.

2.4.2.4 Flexible and extensible representations

Despite the standardization efforts and support for integrated building representations, some researchers criticize integrated representation models as being rigid since they impose fixed organizational structures to the information that give no room for design creativity and exploration (Turk 2001, Haymaker et al. 2004, Stouffs and Krishnamurti

2004). Haymaker et al. developed an approach that promises to augment integrated representation models and render them more flexible. To overcome the limitations of integrated representation models they developed a so-called "Perspective Approach", which is a mechanism for augmenting representation models. The mechanism allows new entity types and dependencies to be defined by the user during building design and/or construction. It incorporates a spatial reasoning mechanism called "perspectors" that, given a spatial criteria (i.e. implicit spatial relationships among entities), searches the model looking for entities fulfilling that criteria and defines explicit relationships among them. Stouffs and Krishnamurti (2004) developed a representational schema for constructing representations that is based on a construct called "sort". This schema is supported by an algebra that allows designers to build and alter representations based on their own needs. Rather than a fixed data schema, the aim is to develop an extensible vocabulary of data components that can be composed into a representational language. In the field of pre-cast concrete product life-cycle, Sacks et al. (2004) define a construct called "abstract functional objects" (AFOs) that represent, store and implement the functional properties of building assemblies and parts, and carry their fundamental parametric behavior and design intent. Therefore, a 3D parametric modeling system can be viewed as collections of pre-defined AFOs that include an initial library of common physical objects that embody them. In any particular design situation, a designer is able to define new physical objects that incorporate the functionality of existing functional objects by using the AFOs. The above approaches shed new light towards more flexible and extensible representations; however, none of them has been tested with conceptual design problems.

2.4.2.5 Representation of spatial information

Building elements (i.e. architectural construction elements and structural elements) need several spatial representations depending on their degree of refinement, as well as their domain view (Martini and Powell 1990, Augenbroe 1992, Zamanian 1992, Rosenman and Gero 1996). IFC (IAI 2004) and CIMSteel (SCI 2004) standards provide multiple geometric representations for the entities they represent. For example, for any structural element and connection CIMSteel provides one geometric description for structural analysis, another description for design and yet another description for fabrication. One approach for handling the multiple geometric descriptions of an entity is to have a single fundamental representation from which all required spatial abstractions can be derived. Zamanian (1992) calls this single fundamental representation a primary spatial representation, while the spatial abstractions are called secondary spatial representations. The primary spatial representation captures the topologic relations among facility entities, which are invariant over the different spatial abstractions of those entities. The primary spatial representation, also called skeletal representation can be regarded as a mixed-dimensional wire-frame, where 2D slabs and walls coexist in the same model with 1D beams and columns. Since the skeletal representation captures the topologic relations of entities it facilitates reasoning about topology. The skeletal representation also facilitates reasoning about geometry at the conceptual stage since structural elements usually have one or two salient dimensions that are sufficient for characterizing them within the structural configuration, i.e. the cross-section of structural elements does not affect their relative position in space (cf. section 2.1.5).

Modern commercial structural engineering applications provide detailed geometric descriptions of structural elements and connections. Maintaining these detailed geometric descriptions is computationally expensive and fruitless for conceptual structural design where simplified geometric descriptions are sufficient. By contrast, for simplicity in this research project architectural and structural elements are spatially represented using a mixed-dimensional primary spatial representation.

2.4.3 Discussion

2.4.3.1 Knowledge representation and reasoning

In an attempt to minimize architectural constraints in the synthesis of structural configurations, most researchers have focused over the years in supporting conceptual design of rectangular office buildings. Office buildings are characterized by having open floor configurations with permanent walls mostly at the perimeter and at the main and service core(s). The geometry and topology of the architecture, and the intent behind them, are therefore reduced to the minimum expression. In addition, the majority of the proposed approaches demand a tight integration between the structural system and the architecture, thus requiring a one-on-one fit (Schodek 2003) between the column grids and the architectural programmatic dimensions. As demonstrated with examples in section 2.2.3.4 b) this is rarely the case.

So far, AI-based computer support for structural synthesis has proven to be effective for buildings with relatively simple geometries or for local structural systems having reduced number of architectural constraints (e.g. roof trusses and space structures). However, using AI-based paradigms to support structural synthesis for buildings with more

complex geometries is still an elusive goal. Architectural geometric and topologic constraints, as well as constraints coming from other disciplines, make AI-based solutions practically intractable.

2.4.3.2 Physical system representation

The overview reveals a trend towards more flexible and extensible computer representations that, instead of imposing fixed data schemes, tend to become a language for enabling designers to build entities and relationships responding to their needs. However, work still needs to be done for developing and testing representations that respond to the particular requirements of conceptual building design, namely: (1) integrate multiple functional views for the different participants involved in the design process, (2) support design refinement/evolution, and (3) favour the exploration of alternative design solutions.

2.5 Research at Concordia University in building design integration

Since the beginning of the nineties researchers at Concordia University have undertaken the task of assisting building design integration with computers. Some researchers have focused in supporting the integration among disciplines using AI techniques, while some other have focused on developing integrated models or representations involving the different views (i.e. architect, engineer, client, etc.) of the building design.

For example, Bédard and Gowri (1990) proposed a knowledge-based expert system (KBES) for the preliminary design of building envelope systems. In this system, based on overall building information (including the location of the building, its type and envelope

areas), the system was able to generate feasible design alternatives and evaluate them in order to select the best one based on the designer's preferences. Another KBES was developed by Bédard and Ravi (1991) for generating space layout alternatives for rectangular multi-storey office buildings. This KBES was later combined with Tall-D, a KBES for preliminary structural design (cf. section 2.4.1.1).

Considering integrated models of the design process, Rivard et al. (1995) proposed a functional model of the preliminary building envelope design process that describes the activities involved in the design of the building envelope, as well as the data required and generated by these activities. In addition, the model considers the main building envelope requirements and the interactions between the envelope and other building systems. The final goal is to use this model to develop a comprehensive object-oriented data model of the building envelope design process, which can then be incorporated in an object-oriented database management system. Mokhtar et al. (1998) developed an information model for managing design changes in a collaborative design environment. The model uses a central database that carries data of the different building components. Rules are used to track and propagate design changes among affected disciplines. These rules organize "linking knowledge" that relates attributes of components from different disciplines in the database, so that whenever a change takes place in an attribute of a particular component, it is propagated to the related components from other disciplines.

This research builds upon the knowledge obtained from these previous research projects carried out at the Centre for Building Studies at Concordia University.

2.6 Conclusions

The first section of this chapter presents a hierarchical description of the structural system that reflects how engineers understand building structures. Next it presents a review of the conceptual design process of building structures and its interdependency with the building architecture. It is claimed that synthesis by problem decomposition is the most suitable approach for structural synthesis since in addition to simplifying the synthesis process it allows the integration of the structural system to the building architecture at various levels of the structural hierarchy.

Under the light shed by the first section, the subsequent sections of this chapter review commercial and research efforts in supporting the conceptual design of building structures. From the overview of commercial architectural CAD and structural engineering software, it is apparent that while architectural design applications emphasize the synthesis process, structural engineering applications emphasize the analysis process. However, both architectural and structural engineering applications are still weak in supporting conceptual design. In both cases, the main limitation comes from their limited capabilities for supporting the abstraction and decomposition of design concepts that would allow design refinement. Early design concepts are much less detailed and have different geometric requirements than required by current CAD and structural design software. In addition, structural engineering applications promote a constructive bottom-up approach for generating the model of a structural system. This approach works well for small and simple buildings however for large and more complex ones the constructive approach for generating the structural system model becomes tedious, cumbersome and error-prone.

Considering research approaches to support conceptual structural design, these generally rely on AI-based techniques to assist the engineering reasoning process. In doing so, they try to make the conceptual design problem more tractable by simplifying the building architecture. Therefore, such approaches are still not able to assist efficiently a synthesis process of structural systems that respond to modern architecturally expressive buildings. Considering the representation of the physical system, several interesting semantically explicit and syntactic research approaches have emerged. This research project builds upon the results from previous research.

Comparing commercial and research efforts to support conceptual structural design, it was found that such efforts go in opposite directions. On the one hand, commercial modeling approaches tend to fully exploit the capabilities of the underlying geometric modelers, and in doing so, they put too much emphasis on details of representation. They incorporate little engineering knowledge mostly from tabulated data available from codes and product manufacturers. On the other hand, research approaches tend to place less emphasis on the geometry and focus more on the problem-solving aspects (e.g. modeling engineering knowledge). Standardization efforts tend to align on the side of the commercial approaches. Henceforth, an intermediate approach is proposed in this research project. A computer representation has been developed that allows engineers to reason while synthesizing a hierarchical model of the structural system as part of the building architecture.

CHAPTER 3

METHODOLOGY

On the basis of the literature review, this chapter presents the main premises of this research project as well as the stages and technologies that are required to achieve the project goals. The first section presents the objectives of this research project, followed by its scope, the research strategy and the stages of project development.

3.1 Objectives

Computer support for conceptual structural design is still ineffective mainly because existing structural engineering applications do not recognize that structural design and architectural design are highly interdependent processes. The main premise of this research project is that, in order to properly support the conceptual design of building structures and to achieve good integration between the structural system and the building architecture, computers must allow structural engineers to configure the structural system as a hierarchical whole within a building architectural context without interrupting the creative workflow of the architect. This will allow engineers to define entities at each level of the structural hierarchy that respond to architectural functional and physical requirements. Even though this research project aims at laying the foundations for supporting the entire conceptual design stage of building structures with computers, it emphasizes the synthesis process. Consequently, the objectives of this research project are the following:

- To identify opportunities for improved computer integration between the building architecture and the structural system. The advantages of making explicit those opportunities are two-fold: (1) acknowledge the role of the building architecture in the structural synthesis process, and (2) provide guidelines for the selection of the right computer technologies that respond to architectural concerns.
- To define a formal model of interactions between architects and engineers during conceptual design of building structures. The formal model facilitates collaboration between architects and engineers during conceptual design of building structures by relying on innovative computer technologies.
- To propose a methodology for computer-assisted structural synthesis. The proposed methodology specifies how the engineer can efficiently configure the structural system hierarchy while providing feedback to the architect at different stages of structural refinement.
- To devise a design assistance environment for conceptual structural design that implements the proposed methodology for structural synthesis. For this research project, such an environment is expected to be neither comprehensive nor complete, but it must incorporate the main components that are required to assist structural engineers during conceptual design of building structures.
- To develop a proof-of-concept software prototype that implements the abovementioned components and demonstrates the main features of the proposed methodology. The prototype will be tested with building cases in order to demonstrate the validity of the methodology.

3.2 Research method

Design research can be carried out in variety of ways. According to Gero (2004), three alternative approaches have been traditionally followed:

1. *Protocol studies* - It can be viewed as an empirical endeavor in which experiments are designed and executed in order to test some hypothesis about some design phenomenon or design behavior. This is the approach adopted in cognitive science. It often manifests itself through the use of protocol studies of designers. The results of such research can form the basis of a computational model.
2. *Logic-based* - A second view is that design research can be carried out by positing axioms and then deriving consequences from them. If the axioms can be mapped onto design situations then the consequences should follow. This is the approach adopted in mathematics and logic and forms the basis of a small but powerful area in design research.
3. *Conjecture, model and evaluate* - A third view, and the most common one in the computational domain, is that design research can be carried out by conjecturing design processes, constructing computational models of those processes and then examining the behaviors of the resulting computational systems.

In this research project the third approach has been adopted. This decision has been based on the extensive studies that have been carried out up to date in the area of conceptual structural design, including books and knowledge acquisition studies. Even though those studies have contributed to the advancement of the knowledge in the field, they have not yet impacted the actual design practices of architects and engineers. This research project

builds on their results and proposes new avenues for improved computer-assisted conceptual structural design practice.

3.3 Stages of project development

This research project has been conducted according to the following stages: (1) analysis of the components required to develop the proposed approach, (2) develop a formal model of conceptual structural design, (3) planning and organization of a design assistance environment for conceptual structural design, (4) implementation of a proof-of-concept software prototype, and (5) testing and validation. These are explained below.

3.3.1 Component analysis

From the discussion in section 2.4, it can be stated that the problem of providing computer support for conceptual design of building structures can be treated as two interdependent sub-problems: develop a computer representation that properly describes the physical system being designed, and develop computational problem-solving (i.e. reasoning) methods that provide assistance in finding satisfactory solutions to the design problem. Consequently, three components have been identified to be required for supporting conceptual structural design: (1) a computer representation of the building architecture and the structural system, (2) computational problem-solving methods relying mostly in geometric modeling and reasoning techniques, and (3) a knowledge-based component that organizes and manipulates knowledge for a competent two-way communication between the engineer and the computer. From these three components, only the first two have been developed further in this research project.

3.3.1.1 A computer representation integrating architecture and structure

At the core of any computational system that can support design is a building representation, which describes all different components that make up the building, along with the manner in which they come together (Khemlani et al. 1998). The representation also encompasses, to some degree, problem-solving methods that affect all the concepts being represented (Woodbury and Damski 1997). In this research project, a semantically explicit representation has been devised for supporting conceptual design of building structures within a building architectural context. The aim is to come upstream from current structural engineering standards that are tailored towards supporting subsequent design stages. The representation provides two functional views of the conceptual design problem, namely: the architectural view and the structural view. The semantic concepts that are incorporated in the representation are standard concepts that are well defined in the architectural and structural engineering domains, and therefore they can be explicitly represented without limiting the flexibility and extensibility of the representation. Having a tailored-made representation has the additional advantage of bringing simplicity to the representation by eliminating redundancy given by unnecessary concepts, entities and relationships that are required only at later building lifecycle stages. Simplicity of representation also facilitates design refinement and exploration during conceptual design.

3.3.1.2 Geometric modeling and reasoning component

During conceptual design the engineer makes decisions regarding the form of the structure by reasoning about its geometry and topology while acknowledging its function

as part of the building architecture (cf. section 2.2.2.2 a). Geometric modeling and reasoning (GM/R) techniques are therefore proposed in this research project as the basic problem-solving methods for assisting engineers during conceptual structural design. The process of reasoning with a model involves analysing and querying the model to extract information and knowledge from the concepts that it represents, either explicitly or implicitly, and using this information to modify and/or refine the model. Therefore, the nature of the reasoning and the querying processes depends on the type of information being extracted and the type of the concepts being represented.

Given that the building architecture and the structural system are physical arrangements of elements, all the decisions and constraints that affect the configuration of the structural system are ultimately transformed into geometry and topology constraints. Consequently, geometric modeling and reasoning techniques play a central role in the configuration of the structural system. Initially, this reasoning involves inspecting three-dimensional (3D) models of the building architecture and the structural system, visualizing continuous support patterns and detecting architectural geometries that may pose structural difficulties. Subsequently, it involves configuring structural models that satisfy the architectural constraints.

a) Geometric modeling kernel

A geometric modeling kernel (GMK) provides low-level geometry (e.g. plane and curve) and topologic (e.g. face and edge) data structures and algorithms (e.g. intersect) that allow the engineer to reason in terms of geometry and topology. Several commercial, academic and open-code geometric modeling kernels were investigated before choosing

ACIS (Spatial Corp. 2004) as the underlying geometric modeler for the software prototype. Section B.6.1 of Appendix B includes a comparative table with the geometric modelers that were investigated in this research. The main criteria for evaluating the geometric modelers were the following:

1. It should integrate mixed-dimensional geometry so that 1D, 2D and 3D entities can be combined in a geometrical model and operations can be performed combining entities of any dimensionality.
2. It should offer parametric capabilities for enabling changes to the model.
3. It should provide open architecture so that it easily accepts third party add-ins.
4. It should be robust and reliable.
5. It should offer an object-oriented, C++ development environment.
6. It should offer good support and comprehensive documentation.
7. Its price should be affordable.

ACIS was chosen among the kernels that were evaluated because it offers most of the capabilities required for conceptual design. For example, it models mixed-dimensional geometry by integrating wire-frame, surface and solid modeling representations in a unified data structure, which is implemented as a hierarchy of C++ classes. Thus, in order to manipulate mixed-dimensional models ACIS provides powerful operations that combine entities of different dimensionalities (e.g. intersect a 2D wall and a 3D space). Considering the parametric capabilities, it should be noted that, at the time of the selection of ACIS as the underlying geometric modeler for the proposed approach, no commercially available parametric geometric modeler was found in the market. At the time of the investigation, only few proprietary parametric geometric modelers were

available. Being proprietary, these geometric modelers did not provide an open architecture for third-party add-ins since they were part of domain specific applications (for example Pro/ENGINEER by PTC for mechanical engineering and Revit by Autodesk for architectural design). Nowadays, the situation appears to have changed. For future research projects particularly dealing with changes, which are natural during conceptual design, a parametric geometric modeler should be investigated. Being the foundation for several commercial CAD packages (e.g. AutoCAD by Autodesk) ACIS has proven to be a robust and reliable geometric modeler.

b) Synthesis algorithms

These algorithms rely on geometric modeling and reasoning (GM/R) as the core techniques for assisting engineers during conceptual structural design. GM/R techniques are based on geometric (e.g. coplanar, collinear, parallel, etc.) and topologic (e.g. adjacent, overlapping, contained, etc.) relationships and operations (e.g. Boolean union, difference and intersection) among entities in a geometric model to provide feedback about the model and enable its refinement. In order to provide meaningful feedback to the engineer, these techniques need also incorporate knowledge and constraints from the structural domain, which are usually contained in the entities being modeled. Appendix B provides a detailed description of GM/R techniques.

Synthesis algorithms make use of the capabilities provided by the representation of the building architecture and the structural system and those of the geometric modeling kernel to enable geometric modeling and reasoning (GM/R). For example, the algorithms assist the engineer in inspecting the building architecture (e.g. verify the vertical

continuity of walls and columns), in finding supports for structural elements and in verifying gravity and lateral load paths to the ground. In particular, in order to enable geometric reasoning, these algorithms rely on the following features: (1) implicit and explicit relationships among entities in the model being created, including spatial relationships such as intersect, adjacent and overlap, (2) constraints from the entities either already configured within the model or to be integrated into the model that limit their configuration as well as the definition of related entities, and (3) the geometry of the model being created through geometric constraints such as collinear, coplanar, aligned and parallel.

3.3.2 Develop a formal model of conceptual structural design

A formal model of the design process identifies and organizes the different activities that are required for carrying out a design. The purpose in developing a formal model of the process of conceptual structural design is to facilitate collaboration between architects and engineers during conceptual design of building structures by relying on innovative computer technologies. The formal model aims also at encouraging structural engineers to adopt improved ways of designing with computers. The model is "formal" in the sense that it intends to enable the process of conceptual design of building structures to be implemented in computers. The model is representative of conceptual structural design as it incorporates the most relevant aspects of this design stage as reviewed in section 2.2.

3.3.3 Design assistance environment for conceptual structural design

This stage consists basically of the detailed planning and organization of the two main components that are required by a computer environment aiming to support conceptual structural design, namely: (1) a representation of the structural system and the building architecture, and (2) a geometric modeling and reasoning component (cf. section 3.3.1). The planning of these two components is directed towards supporting the activities specified by the formal model of conceptual structural design (cf. section 3.3.2). The organization of the representation follows standard object-oriented principles, where objects (i.e. entities) encapsulate attributes and methods to specify how each entity should interact with other entities in order to integrate properly within the design model. In order to make the representation simpler and extensible, a decision has been made to minimize explicit relationships among entities so that whenever possible these relationships can be derived from the model through synthesis algorithms. This implies a trade-off between hard-coding relationships in the representation and developing more complex algorithms.

3.3.4 Prototype implementation

Three alternative strategies are devised for implementing the proof-of-concept software prototype: (1) directly on top of a generic, general purpose CAD platform, (2) on the basis of a CAD platform specifically designed to support building modeling, and (3) from scratch, directly on top of a geometric modeler. Time wise, the first two options appear more convenient since CAD and building modeling platforms already provide an intermediate layer between the geometric modeler and the developer. Furthermore, a

building modeling platform provides yet a higher semantic layer. However, the more capabilities are provided by those platforms, the more limiting they become to support the ideas proposed in this research project. This is because CAD or a modeling platform provides only the option to adjust to its own data representation. For option (1) AutoCAD by Autodesk and Microstation by Bentley (Table A.3) were evaluated, while for option (2) Autodesk Architectural Desktop and Revit (Table A.2) were evaluated. As expected, neither of those alternatives allowed developing a computer representation consisting of entities and relationships other than the ones that the platform incorporates. Architectural Desktop and Revit offer proprietary representations of the building architecture. Revit is a parametric building modeler that provides explicit descriptions of spaces and physical entities. However, it provides no open architecture for third-party add-ins. Architectural Desktop provides open architecture but its description of the building architecture is rather limited; for example, it does not provide explicit description of architectural spaces. In addition, in spite of being developed on top of powerful geometric modeling kernels, neither of the above platforms allows the engineer to model mixed-dimensional geometries as required during conceptual design (cf. section 2.1.5 and 2.4.2.5). Alternative (3) has therefore been chosen. Even though it demands longer development time, it also gives more freedom for developing a tailor-made object-oriented application.

3.3.5 Testing and validation

Several alternatives have been contemplated for testing and validation of the software prototype, namely: (1) have architects and engineers concurrently test the software prototype during conceptual design for a real building, (2) have architects and engineers

independently test the prototype with a project specifically conceived for the test, and (3) simulate the work of architects and engineers using existing projects as case studies. Alternative (1) was discarded because it would require the development of a sophisticated graphical-user-interface (GUI) as well as the implementation of networking mechanisms for having architects and engineers work concurrently. In addition, it would be difficult to find an architect and an engineer that would agree to test a software prototype with a real project and in actual working conditions, given the time commitment. Alternative (2) was more feasible than alternative one, however it still required developing a sophisticated GUI, which is not the goal of this research project. Finally, alternative (3) was selected.

The following Chapters present the development and results from each of the stages of the project development described above. Thus, Chapter 4 describes the formal model for conceptual structural design that corresponds to section 3.2, Chapter 5 presents the design assistance environment for conceptual structural design corresponding to section 3.3, Chapter 6 describes the proof-of-concept software prototype that elaborates section 3.4, and Chapter 7 presents the testing and validation of the approach proposed in this research project and corresponds to the stage described in section 3.5.

CHAPTER 4

FORMAL MODEL OF CONCEPTUAL STRUCTURAL DESIGN

4.1 Introduction

Over the years, there have been numerous attempts to provide formal models of the design process of different types of artifacts. These models represent the pattern of activities that take place during design. In general, two types of models have been identified (Cross 1989): (1) descriptive models that describe the sequence of activities that typically occur during the design process, and (2) prescriptive models that attempt to prescribe better or more appropriate patterns of activities. Formal models do not limit creativity, but they are complementary to creative design methods. Therefore, rather than "straightjacket", formal models should be regarded as "lifejacket" that help designers to keep afloat (Cross 1989).

A formal model of the design process must closely map the reasoning patterns that designers follow while performing design. These patterns are described in section 2.2 and summarized below as follows:

- The design of a structural system, particularly during the conceptual stage, is the result of exchanges leading to an agreement, mainly between the architect and the structural engineer.

- During conceptual design, the engineer deals with different levels of problem decomposition to facilitate the reasoning process. The hierarchical organization of

the structural system results from such a problem decomposition. Each level of structural problem decomposition provides opportunities for architecture/structure integration.

- The emphasis during conceptual structural design is on the synthesis (i.e. form and function) of structural solutions. Consequently, the engineer makes decisions regarding the form of the structure by reasoning about its geometry and topology, while acknowledging its function. These decisions are based mostly on knowledge about the suitability (i.e. behavior, efficiency, cost, constructability, etc.) of structural systems and materials for a given design situation.
- Depending on the size and complexity of the building, and on the experience of the engineer, simplified analysis may also be required. Structural analysis at the conceptual stage enables more informed decision-making by allowing structural behavior to be accounted for in the early structural evaluations.

Several descriptive and prescriptive models of the design process have been proposed for engineering and for architecture. An interesting feature that distinguishes engineering models from architectural models is that the latter place more emphasis on creativity, while the former place more emphasis on analysis. Cross (1989) proposes a generic model that abstracts the main activities from most engineering models, which is applicable to both engineering and architectural design (see Figure 4.1). In this model designing proceeds by decomposing the problem into sub-problems and oscillating between sub-problems and sub-solutions, which are then combined in order to produce the overall solution. The overall solution is evaluated and compared against the

requirements of the overall problem. Then, the problem is clarified and may be redefined, which requires the process to be repeated again.

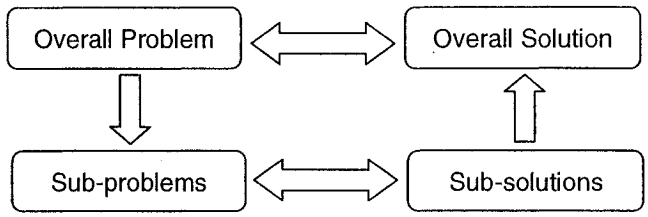


Figure 4.1 Generic model of the design process (Cross 1989)

This research project proposes a formal model for the conceptual design of building structures that is based on the generic model proposed by Cross. The model is prescriptive since it aims at encouraging structural engineers to adopt improved ways of working with computers. The model follows a top-down design hierarchical refinement approach for structural problem decomposition during the synthesis process that closely matches the way structural engineers reason while synthesizing structural systems. As discussed in section 2.2.2, this type of reasoning is not presently being supported by computers. The top-down approach does not limit the designer's freedom since it enables higher level entities to be quickly defined, thus allowing the designer to efficiently move between sub-problems and sub-solutions within the context provided by those higher level entities.

During conceptual design, the engineer's work flow is highly dependent on the amount, quality, and type of information that is exchanged with the architect (cf. section 2.1.1.2). This in turn affects the quality and effectiveness of the structural engineering feedback provided to the architect. A formal model of conceptual structural design must acknowledge such interactions between architects and engineers. Section 4.2 proposes a simplified view of the interactions that take place between architects and engineers

during early design. This view constitutes the basis for developing the formal model of conceptual structural design.

4.2 Simplified interactions between architects and engineers

The goal of this research project is to facilitate collaboration between architects and engineers during early stages of the design process (cf. section 1.2). Since the building architecture and the structural system are tightly coupled, every decision that is made by the architect during early design has a direct impact on the structural system (cf. section 2.2.3). Therefore, from a structural engineering standpoint collaboration between architects and engineers should begin as early as possible during the architectural design process. This collaboration can take place either concurrently (i.e. in parallel) or sequentially.

Concurrent design means that the architect and the engineer work together to develop the architectural concept from the outset. While sequential design means that the engineer begins participating in the building design process once the conceptual stage of architectural design is over. In the former case, the architect and the engineer select together construction materials and explore architectural design alternatives. In the latter case, the architect alone explores architectural design alternatives and chooses the most appropriate which after some refinement is communicated to the structural engineer; then while the engineer performs structural design, the architect continues refining the selected alternative. From a structural engineering standpoint concurrent design leads to more optimum results while sequential design is inefficient (Bachmann 2003).

Architects often prefer to develop their design concepts privately (cf. section 2.2.1.2). Therefore, following the main premise of this research project (cf. section 3.1) supporting strictly concurrent conceptual structural design with computers is questionable since the goal, during early building design, is not to interfere with the creative workflow of the architect. An intermediate approach is therefore proposed in this research project. It mediates between both parties' needs and work styles. It is a sequential approach where early architectural alternatives are generated by the architect privately and communicated to the engineer for quick structural configuration. In this way, the structural engineer can validate these architectural alternatives, discuss the structural implications with the architect and show how they can integrate with the structure during the exploration of the architectural design.

First and foremost, the quality and type of structural feedback depends on the quality and type of information provided by the architect. The architect can communicate with the engineer using simple qualitative descriptions (e.g. building type and architectural preferences) and quantitative building parameters (e.g. building area and number of stories), as well as sketches and/or a computer (i.e. digital) model of the building architecture (cf. section 2.3.1.1). The more informal, ambiguous and vague the architectural descriptions are, the less accurate the responses should be expected from the engineer. The main challenge in devising methods for supporting conceptual design of building structures comes from the need to come up with structural solutions that correspond to the architectural descriptions as they are made available to the engineer. This issue is examined in the remainder of this chapter.

4.3 Structural synthesis in response to the early building architectural design process

From the discussion in section 4.2 two stages of the early architectural design process can be differentiated: a first stage where the architectural representations are informal and vague (i.e. parametric descriptions and possibly sketches), and a second stage where a formal computer representation of the building architecture is produced and made available to the engineer (i.e. a digital model of the architecture). For both stages general building information and architect's qualitative preferences are also available. For computer support, a distinction needs to be made between two sub-processes that are carried out by the engineer in response to the abovementioned stages of refinement of architectural design, namely: (1) structural layout planning and (2) architecture/structure (A/S) integration. This distinction is required because the input, the processing, the reasoning, and the output of these two sub-processes are different. Therefore, computer requirements are also expected to be different.

4.3.1 Structural layout planning

The process of structural layout planning begins after structural problem definition (cf. section 2.2.2.1 a). Consequently, to carry out this process engineers rely on information about the building that has been defined during the structural problem definition stage plus additional information communicated by the architect either verbally or through sketches. This information includes the following: location and type of building, approximate number of stories, approximate area of each storey, shape of the land, and client's budget. The architect's personal preferences are also communicated to the

engineer (e.g. concerning comfort and aesthetics, as well as the selection of structural and non-structural materials).

Given that at this point no actual architectural design or only a rough one may be available to the engineer, the engineer has the flexibility to explore multiple structural alternatives and, from the information available, propose optimum structural solutions to the architect and the client. Thus, order of magnitude parameters from the architecture (e.g. approximate number of stories, floor spans and construction area) can be used by engineers to perform parametric studies by comparing structural system types, materials, layouts, costs and weights. In order to evaluate different alternatives, the engineer needs to define feasible and "generic" structural system layouts (including initial sizing of structural elements). These layouts are "generic" in the sense that they are not meant to fit accurately within a model of the building architecture. The feasibility of these structural layouts may be provided either by performing simplified structural analysis or by using artificial intelligence (AI) techniques that rely for example on heuristic thresholds for selecting and sizing elements. Feedback to the architect is provided mostly in terms of costs, materials, feasible structural system types and their layouts and tentative element dimensions. At this stage, the engineer can also advise the architect on the structural aspects involved in the selection of the types and materials of non-structural partition walls and façade elements (cf. section 2.2.3.4).

4.3.2 Architecture/structure (A/S) integration

Once a digital model of the building architecture is made available to the engineer, he/she can integrate the structural system to it. That is, the engineer can configure a

structural system that more accurately responds to the architectural design. This implies that more detailed feedback can be provided to the architect regarding locations and dimensions of potential structural elements. For simple building architectures, the results from structural planning can be mapped and integrated directly into the building architectural design. However, this is not the case for more complex buildings as built today. A/S integration is the main subject of this research project. The following four activities have been identified to be carried out by the engineer while integrating the structural system to the building architecture:

(1) ***Inspection of the building architecture*** - During this activity the structural engineer explores the architectural model and detects potential structural problems (e.g. abrupt changes in the building form), identifies structural opportunities (e.g. architectural elements that may become structural) and uncovers structural layout constraints at the global and local levels (e.g. column-free spaces). It is during this activity that the engineer first captures the intents of the owner and the architect implicit in the design. After the inspection activity it is expected that the architect will make changes that will solve or at least attenuate potential structural problems, validate structural opportunities, and relax structural layout constraints, if possible.

(2) ***Configuration of the structural system*** – The engineer configures the structural system within the context of the building architecture. As described in section 2.2.2.3 (Figure 4.2), following a top-down approach the engineer defines structural massings first (i.e. independent structural volumes and structural zones), followed by structural subsystems (horizontal, vertical gravity and vertical lateral subsystems), and the configuration of structural assemblies and structural elements.

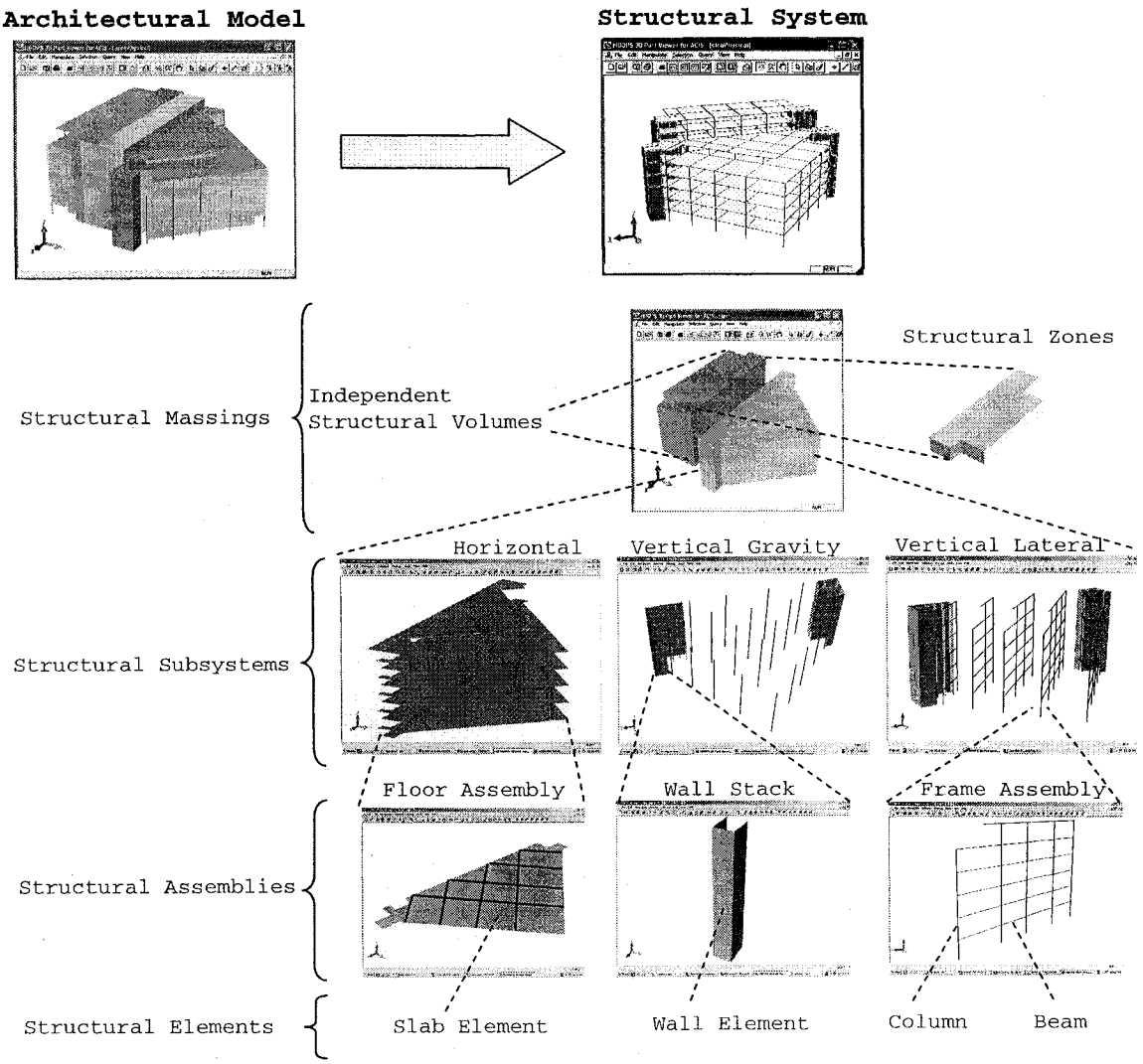


Figure 4.2 Structural system configuration following a top-down approach

(3) Verification of the integrated model – This verification involves confirming that the integration has been successful and that the structure is sound and stable. At the conceptual stage structural "soundness" means providing continuous, i.e. un-interrupted gravity and lateral load paths to the ground, and structural stability means that the support system is sufficient for resisting gravity and lateral loads. At this stage, no time-consuming stability analysis is performed but only initial verifications regarding the type, completeness and proper distribution of supports.

(4) ***Evaluation*** – A structural solution is evaluated using methods that could include not only structural factors, but also qualitative architectural factors that reflect how the structural system respects architectural constraints and responds to the architect's intent. For example, if one alternative requires columns to be placed within spaces it should be penalized. However, if another alternative does not require such columns, but due to longer spans demand thicker slabs, it may be penalized structurally but not architecturally, as long as the slab thickness is within the margins accepted by the architect and does not interfere with any key architectural features (e.g. reduced window heights due to thicker ceilings). Simplified analysis could also be carried out at this stage if necessary to validate structural solutions. As discussed in Chapter 1, this research project includes neither evaluation nor analysis of structural solutions.

A model for A/S integration must consider a) the potential problems coming from the architecture, b) the structural opportunities, and c) the structural layout constraints. Considering the potential structural problems coming from the architecture, the most visible ones are expected to be detected during the initial inspection of the architecture. These problems usually come from irregularities of the building form or irregularities in the geometry and layout of spaces and structurally relevant SEEs (cf. section 2.2.3). If after inspection, the architect decides not to take any action to help to solve those problems, the engineer may have to "deal" with them and wait until the configuration, verification and evaluation activities to quantify them and check how serious they are (while possibly detecting new problems).

The structural opportunities and constraints coming from the architecture are mostly considered during the structural configuration activity. Depending on the view point,

architectural constraints can be considered potential structural opportunities since they permit the engineer to explore alternative solutions to satisfy those constraints. In most cases however, they impose support locations that reduce the alternatives available to the engineer. The structural system and the building architecture are tightly coupled. As a consequence, once an architectural design is made available to the engineer, numerous constraints follow that need to be considered in proposing alternative structural system layouts such as: function, behavior, performance, reliability, material, cost, compatibility and constructability constraints (Luth et al. 1991). The building architecture imposes stringent constraints to the configuration activity that can be classified either as functional (i.e. related to the use of spaces) or physical (i.e. coming from the actual building construction elements, such as walls). These constraints in turn lead to corresponding functional and physical types of integration between the structural system and the building architecture.

4.3.2.1 Functional integration – functional constraints

As discussed in section 2.2.3.2, the functionality of spaces imposes constraints on the configuration and the type of structural system to be used, namely: expected load patterns, amount and distribution of vertical supports, allowable floor spans, and allowable ceiling heights. Structural zones group spaces with the same or similar functionality that pose uniform structural requirements. Therefore, structural zones are key structural entities for transferring architectural functional constraints to the engineer and enabling functional integration. From the literature (Arnold and Reitherman 1982,

Meyer 1995, Schodek 2003) functional integration can take place at two levels as follows:

Level 1 Integration within structural zones (intra-zone) – This level of integration follows from the fact that the structural system must be appropriate to the function it is to shelter. For example, a linear function demands a linear structure, and therefore it would be improper to roof a bowling alley with a dome (Shaeffer 1998). Structural zones translate the functionality of the spaces that they aggregate into structural requirements and constraints for the layout of structural solutions.

Level 2 Integration among adjacent structural zones (inter-zone) – In this level, the patterns of adjacent structural zones are matched so that transfer structures are avoided whenever possible. For example, Meyer (1995) developed a grammar for generating matching patterns among dissimilar structural zones on different floors (cf. section 2.4.1.6). In addition, the relative location between structural zones is assessed for example to avoid heavy zones that may be prone to vibrations to be placed on top of quiet and lightweight zones.

The formal model of conceptual structural design that is proposed in this research project enables functional integration between the structural system and the building architecture.

4.3.2.2 Physical integration – physical constraints

As described in section 2.2.3.4, there is a strong interdependency between the layout of structural elements and the patterns defined by the construction elements in the building architecture. An engineer's goal, in relation to the architecture, is therefore to match as closely as possible structural patterns to those defined by the construction elements,

mainly the permanent walls. The degree of architectural constraining imposed on the configuration of structural solutions depends on whether the building is single-cell or multiple-cell (cf. section 2.2.3.4 a). Consequently, architectural physical constraints can be classified in two groups:

- ***Internal constraints*** that restrict the internal geometry and topology of the structural system.
- ***External constraints*** that affect only the external shape of the structural system.

For single-cell buildings, internal constraints are minimized and sometimes eliminated. Thus, the external constraints prevail. For multiple-cell buildings, both internal and external constraints are relevant. Therefore, single-cell buildings, as well as large building zones (e.g. an indoor swimming pool or the conference room in a building) are suitable for the exploration of alternative structural configurations. By contrast, multiple-cell buildings pose greater limitations for proposing alternative structural configurations.

By observing actual buildings and studying the patterns formed by the elements of the vertical support system and those of the space establishing elements (cf. section 2.2.1), four levels of physical integration are identified below and described using set theory as follows:

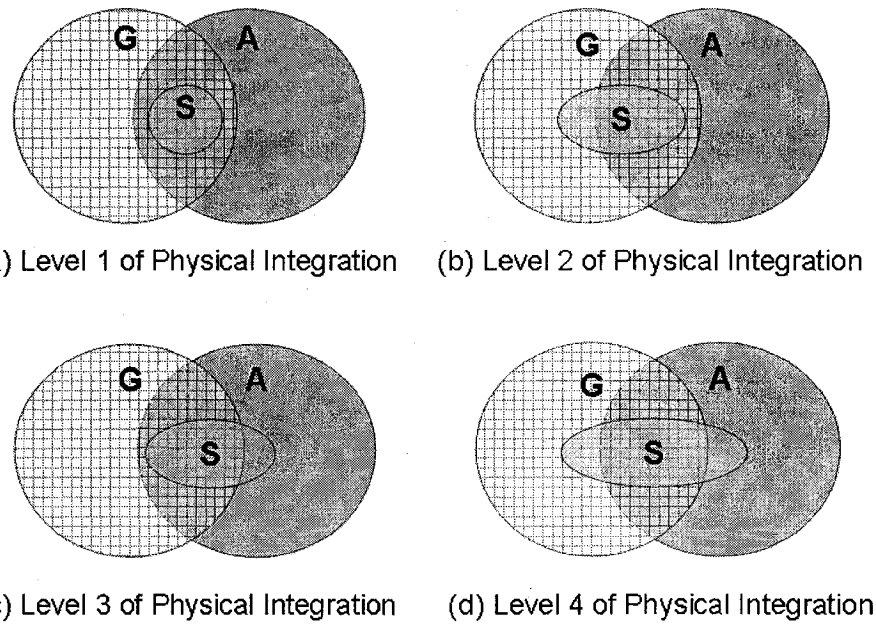
Set G: represents the patterns formed by the project grids.

Set A: represents the patterns formed by the space establishing elements.

Set S: represents the patterns formed by the vertical structural elements.

Level 1 (Figure 4.3a) - This is an ideal case as it represents the best level of physical integration because the structural patterns are fully contained within the architectural

patterns, so that all structural members are either hidden within architectural elements or intentionally visible, with no unwanted structural members lying in unwanted locations. Note that structural elements that are intentionally visible are also architectural elements since they are part of the architectural concept. Architectural elements that do not adjust to the common patterns between the architecture and the structure are not structurally relevant.



(a) Level 1 of Physical Integration (b) Level 2 of Physical Integration

(c) Level 3 of Physical Integration (d) Level 4 of Physical Integration

Figure 4.3 Physical integration between the vertical sub-system and SEEs

Level 2 (Figure 4.3b) - In this level structural elements still adjust to the common patterns between the architecture and the structure. However, due to structural dimensional constraints the engineer must place columns inside spaces, i.e. outside the set of space establishing elements.

Level 3 (Figure 4.3c) - In this level, some architectural elements that fall outside the common patterns are structurally relevant, thus producing local "misfits" (cf. section

2.2.3.4.b). Therefore, the engineer provides a special local framing lying outside the common patterns that integrates such architectural elements to the structural system.

Level 4 (Figure 4.3d) - This is the most general case where most structural elements match the common architecture-structure patterns. However on the one hand, some structural elements inevitably fall inside spaces. On the other hand, due to local architectural conditions structural elements also fall outside the common architecture-structure patterns thus requiring a special local framing.

In general, the engineer seeks to accommodate the structural system to common architecture-structure patterns, unless particular architectural conditions exist that force the structure to fall outside these patterns. These conditions, as described in levels 3 and 4, are the most difficult to support with computers since they demand special local structural solutions. Levels 1 and 2 of physical integration correspond to the "degrees-of-fit" as described in section 2.2.3.4.b by Schodek (2003). Considering the AI-based research projects from the literature review in section 2.4.1, most of them provide support for levels 1 and 2, but for rectangular office buildings only, while the semantically explicit representations proposed by Björk (1992), Dias (1996) and Khemlani et al. (1998) support level 1 only. The formal model of conceptual structural design that is proposed in this research project enables functional A/S integration, as well as all four levels of physical A/S integration.

4.3.3 Architecturally constrained structural synthesis

Sections 4.3.1 and 4.3.2 identified and classified the architectural constraints that restrict the architecture/structure integration as part of the synthesis process. Based on those

architectural constraints, the problem faced by engineers during structural synthesis can be decomposed into specific activities as illustrated in Figure 4.4. For the structural layout planning activity architectural constraints are minimized because at this point only a rough architectural design may be given to the engineer. This gives some room for the engineer to explore different structural alternatives.

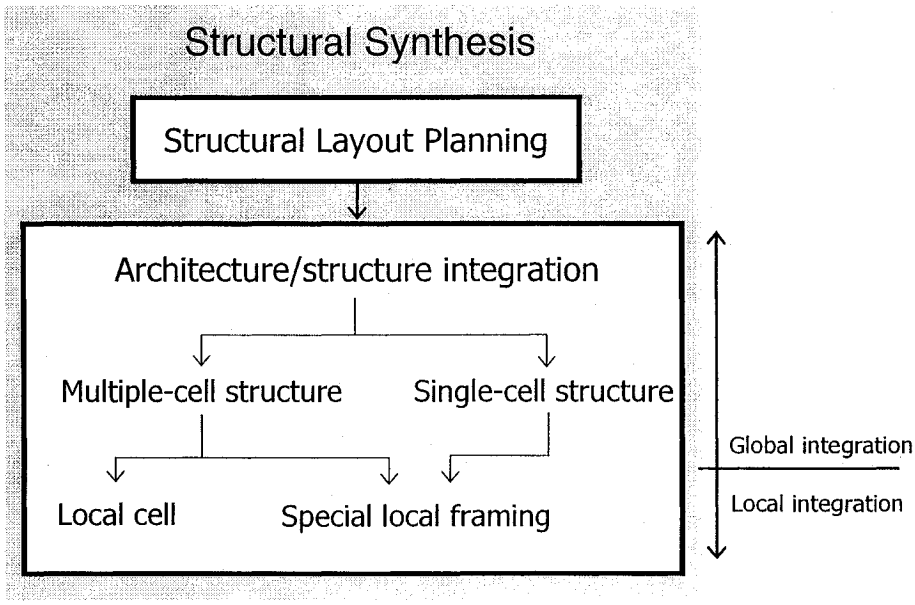


Figure 4.4 Architecturally constrained structural synthesis

The list of promising alternatives is considerably pruned when the engineer is presented with a digital model of the building architecture. Starting with this, the engineer creates the configuration of the structural system for the entire building (i.e. global integration). The reduced amount of internal architectural constraints simplifies this activity for single-cell structures as compared to multiple-cell structures. For multiple-cell structures, specific structural solutions are required for large building zones (i.e. local-cell integration). In both cases (i.e. for multiple-cell and single-cell structures), it is likely that special local framing solutions will also be required to respond to structurally relevant

architectural configurations that lay outside the common grid. In response to building architectural constraints, each of the activities indicated in Figure 4.4 demands a particular type of support from computers. Section 4.3.4 identifies the computer technologies that are suitable for supporting these activities.

4.3.4 Computer technologies for structural synthesis

The use of a specific computer technology depends mainly on the type and stringency of architectural constraints involved. For some of the activities indicated in Figure 4.4 several computer technologies have already been investigated. Figure 4.5 shows the same activities as illustrated in Figure 4.4 with related computer technologies below each activity, within parenthesis.

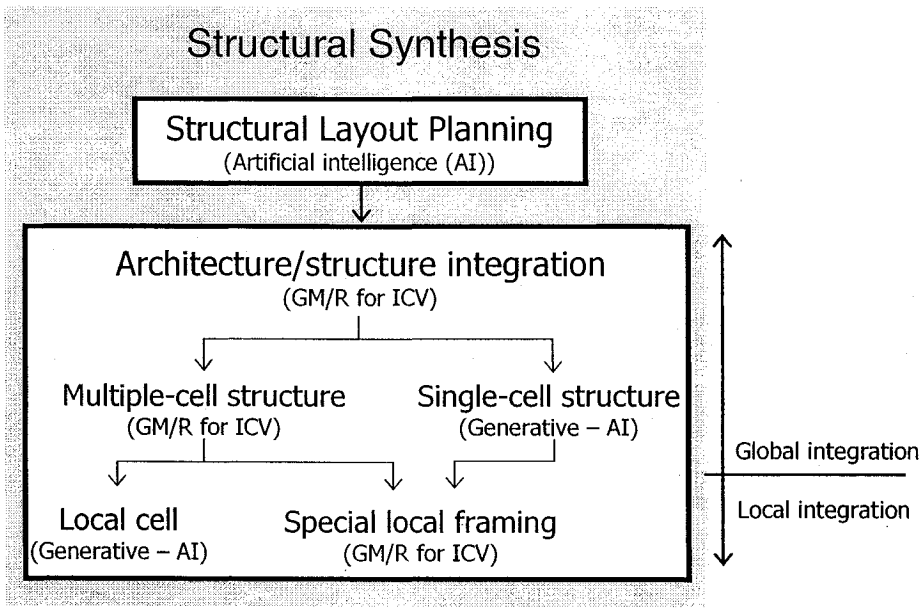


Figure 4.5 Structural synthesis activities and suitable computer technologies

Structural layout planning – Most previous research in conceptual design of building structures provide support for this activity (cf. section 2.4.1). This support is mainly

based on artificial intelligence (AI) techniques, which assist engineers in exploring several alternatives under constraints. Given that at this point no digital model of the architectural design is yet made available to the engineer, architectural constraints are minimized and can be handled and stored as parameters in a table or a relational database.

Architecture/structure integration – The presence of architectural functional and physical constraints precludes the use of AI techniques as the main mechanism for assisting this process. Therefore, this research project proposes using geometric modeling and reasoning (cf. section 3.3.1.2) for the inspection, configuration and verification (GM/R for ICV) of a design model combining the building architecture and the structural system. Geometric modeling and reasoning techniques have still untapped potentials for assisting engineers in uncovering structural problems and opportunities in the building architecture, configuring the structural system as part of the architecture, and verifying structural configurations. Geometric modeling and reasoning techniques can also be enhanced with knowledge-based AI techniques for improved computer assistance to the engineer (e.g. in material selection, element pre-dimensioning and feasible alternative generation).

Configuration for single-cell structures and local-cells – For such cases, the reduced amount of internal physical constraints makes generative techniques suitable to automatically or semi-automatically produce structural system layouts. Typical applications are found in the configuration of floor, roof and truss systems. Several techniques, some of which are AI techniques, have been investigated in section 2.4.1.6.

Special local framing – Special local framing solutions are required to respond to particular architectural design situations. For such specific design decisions, it is likely

that the engineer will have to interact directly with the design model locally to devise particular framing solutions. Geometric modeling and reasoning techniques can facilitate the integration (i.e. inspection, configuration and verification) of those special local design solutions to the global structure and the building architecture.

This research project tackles the A/S integration process as a whole. Therefore, it aims at developing the methods and techniques for supporting the entire A/S integration process, instead of focusing on finding solutions for specific sub-problems (e.g. single-cell structures, local-cell structures or special local framings).

4.4 Formal model of the conceptual structural design process

This model aims at representing a close collaboration between architects and engineers from the earliest stages of building design. Consequently, it includes activities that are performed by the architect and corresponding activities performed by the engineer. The model is illustrated in Figures 4.6 and 4.7. The left part of the diagrams presents activities that are performed by the architect while the right part gives activities that are performed by the structural engineer. In both diagrams, arrows indicate the sequence between activities, except for discontinuous horizontal arrows that indicate one-way and two-way feedback between the architect and the engineer. As illustrated in Figures 4.6 and 4.7, the process follows two stages of refinement, indicated as I and II.

Stage I - Corresponds to the earliest phases of the architectural design process in which the architectural representations are informal and vague. During this stage, considerable changes are still expected to the geometry and topology of the building architecture (e.g. changes to building shape and space layout).

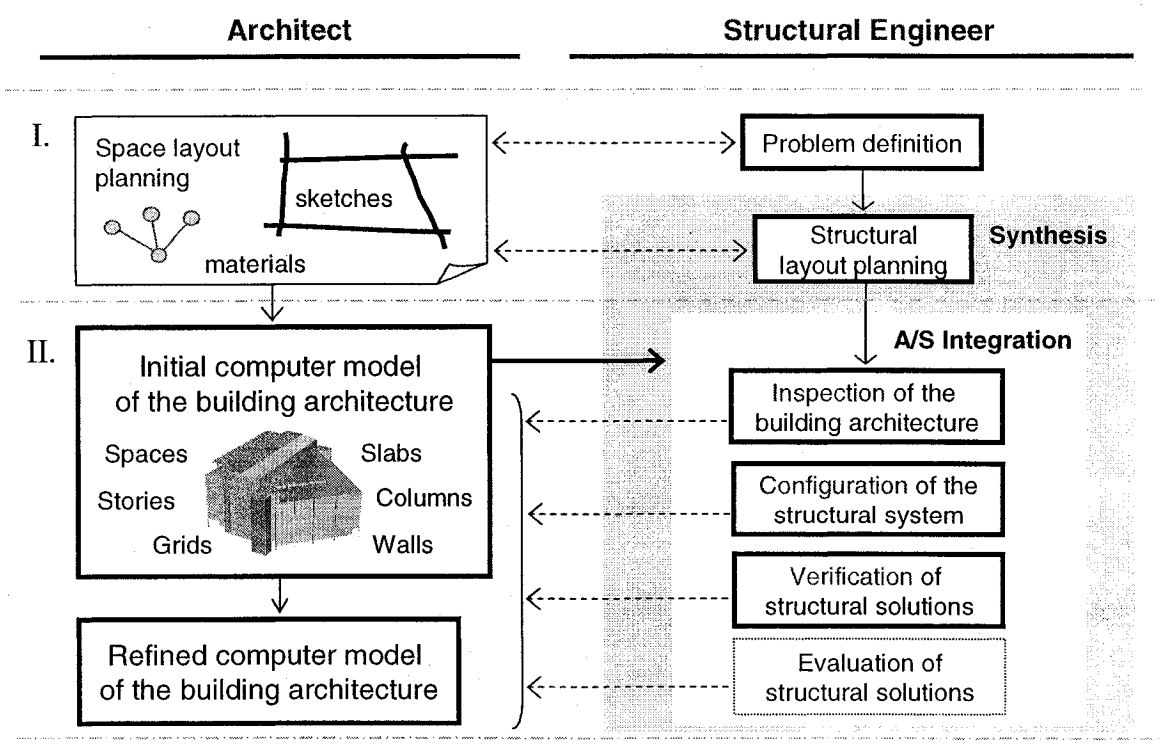


Figure 4.6 Formal model of the conceptual structural design process

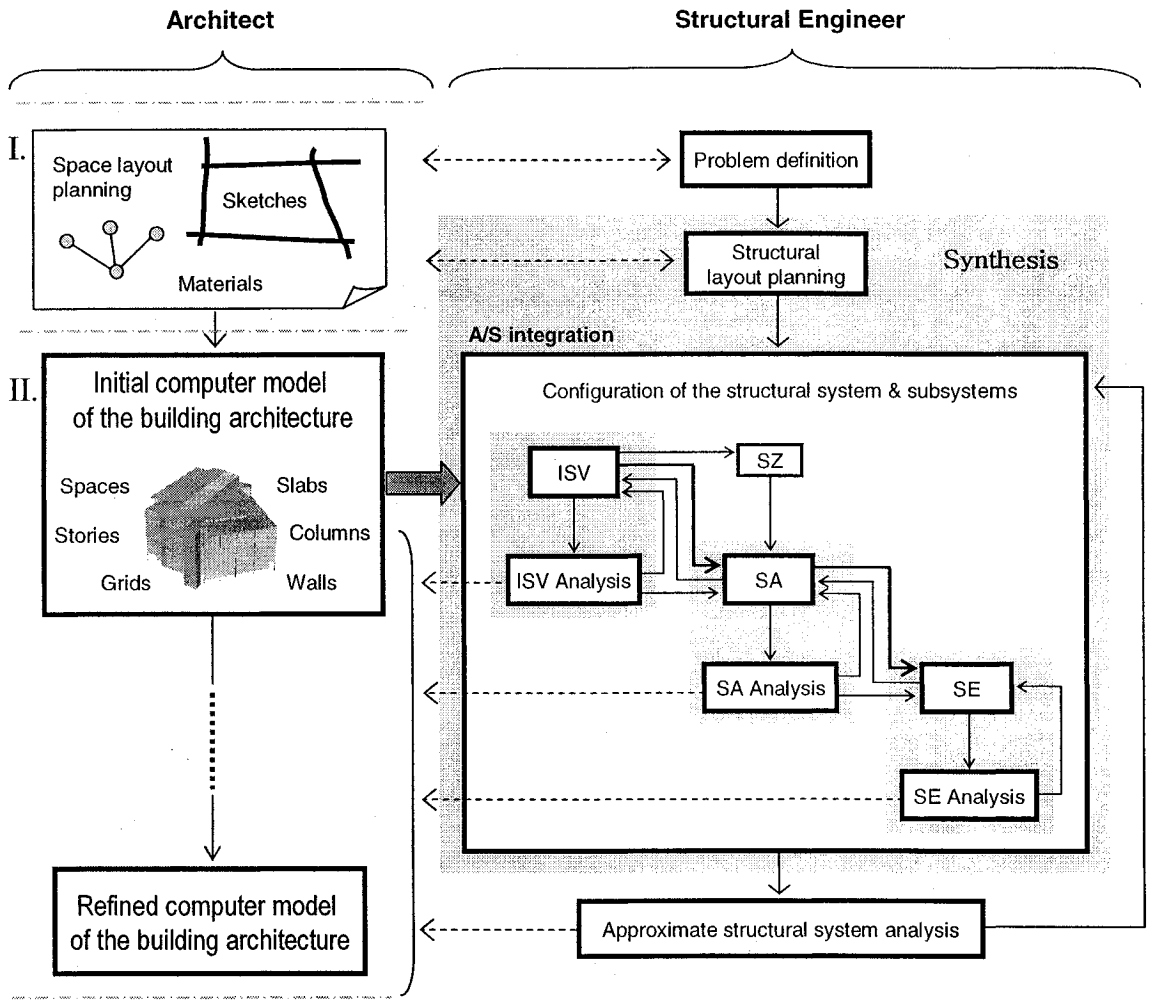
Structural problem definition and structural layout planning activities take place at this stage. To perform these activities the engineer relies on architectural parametric descriptions (i.e. qualitative and quantitative) and possibly sketches. As discussed in section 2.2.1.1.b), sketches are tentative, rough, ambiguous and imprecise, and are meant to represent only the salient features of a design. In addition, sketch dimensions are not accurate and repetition (cf. section 2.2.3.4.a) is not evident. Therefore, in order to provide computer assistance during conceptual structural design with architectural sketches an intermediate task of sketch interpretation is required, which involves inferring intended patterns from sketches. For example, lines, that are meant to be aligned, continuous, parallel or perpendicular by the architect, are not strictly shown like that in a sketch. Once a sketch has been interpreted, the type of structural engineering feedback that can

be provided to the architect has already been described in section 4.3.1. The task of sketch interpretation is out of scope for this research project.

Stage II – At this stage, a digital model of the building architecture is made available to the engineer. During this phase, and after the engineer's inspection of the architecture, less drastic changes are expected to the geometry (e.g. move a wall, enlarge a space, etc.) and the topology of the building architecture (e.g. remove a wall). Consequently, it is during the second stage that the configuration of the structural system and its integration to the building architecture can take place. As discussed in section 4.3.2, structural configuration follows a synthesis by problem decomposition approach where the engineer defines independent structural volumes and structural zones first, followed by structural subsystems, assemblies and then elements, which are connected together through structural connections. Thus, by making use of a computer model of the building architecture, the engineer can provide feedback during or after inspection, configuration and verification, using his/her own knowledge about structural systems and materials. At the end of the process of conceptual structural design, a refined computer model of the building architecture is obtained as a result of engineering feedback to the architect.

Alternatively, as illustrated in Figure 4.7, at each level of the structural decomposition simplified analysis may be performed on demand. In the diagram the thicker arrows, going from independent structural volumes (ISV) to structural assemblies (SA) and structural elements (SE), show a direct path that may be followed by the engineer if no analysis is required at each level. However, depending on the complexity of the problem at hand and on the experience of the engineer, he/she may need to perform simple analysis at each level. After this analysis, the engineer may either make adjustments to

the model at the current level or continue to the next level. In any case, as a consequence from the analysis, informed feedback is likely to be provided to the architect at each level of the problem decomposition. Therefore, in this model, the synthesis process may actually include simplified analysis at different levels of refinement.



Abbreviations:

ISV: Independent Structural Volume
SZ: Structural Zone
SA: Structural Assembly
SE: Structural Element

Figure 4.7 Formal model of conceptual structural design with analysis

The diagram also shows arrows going back from SE to SA and to ISV, which means that there is always a possibility for backtracking. From Figure 4.7, it should also be noted

that structural zones (SZ) are defined by the engineer to constrain the configuration of SAs. Finally, the diagram shows that, even though the structural subsystems are not configured explicitly by the engineer, they are implicitly considered when defining ISVs, SAs, and SEs. Once all SEs have been defined, the entire structural system can be analyzed using the most appropriate analysis technique. As a result from this analysis, additional feedback to the architect may be provided and further refinements to the configuration of the structural system may be carried out.

As mentioned in section 4.3.2, this research project focuses on the integration of the structural system to the building architecture. This process is carried out during the second stage of conceptual structural design as presented in Figures 4.6 and 4.7. The next section presents a methodology for A/S integration as part of the formal model of conceptual structural design.

4.4.1 Methodology for computer-assisted A/S integration

The methodology includes the first three of the four general activities presented in section 4.3.2, namely: (1) inspection of the building architecture, (2) configuration of the structural system, and (3) verification of the integrated model. Activity number four (evaluation) is out of scope for this research project. The methodology is illustrated in Figure 4.8. The first activity is the inspection of the building architectural model as described in section 4.3.2. Then, the configuration activity follows a synthesis-by-decomposition approach (cf. section 4.3.2). In Figure 4.8, an arrow pointing up is added between consecutive activities indicating that backtracking should be allowed. Thus, the engineer has the flexibility to test decisions at lower levels of the structural hierarchy and

backtrack, if necessary, to make corresponding changes at higher hierarchical levels. At the end, verification of integrity and stability of the structural system takes place as described in section 4.3.2. The star at the end of activities number 1, 4 and 5 indicates that simplified analysis can take place during these activities.

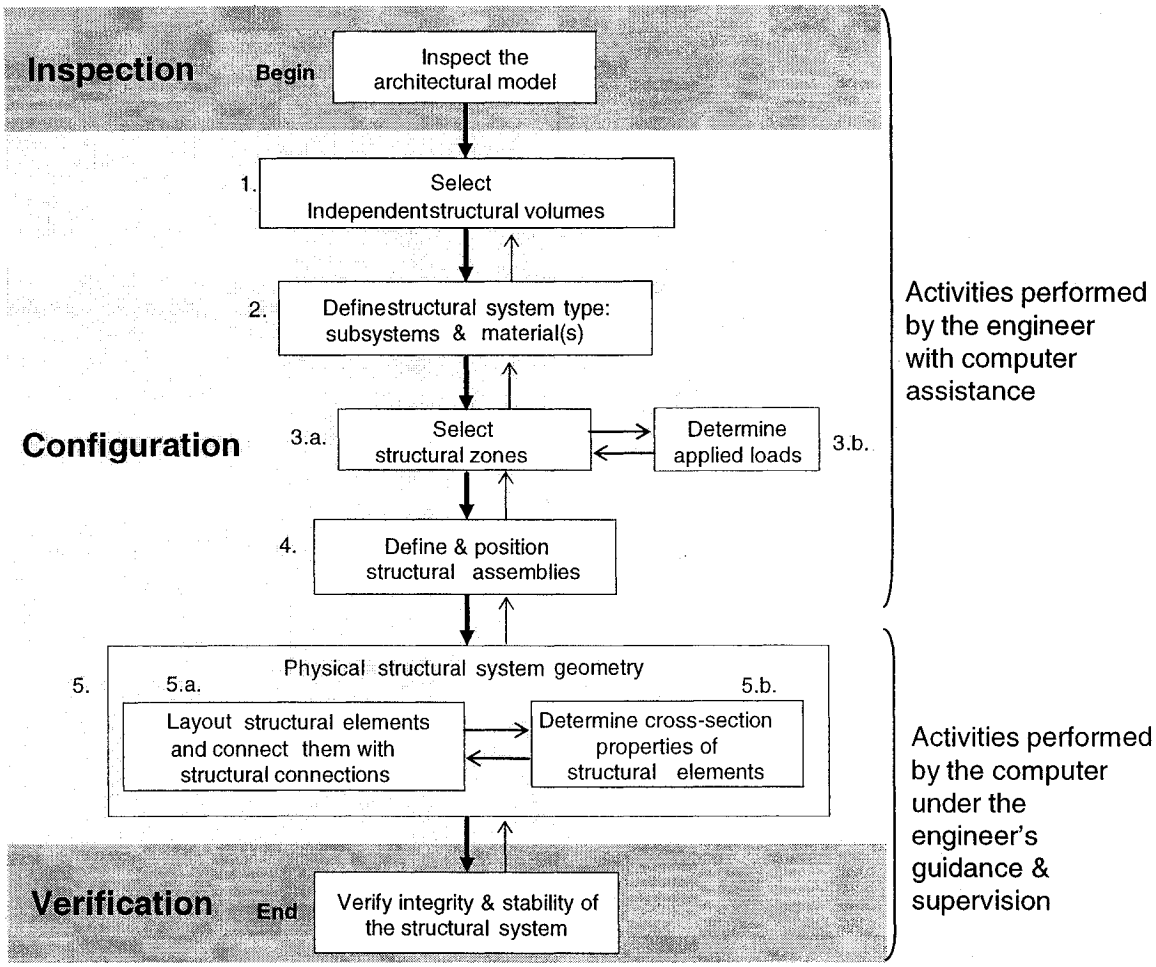


Figure 4.8 Methodology for A/S integration during conceptual design

As indicated in Figure 4.8, activities numbered 1, 2, 3 and 4 are performed by the engineer with computer assistance whereas activities numbered 5.a and 5.b, as well as the structural verification activity, are performed by the computer under the guidance and supervision of the engineer. Therefore, the engineer is in charge of making decisions such as selecting structural system type and material(s), as well as subsystems and assemblies,

and positioning them, while the computer takes care of time consuming tasks such as arranging and connecting elements into assemblies, under the engineer's guidance and supervision. Given that the computer takes care of the most time consuming tasks means that the A/S integration process can take place quickly enough so that opportune feedback can be provided to the architect.

Note that this methodology has been proposed in the context of modern buildings with complex geometries. However, the top-down approach does not have to be seen as a "straight-jacket" that forces the engineer to work in a rigid manner. For simple buildings (e.g. having simple geometry and/or having one independent structural volume and one structural zone), the engineer may skip activities number 1, 2 and 3 in Figure 4.8. Then, after the inspection of the architecture the engineer simply defines and positions structural assemblies. Therefore, the computer has to assume that the more abstract structural entities are not required. In this way, the synthesis process for the engineer is reduced to defining and positioning structural assemblies and letting the computer generate the physical structure.

4.6. Conclusions

In this chapter, a formal model of the process of conceptual design of building structures has been developed. The formal model aims at describing the interactions between architects and engineers during conceptual structural design, and acknowledging the different types of constraints imposed by the architect to the configuration of the structural system. Following such interactions, the structural synthesis process has been decomposed into two sub-processes: (1) structural layout planning and (2)

architecture/structure (A/S) integration. During the first sub-process the engineer relies on overall building information and possibly architectural sketches to explore structural alternatives. During the second sub-process the engineer makes use of a digital model of the building architecture to configure the structural system. A/S integration is the main subject of this research project.

It has been discussed how geometric modeling and reasoning can be used as the main computer techniques for assisting engineers during the A/S integration process. A methodology was developed for A/S integration as part of the formal model of the conceptual structural design process. The proposed methodology tackles the entire A/S integration process, as opposed to focus on a particular sub-activity or hierarchical level. This is in contrast to previous research projects (cf. section 2.4.1) for conceptual and preliminary structural design that focused on activities 5.a) and/or 5.b), either at global or local levels. The methodology takes into consideration the architectural constraints identified in section 4.3.2, and fits within the architecturally constrained structural synthesis process defined in section 4.3.3. Architectural functional and physical constraints are initially uncovered when the engineer inspects the building architecture. Then, through structural problem decomposition, the methodology lends itself to the configuration of global and local structural solutions. Independent structural volumes facilitate configuration at the global level, while structural zones facilitate configuration at the local level. Thus, structural zones enable functional integration between the structural system and the building architecture, while becoming the main mechanisms for structural configuration of architectural local cells (cf. section 4.3.3). Physical integration is also supported by activities number 4 and 5 in Figure 4.8. During activity number 4 the

engineer positions structural assemblies that fit within the patterns of the building architecture. During activity number 5 the computer arranges structural elements within the overall layout set by the already defined structural assemblies. Engineering feedback to the architect can be provided at any moment during or after model inspection, configuration and verification. This feedback can be based on engineering knowledge and/or simplified analysis.

Chapter 5 presents a design assistance environment that implements the above methodology for architecture/structure integration during conceptual design of building structures. This environment is intended to facilitate geometric modeling and reasoning for the inspection, configuration and verification of a model integrating the building architecture and the structural system.

CHAPTER 5

DESIGN ASSISTANCE ENVIRONMENT FOR CONCEPTUAL STRUCTURAL DESIGN

5.1 Introduction

Three main components are required for an automated environment that assists conceptual structural design (cf. section 3.3.1), namely: (1) a representation of the structural system and the building architecture, (2) a geometric modeling and reasoning component, and (3) a knowledge-based reasoning component. As mentioned in Chapter 3, only the first two components are considered further in this research project. The planning and organization of these two components is presented in this chapter with the aim at supporting the formal model for conceptual structural design and, in particular, the methodology for architecture/structure integration presented in the section 4.4. The meaning of the different typefaces that are used in this chapter for describing components, entities, algorithms, methods and attributes, is presented at the beginning of this thesis under the title **TYPEFACES**.

5.2 Representation of the structural system as part of the building architecture

Given that the structural system is an integral part of the building architecture, a representation of the structural system during conceptual design must be conceived as an extension of a representation of the building architecture. While such representation (i.e. architectural domain representation) provides the templates for creating digital

architectural models (as described in section 2.3.1.1), a representation of the structural system (i.e. structural domain representation) provides the templates for creating structural models (as described in section 2.1) following the methodology provided in section 4.4.1. A representation that integrates both systems (i.e. the building architecture and the structural system) is called hereafter ***Integrated Representation***.

5.2.1 Overall organization of the *Integrated Representation*

This organization includes a description of the more abstract entities in the representation and the main relations among them, while the details are discussed in section 5.2.2. As discussed earlier in section 3.3.1.1, the aim is to develop a simple representation (i.e. devoid of unnecessary details) that incorporates the most relevant concepts required for conceptual structural design. For example, the description of a wall from the building architecture includes features such as its location in space as well as its geometry, function(s) (i.e. architectural and/or structural) and main material, but does not consider the various layers that it is made of.

The ***Integrated Representation*** is developed following the object-oriented paradigm, which is a formal mechanism for organizing and modeling information in computers. The object-oriented paradigm is centered on the definition and organization of special kinds of entities, which are termed classes. A class incorporates attributes and operations that guarantee completeness of information and proper definition within a system. An instance of a class is an object that contains attribute values that are specific to the problem at hand. Object-oriented programming has become a de-facto standard for

developing computer programs since it promotes extensibility and reusability of programming resources.

This research uses UML (Unified Modeling Language) notation to describe the various classes of the proposed representation along with their interactions (Booch et al. 1999). In UML, classes are represented as rectangles and relationships among classes are represented as lines between the given objects. Three main types of object-oriented relationships are defined in UML: association, aggregation and inheritance. An association is a bidirectional semantic connection between entities. The UML notation for association relationships is a line connecting the associated entities. In order to describe a given type of association, the relationship is given a name and a triangular arrow head that indicates which way to read the name. An aggregation (also called *part-of*) relationship is a specialized form of association in which a whole is related to its parts (e.g. the stories are part of the architectural model of a building). This is the most basic type of abstraction. The UML notation for an aggregation relationship is a line (association) with a diamond next to the aggregate (whole). Inheritance (also called *is-a*, or *kind-of*) allows a more sophisticated form of abstraction: generalization and specialization (e.g. a primary space is a kind of space). Its UML notation is a line with a triangle next to the generalized entity. In addition, the cardinality of a relationship, defines the number of entities that are related to one another (e.g. one-to-one or one-to-many relationships). The following sub-sections describe the architectural domain representation followed by the structural domain representation and their integration.

5.2.1.1 Building architecture domain representation

The architectural domain representation includes purely functional entities as required to model the architect's intentions, as well as physical entities that substantiate them. As shown in Figure 5.1, the **`Architectural Model`** is the root entity of the architectural domain representation.

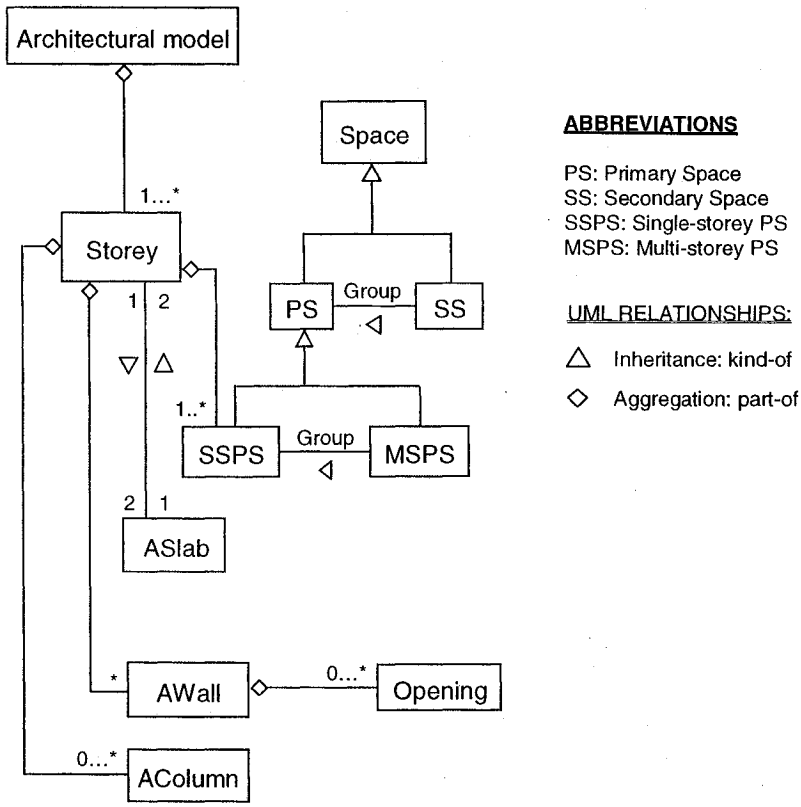


Figure 5.1 Building architectural domain representation

The **`Storey`** is a functional entity that acts as a main organizer of the **`Architectural Model;`** all other entities in the **`Architectural Model`** are directly or indirectly related to the **`Storey`**. **`Space`** is another key concept in the architecture. There are two complementary ways of defining a **`Space`** (Björk 1992): implicitly through its boundaries and/or explicitly as a functional entity. This approach adopts the functional view which is

important for architects during early design. A **Space** is therefore a functional entity that is specialized into **Primary Space** (PS) and **Secondary Space** (SS). A **Primary Space** is the smallest space that can be occupied at any given time. **Secondary Spaces** group **Primary Spaces** that share some functionality (Thiel 1963). For example, they can be used for zoning the building in order to differentiate private zones from public ones. **Primary Spaces** are further specialized into **Single-storey Primary Spaces** (SSPS) spanning a single **Storey** and **Multi-storey Primary Spaces** (MSPS) spanning several **Stories** (e.g. the atrium of a building). **MSPSs** are created by stacking **SSPSs** on top of each other. Staircases and other vertical shafts in the building are modeled as **MSPSs**. This space decomposition allows the later association of structural entities and functions to architectural spaces.

A **Storey** is delimited from above (i.e. ceiling) and below (i.e. floor) by physical entities called **ASlabs** (i.e. the architectural view of the **Slab**). For simplicity, each **Storey** accepts only one **ASlab** as floor and one as ceiling (however in the future, **Stories** with many **ASlabs** at varying elevations are expected to be modeled). **Stories** also have **AWall** and **AColumn** entities. These are physical entities that correspond to architectural views of **Walls** and **Columns**. **ASlabs**, **AWalls** and **AColumns** are defined by the architect without necessarily being concerned about structural aspects. **AWalls** encapsulate attributes that specify if they are permanent or removable and translucent or opaque. These attributes facilitate the work of the engineer in selecting the ones that may play a structural role. However, since these entities are not laid out as part of a functional structural whole they need to be later verified and confirmed by the engineer and then

integrated to the **Structural System**. Finally, **AWalls** have **Openings** representing windows and doors. These **Openings** are defined by the architect.

Note that unlike Björk (1992), Dias (1996) and Khemlani et al. (1998), this representation does not include explicit relations between spaces and their boundaries. Therefore, **ASlabs**, **AWalls** and **AColumns** are not related to **Spaces**. Similarly, in the representation **AWalls** are not explicitly related to each other. Imposing a rigid data-structure limits design flexibility and applicability of the representation for solving general problems. For example, the representation acknowledges that **AWalls** and **AColumns** not only enclose spaces but that they may also partition spaces. Furthermore, in many buildings structural elements and assemblies are positioned inside **Spaces** with reference to their boundaries (e.g. offset). Not considering such relationships explicitly also makes the representation simpler. In this way the design model does not get unnecessarily overcrowded with relationships that can be derived from the geometric model using spatial operations.

This approach does not go against the architect's design intent since it has been well established in practice that walls in buildings have the main architectural function of defining and separating **Spaces** (Rosenman and Gero 1996). When implicit, the bounding relationship between **Walls** and **Spaces**, which is apparent from the architectural model, can become explicit on demand using ***Synthesis Algorithms*** (cf. section 3.3.1.2 b and section 5.3). Such algorithms use the partial or complete geometry and topology of the design model for constructing new geometry and topology, and for verifying the model. Thus, whenever required, space-enclosing **AWalls** can always be derived from enclosed **Spaces** or vice-versa. **AWall** adjacencies and intersections can also be evaluated using ***Synthesis Algorithms***. For example, such relationships can become explicit in building

environmental design where the quality of the indoor environment is directly related to the characteristics of the **ASlabs** and **AWalls** that surround the **Spaces**. For structural engineering, however, maintaining such relationships is expensive and fruitless.

5.2.1.2 Structural domain representation

The structural domain representation includes both purely functional entities (e.g. **Structural Volumes**, **Structural Subsystems** and **Structural Assemblies**) and physical entities (e.g. **Columns** and **Beams**) as for the architectural domain representation (see Figure 5.2). This is unlike most previous efforts in representing the structural system (Björk 1992, Dias 1996, Khemlani et al. 1998, IAI 2004, and SCI 2004), which focus mostly on modeling physical structural elements. Only few of them represent structural assemblies. As described in section 2.2.2.3, purely functional entities are required for modeling the engineer's design intentions and at the same time enabling design refinement (Rosenman and Gero 1996).

As shown in Figure 5.2, **Independent Structural Volumes** are functional entities that act as the main organizers of the **Structural System**. The **Structural System** is subdivided into one or several **Independent Structural Volumes**, each one containing an independent system of load paths to the ground (i.e. a self-contained structural skeleton). **Independent Structural Volumes** also aggregate **Structural Zones**, which are functional entities that group **Spaces** and specify additional structural requirements, such as prescribed loads and column spacing. **Independent Structural Volumes** and **Structural Zones** are volumetric entities and therefore are specializations of **Structural Volumes**.

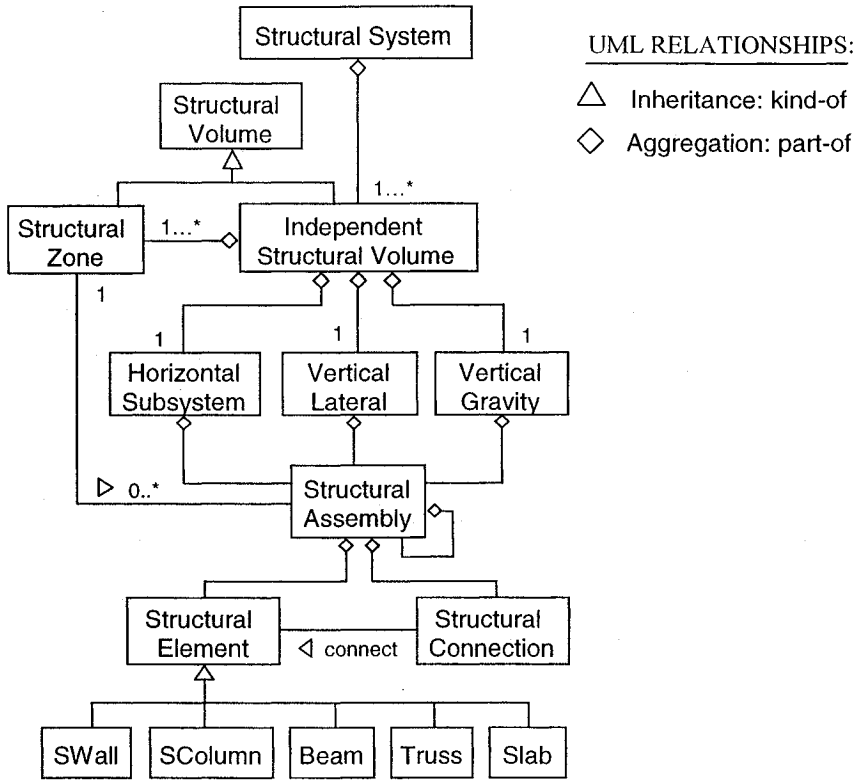


Figure 5.2 Structural system domain representation

Independent Structural Volumes are also subdivided into three types of **Structural Subsystems**: **Horizontal Subsystem**, **Vertical Lateral subsystem**, and **Vertical Gravity subsystem**. **Structural Subsystems** are functional entities that are composed of **Structural Assemblies**, which are arrangements of **Structural Elements**, and may be composed of other **Structural Assemblies** (sub-assemblies). **Structural Assemblies** are specialized into more specific types such as **Frame Assembly**, **Floor Assembly**, **Wall Stack** and **Column Stacks** which are not shown in Figure 5.2 for clarity. The representation includes five types of **Structural Elements**, namely: **SWall** (i.e. the structural view of a **Wall**), **SColumn** (i.e. the structural view of a **Column**), **Beam**, **Truss Element**, and **Slab Element** which is a planar horizontal element that supports

gravity forces directly. **Structural Elements** are joined together through **Structural Connections**.

At the **structural assembly** level, entities are hierarchically classified following standard categories that are found in most structural engineering textbooks. Using UML notation, Figures 5.3, 5.4, 5.5 and 5.6 show examples of some of the hierarchies of **structural assemblies** defined in this research project. As shown in the abovementioned figures, in this research project the description of **structural assemblies** is generic in the sense that it is not associated to any particular construction technology, material(s) or loads. For example a "hollow composite beam on girder with steel deck" is described in the representation as a one-way, three-level floor assembly. The reason for such description is that the focus of this project is to enable reasoning from geometry and topology. Turning generic **structural assemblies** into specific ones requires knowledge (e.g. constructability knowledge) from a knowledge-based component, which is not considered in this project.

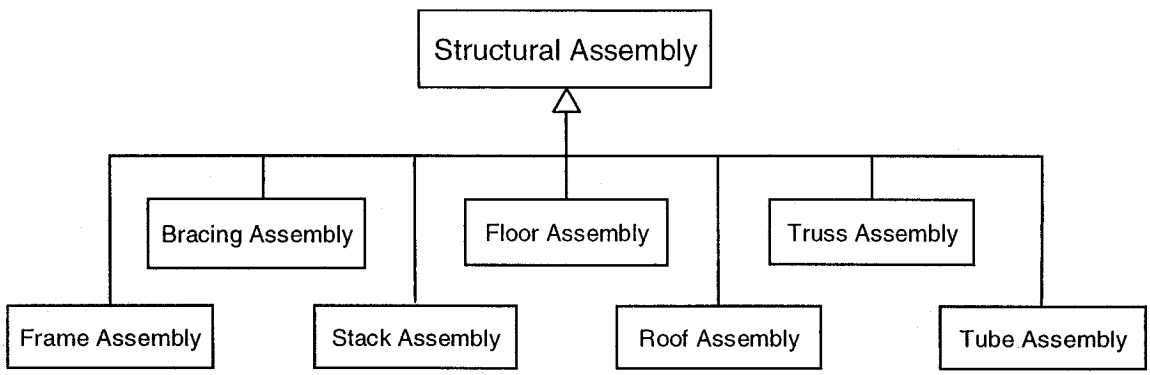


Figure 5.3 Hierarchical classification of **Structural Assemblies**

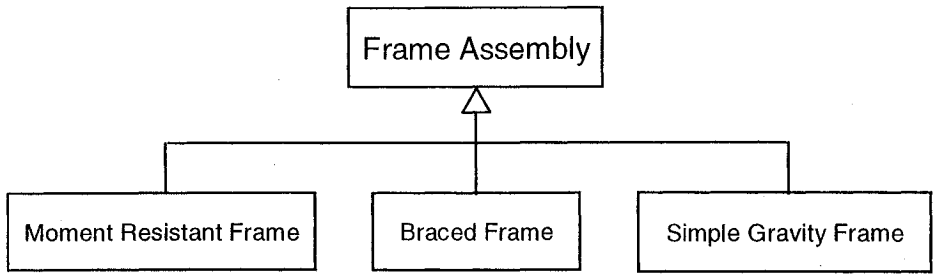


Figure 5.4 Hierarchical classification of **`Frame Assemblies`**

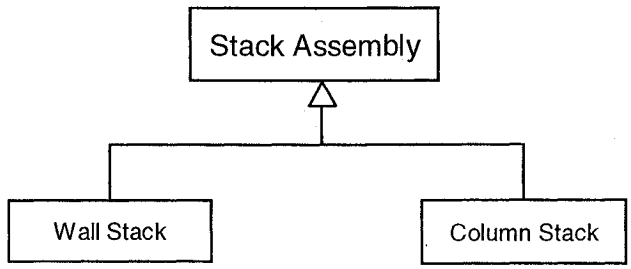


Figure 5.5 Hierarchical classification of **`Stack Assemblies`**

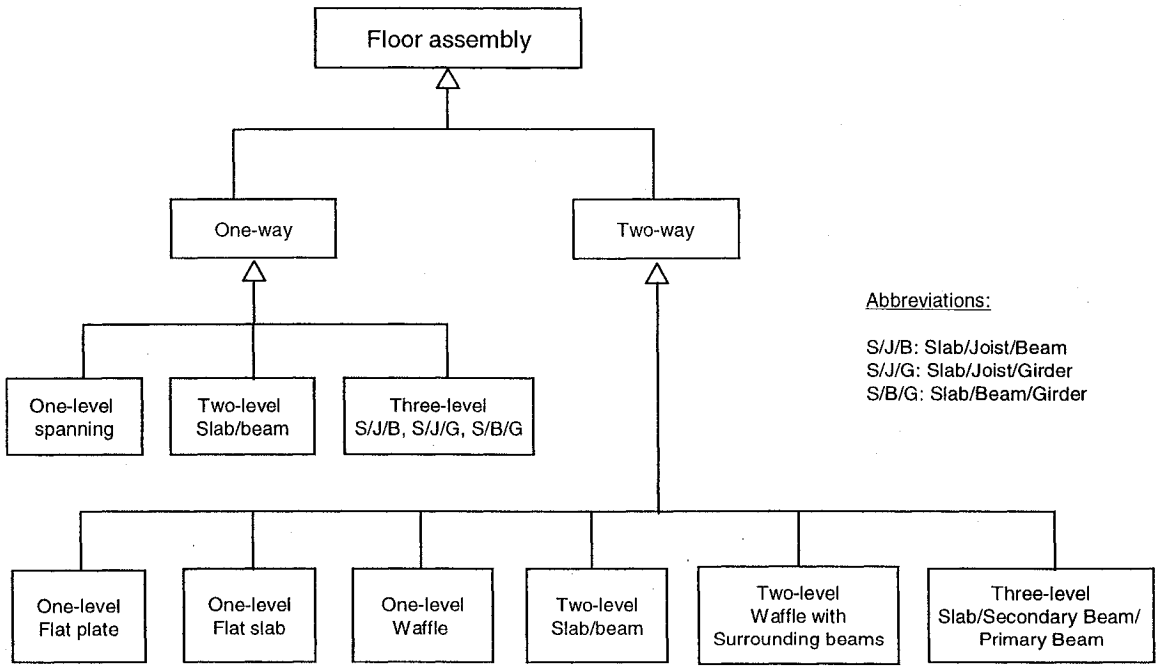


Figure 5.6 Hierarchical classification of **`Floor Assemblies`**

At the **`Structural Element`** level the structural hierarchy expands into a network of interconnections, formally called a graph. More specifically, the physical structural

system is represented by a finite, connected graph, which is made out of **Structural Elements** interconnected together through **Structural Connections**. **Structural Elements** and **Structural Connections** are therefore the indivisible, atomic elements of the representation hierarchy. Graphs are mathematical structures with wide applications in engineering (Shai 2001). Their mathematical properties make them suitable for representing a wide range of engineering problems and systems for computation. A graph is defined by the ordered pair G = (V, E), where V is the vertex set and E is the edge set, and every edge is defined by its two end vertices. A computer representation that relies on graph structures is therefore able to take full advantage of the well understood mathematical properties of graphs, as well as the wide variety of algorithms that have been written to manipulate them.

There are several types of graphs. However, for the present application the most suitable type is the adjacency graph (Meyer 1995), which represents entities at the vertices of the graph while the edges represent the adjacencies among entities. Figure 5.7 shows a simple structure at the left and its adjacency graph at the right. For simplicity, the structure illustrated in Figure 5.7 does not include **Slab Elements**. Therefore, the entities that are represented at the nodes of the adjacency graphs are **Structural Elements** and **Structural Connections**. The adjacency graph representation has been found suitable for conceptual structural design because it is extensible and can later be adapted for detailed analysis and finite element analysis (Meyer 1995).

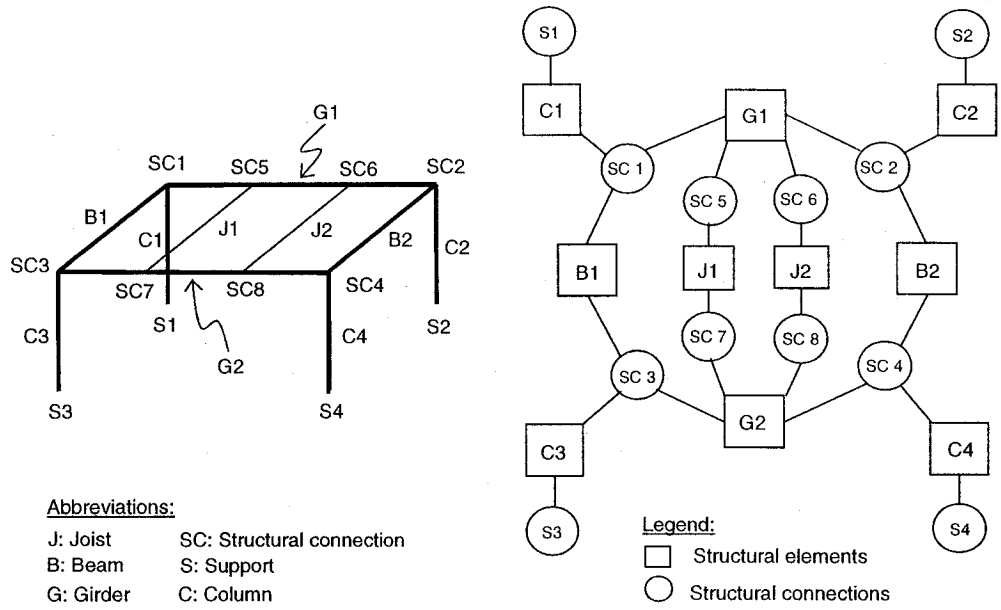


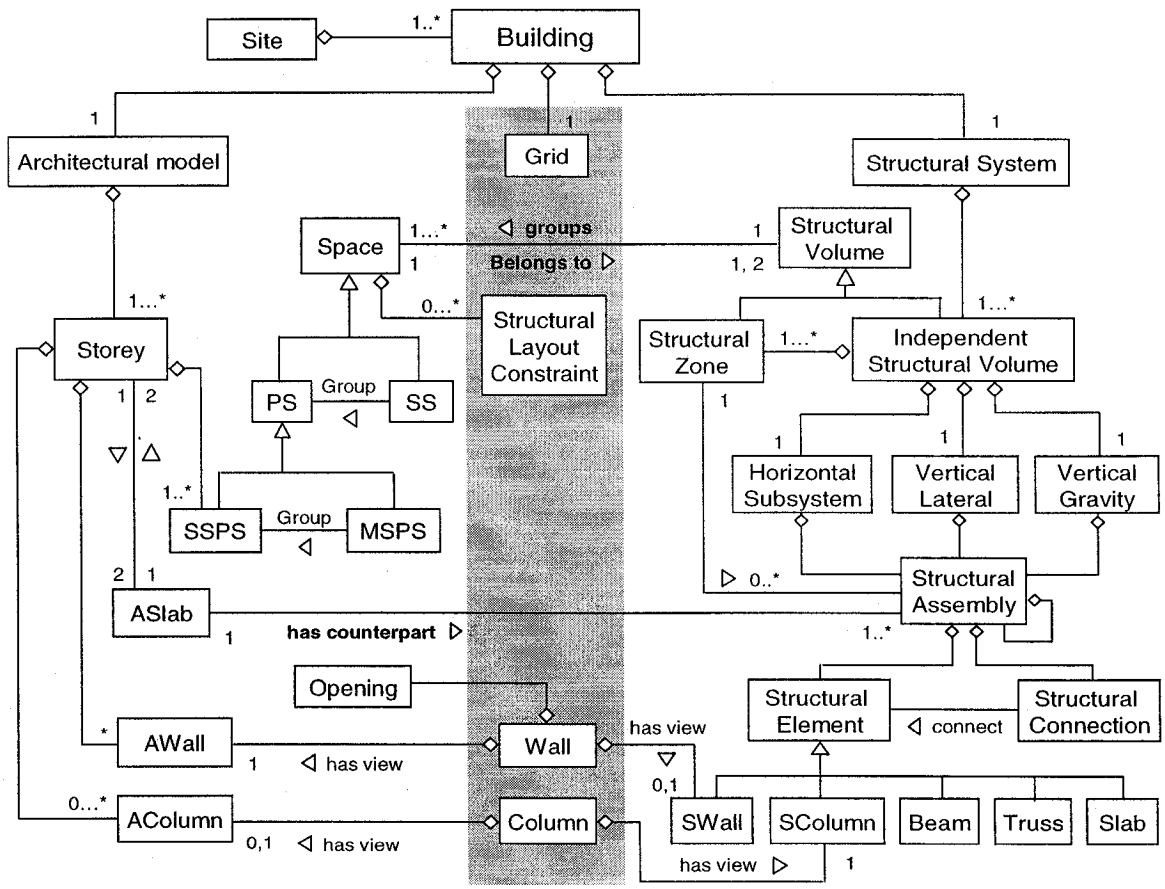
Figure 5.7 Simple structure and its adjacency graph

5.2.1.3 *Integrated Representation*

The principle behind the ***Integrated Representation*** is that the architectural model is mostly configured storey-wise (Leclercq 1996). Therefore, the majority of the relationships and constraints between architectural entities take place within **`Stories`**. This means that relatively few vertical continuity constraints are defined by the architect. However, when the structure is integrated with the architecture, **`Stories`** get connected at suitable locations and vertical continuity constraints are added to the model. As shown in Figure 5.8, the ***Integrated Representation*** combines the structural system domain representation and the building architecture domain representation.

The initial and more abstract entities, which are defined by the architect, are the **`Site`** and the **`Building`**. The **`Site`** holds general information about its geometry, type of soil and location, while the **`Building`** holds information pertaining to its type (office, apartment, mixed-use, etc.) and number of stories. There can be one or more **`Buildings`**

associated to a given **Site**. The **Building** also includes the **Grid** entity that is in charge of simplifying the configuration of the **Spaces** and the **Structural System** in the floor plan; this is a common **Grid** that is shared by the architect and the structural engineer. In addition to the **Grid** entity, the **Building** entity has two sub-entities: **Architectural Model** and **Structural System**, which are decomposed into their constituent entities. The left part of Figure 5.8 shows the **Architectural Model** and the right part shows the **Structural System**. At the center of the figure are the key entities and relationships that act as links between both disciplines.



Abbreviations:

PS: Primary Space
SS: Secondary Space
SSPS: Single-storey PS
MSPS: Multi-storey PS

Figure 5.8 UML diagram of the ***Integrated Representation***

The integration between the **Architectural Model** and the **Structural System** is carried out through a set of entities and relationships that link the two models at different levels of refinement (see gray zone at the center in Figure 5.8). The first link between the **Architectural Model** and the **Structural System** takes place between **Spaces** and **Structural Volumes**. When **Spaces** are grouped into **Structural Zones** the functionality of the **Spaces** is translated into structural requirements and constraints that limit the synthesis of structural configurations. This is achieved through **Structural Layout Constraints** that allow the architect to restrict the layout of **Structural Elements** within certain **Spaces**. Examples of **Structural Layout Constraint** are the **column_free Constraint** and the **beam_free Constraint** (e.g. limiting visible beams spanning **Multi-Storey Primary Spaces**). The above constraints incorporate some architectural design intent into the model that may not be obvious to the structural engineer. Other types of **Structural Layout Constraints** are explained in section 5.2.2.7.

The next link takes place at the **Structural Assembly** level. From the architectural view of a **Slab** (i.e. the **ASlab**) the engineer generates one or more **Floor Assemblies**. Usually, the geometry of a **Floor Assembly** is limited by the **Independent Structural Volume** where it belongs. However, if an **ASlab** overlaps **Structural Zones** with dissimilar support requirements, then it may require to be divided by the engineer into various **Floor Assemblies**.

The lowest link between the two disciplines takes place through **Wall** and **Column** entities. On the one hand, **Walls** are architectural elements that can be given structural roles, and therefore they have two views. The architectural view (**AWall**) holds

information that is relevant to the architect, for example its link to a given **Storey**, while the structural view (**SWall**) holds structural information only, as well as its relationship to a **Wall Stack**. On the other hand, **Columns** are structural elements (**SColumn**) that may also play an architectural role (**AColumn**); therefore they also have two views. Furthermore, when **AColumns** are tentatively laid out by the architect and verified by the engineer, if they are structurally relevant, they are aggregated into **Column Stacks** and integrated to the **Structural System**.

The following subsection describes briefly how spatial information is included in the ***Integrated Representation***. Therefore, an overview of how basic knowledge and constraints encapsulated in the entities from the ***Integrated Representation*** are used for assisting the engineer during structural synthesis.

a) Representation of spatial information

In this research project, architectural and structural building elements are spatially represented with a single fundamental representation (i.e. primary spatial representation) from which all the required spatial abstractions can be derived (cf. section 2.4.2.5). A primary spatial representation is adequate for describing the geometry of the entities that make up the building during conceptual design. Considering architectural entities, **Spaces** are represented using a primary spatial representation as three-dimensional (3D) volumes, **Walls** and **Slabs** are represented as two-dimensional (2D) planar surfaces, and **AColumns** are represented as one-dimensional (1D) wires. Correspondingly, the decomposition of structural entities is also supported by different geometric representations: **Structural Volumes** are represented as (3D) entities, **Structural Sub-systems** do not have geometry of their own, but their geometry is defined through

aggregation of their constituent **Structural Assemblies** which are represented as (2D) planar surfaces (e.g. **Wall Stacks** and **Floor Assemblies**) and (1D) wires (e.g. **Column Stacks**). Therefore, **Structural Assemblies** have an abstract geometry that defines their position, orientation and extension, and a specific geometry that is defined by the geometry of their constituent **Structural Elements**. At the **Structural Element** level, **Slab Elements** and **Walls** are represented as 2D planar surfaces while **Beams** and **Columns** are represented as 1D wires.

b) Structural knowledge and constraints

Structural entities incorporate a basic description of themselves and how they interact with other entities (i.e. design product knowledge). So far, the structural entities incorporate knowledge about the structural form, while knowledge about their function and expected behavior is not included. The initial intent is therefore, to provide a basic level of assistance to competent structural engineers that have a thorough understanding of the structure function and behavior, which they use to produce the structural form. Once configured by a competent structural engineer, a structural form can be transferred to structural analysis programs for assessing its behavior.

Structural entities incorporate knowledge about the structural form as object-oriented methods that constrain their configuration as well as the configuration of related structural entities, thus reducing the range of configuration possibilities (Gero 1990). Such methods rely mostly on geometric and topologic concerns and on generic structural engineering knowledge. The types of constraints that are encapsulated by the entities in the representation are feasibility or hard constraints because they are enforced in order to

produce feasible structural solutions. Since the structural system is hierarchically described, entity interactions take place vertically (i.e. between abstract entities and their constituents), and horizontally (i.e. among entities at each hierarchical level). Vertically, parent entities verify that their child entities are well connected and well positioned within the spatial limits defined by the parent entity. Thus, the integrity of the structural system is guaranteed mostly by part-whole constraints that are verified using ***Synthesis Algorithms*** (cf. 3.3.1.2 b). First, **`Independent Structural Volumes`** constrain the geometry of **`Structural Assemblies`** so that they lie within the volume that they contain. **`Structural Subsystems`** limit the types of **`Structural Assemblies`** as required to make the subsystem behave as expected. **`Structural Zones`** constrain the type and location of **`Structural Assemblies`** that they enclose. At lower levels of structural decomposition, a **`Frame Assembly`** restricts the location of its constituent **`Structural Elements`** to be within its plane (geometric, planarity constraint). Horizontally, **`Structural Elements`** also incorporate structural knowledge to verify if they are properly related to other **`Structural Elements`** (i.e. properly supported). Hence, vertically in the hierarchy **`Structural Assemblies`** enforce valid arrangements of the **`Structural Elements`** that make them up, while respecting the horizontal hierarchical interaction constraints coming from their constituent elements. For example, a **`Joist`** may be directly supported by a **`Column`**, however in a three-level (slab/joist/beam) **`Floor Assembly`**, **`Joists`** must be supported by primary **`Beams`** (topologic and associativity constraint). In addition, according to its type, each particular **`Structural Assembly`** requires specific types of **`Structural Connections`** as well as allowable spans for its **`Structural Elements`**.

5.2.2 Detailed organization of the *Integrated Representation*

As mentioned in section 5.2.1, the description uses standard UML notation. However, the only change that has been made to this notation is to represent geometry using a pentagon instead of the standard rectangle to distinguish it from other types of entities. This serves two purposes, first to highlight the geometry of entities, and second to identify those entities that come from ACIS (Spatial Corp. 2004) ***Geometric Modeling Kernel***. In the following discussion some relationships between entities have been intentionally omitted from the diagrams for the sake of clarity.

5.2.2.1 `Representation Entities`

At the highest level of abstraction in the representation, object-oriented inheritance relationships are merely used to classify entities according to their type. As shown in Figure 5.9, all entities in the representation are **`Representation Entities`**, therefore they all inherit name and identity attributes. While a name is specific to a particular entity, the identity defines a type for an entity. For example, a **`Slab Assembly`** having different types of **`Structural Elements`** can identify their type (e.g. a **`Girder`** or a **`Beam`**) by requesting their identity.

All the entities in the diagram have already been explained in Figure 5.8 except for the **`reference Entities`**, which are used as aids in creating the design model. Two types of **`Reference Entities`** have been defined: these are **`Construction Lines`** and **`Grids`**. The rest of the entities in the representation are classified either as **`Architectural Entities`** or **`Structural Entities`**. This classification is intended to serve for manipulation and visualization of entities for a given domain; for example, this gives the

structural engineer the option of visualizing and manipulating **Structural Entities** only. Note that the **Ground** has been classified as a **Structural Entity** since it is used only for checking gravity and lateral load paths.

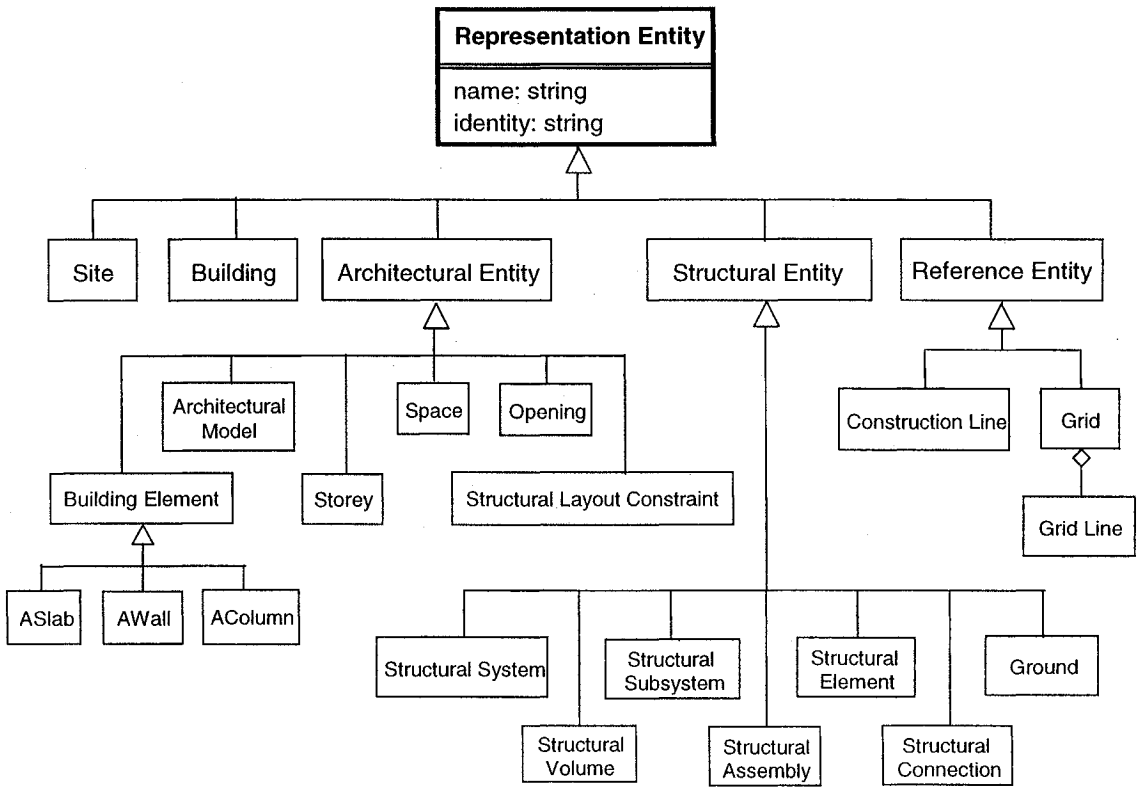


Figure 5.9 Classifications of **Representation Entities**

5.2.2.2 Structural Connections

As shown in Figure 5.10, the ***Integrated Representation*** includes three types of **Structural Connections**, namely: **Node Connection**, **Line Connection** and **Plane Connection**, which are respectively zero-dimensional (0D), one-dimensional (1D) and two-dimensional (2D) respectively as described below.

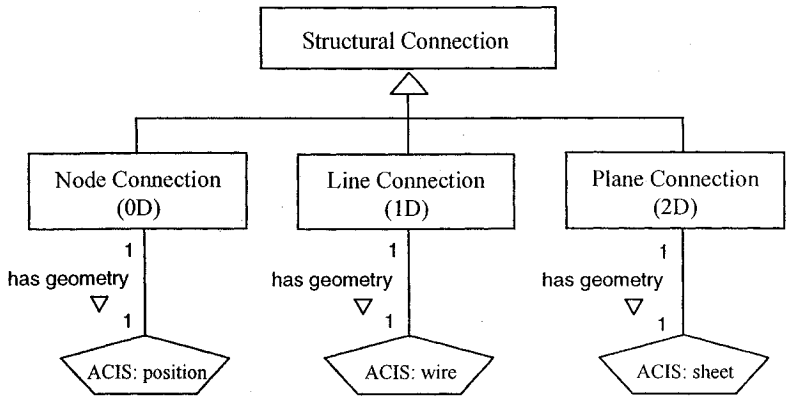


Figure 5.10 Types of **`Structural Connections`**

a) `Node Connections`

A **`Node Connection`** is a zero-dimensional entity that joins linear **`Structural Elements`** together (e.g. **`Columns`** and **`Beams`**). It is also used whenever it is required to join 1D **`Structural Elements`** with 2D **`Structural Elements`** for example, a **`Slab Element`** that may be directly supported at each corner by a **`Column`**, without using perimeter **`Beams`**. As another example, a **`Beam`** may be supported at one of its ends by a **`Wall`**. A **`Node Connection`** is characterized by a position in space and an array of support conditions. These conditions specify six displacement degrees of freedom (three translations and three rotations), which are aligned along the global Cartesian coordinate system.

The position of the **`Node Connection`** is defined within the model through an ACIS® *`position`* entity. The support conditions at the node are specified by the **`Structural Assembly`** that creates the **`Node Connection`**. The **`Node Connection`** stores the support conditions in a six-element Boolean array as follows:

supportConditions[0] = displacement in global "x" direction = true/false
supportConditions[1] = displacement in global "y" direction = true/false

supportConditions[2] = displacement in global "z" direction = true/false
supportConditions[3] = rotation about global "x" axis = true/false
supportConditions[4] = rotation about global "y" axis = true/false
supportConditions[5] = rotation about global "z" axis = true/false

In order to connect structural entities of higher-dimensionalities **Line Connections** and **Plane Connections** are required. The need for these two types of connections comes from the very definition of a structure which consists essentially of structural elements joined together through connections (i.e. any two **Structural Elements** cannot be directly related).

b) Line Connections

Line Connections are 1D entities that connect 2D **Structural Elements** together. As indicated in Figure 5.10 their geometry is an ACIS® *wire* that could have the shape of a straight line or a curve; however for the prototype presented in Chapter 6, only straight lines are accepted. A **Line Connection** is used to connect two **Walls** together and a **Slab Element** with a supporting **Wall**. **Line Connections** are also used to connect 1D **Structural Elements** with 2D **Structural Elements**. For example, a **Slab Element** supported by four perimeter **Beams**.

c) Plane Connections

Unlike **Node Connections** and **Line Connections** that are used to join **Structural Elements** belonging to the physical **Structural System**, **Plane Connections** join 3D **Structural Volumes** at the highest level of structural decomposition. In this

research project however, the connections among **Structural Volumes** are not considered. Therefore **Plane Connections** are not developed further.

5.2.2.3 Columns and Column Stacks

Columns (Figure 5.11) can be created either by the architect or by the engineer. When a **Column** is created by the architect, the computer automatically creates its architectural view (i.e. **AColumn**), whereas when a **Column** is created by the engineer, the computer automatically creates both its structural view (i.e. **SColumn**) and also its architectural view. This is because a **Column** always needs to be associated with a **Storey**, and this association is given by the **AColumn** view only. If a **Column** that is created by the architect is used by the engineer, then the structural view is also created. Therefore, a **Column** is likely to have both views. However, in few cases a **Column** may only have its architectural view. For example, a **Column** that is defined by the architect for decorative purposes only has no structural purpose.

The **Column** entity holds information common to both views, mainly its geometry and position in space. The geometry of a **Column** is represented by an ACIS® *wire body*, while its position is represented by an ACIS® *position* entity. The **AColumn** view holds information that is relevant for the architect, such as its relationship with a particular **Storey**. The **SColumn** view holds information that is required from the structural engineering standpoint, such as integration (i.e. connectivity) with the **Structural System**, material and the loads it carries.

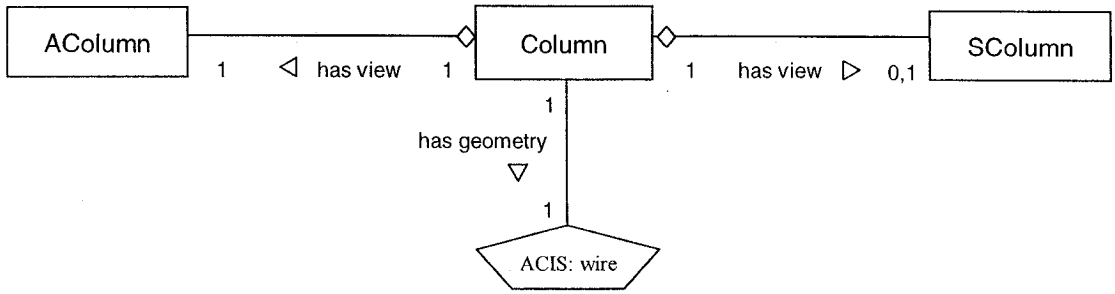


Figure 5.11 A `Column` and its domain views

A `SColumn` is therefore a vertical rectilinear `Structural Element` that has an architectural view and spans a single `Storey`. When adjacent `SColumns` are "stacked" one on top of the other, they make up `Column Stack` assemblies. A `Column Stack` usually runs from the foundation up to the roof of the building. However, it may be interrupted at the top, before reaching the roof, or at the bottom, before reaching the foundations. In the latter case, a transfer structure is required to take the load coming from the `Column Stack` and transfer it to supporting members below. Transfer structures are not considered in the current ***Integrated Representation***. Therefore `Column Stacks` are always assumed to be connected to the foundations. Figure 5.12 illustrates the topological description of a `Column Stack`.

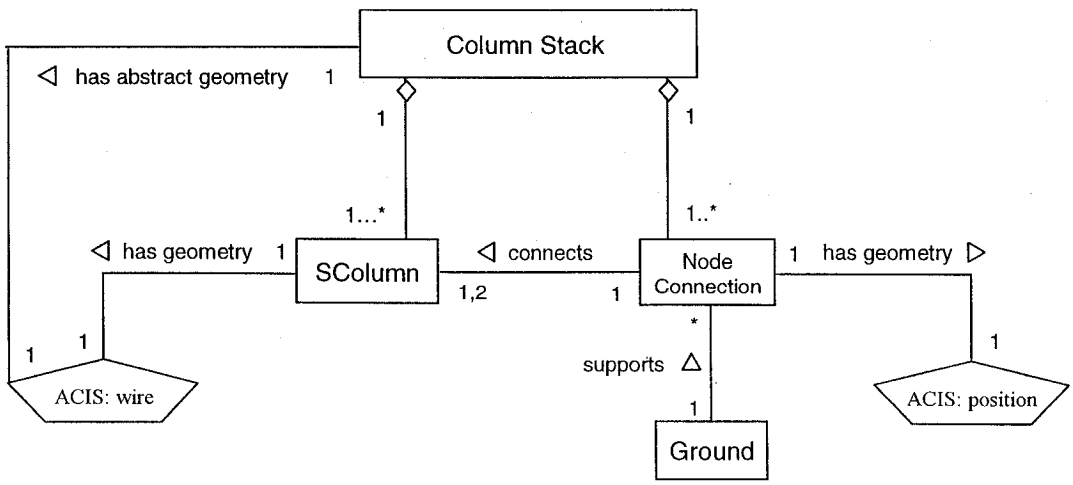


Figure 5.12 Topological description of a `Column Stack`

In Figure 5.12 it is shown that a **Column Stack** aggregates one or many **SColumns**. A **Column Stack** also aggregates **Node Connections** connecting two adjacent **SColumns** or a single **SColumn** to the **Ground** entity, which is used for load-path verification. The abstract geometry (cf. 5.2.1.3.a) of a **Column Stack** assembly is an ACIS® *wire*, which is computed as the Boolean union of the geometries of its aggregated **SColumns**. As part of the **Vertical Gravity Subsystem**, **Column Stack** assemblies have the responsibility of guaranteeing **SColumn** continuity down to the **Ground**.

5.2.2.4 Walls and Wall Stacks

Just like **Columns**, **Walls** can be created either by the architect or by the engineer. However, they are usually created by the architect. Whenever the architect creates a **Wall**, an architectural view (**AWall**) is automatically created by the computer. Whenever, the engineer creates a **Wall** its structural view (**SWall**) is automatically created as well as its architectural view (**AWall**) that relates the wall to a given **Storey**. Therefore, as shown in Figure 5.13, a **Wall** entity always has an **AWall** view, but its **SWall** view is created either when the engineer creates the **Wall** or when the engineer uses a **Wall** as created by the architect for structural purposes.

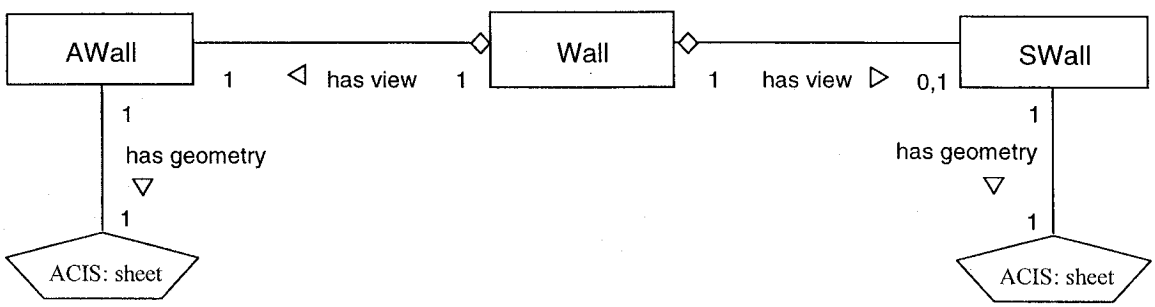
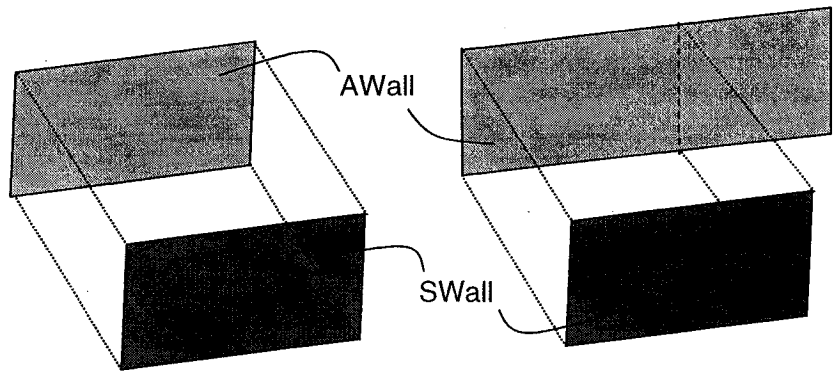


Figure 5.13 A **Wall** and its domain views

There can be one-to-one mapping between the geometry of an **AWall** and its corresponding **SWall** (Figure 5.14 a). In some situations as illustrated in Figure 5.14 b, the engineer may use only part of the **Wall** as defined by the architect for structural purposes. Therefore, each **Wall** view has its own geometry (Figure 5.13), which is an ACIS® *sheet*.



(a) One-on-one mapping (b) **SWall** is only part of **AWall**

Figure 5.14 Illustration of the Architectural views and structural views of a **Wall**

The **Wall** entity holds information that is relevant to both disciplines such as position in space, geometry and direction for shear action. It also has a list of **Openings**. On the one hand, the **AWall** view holds information that is relevant only to the architect, such as the relationship with the **Storey** where it belongs and its function within the building architecture. This function specifies whether a **Wall** is intended to be permanent or removable, translucent or opaque. On the other hand, **SWall** view holds information that is relevant to the engineer, such as connectivity to the rest of the structural system, material and the loads it carries.

Several **SWall**s stacked one on top of the other make up a **Wall Stack** assembly. Figure 5.15 shows the topological structure of a **Wall Stack** as part of the **Structural System**.

Therefore, as illustrated in Figure 5.15, a **Wall Stack** has one or many **SWalls** and many **Line Connections** joining together any two adjacent **SWalls**, as well as the bottom **SWall** to the Ground. The abstract geometry of a **Wall Stack** is an ACIS® *sheet body*, which is computed as the Boolean union of the geometries of its aggregated **SWalls**.

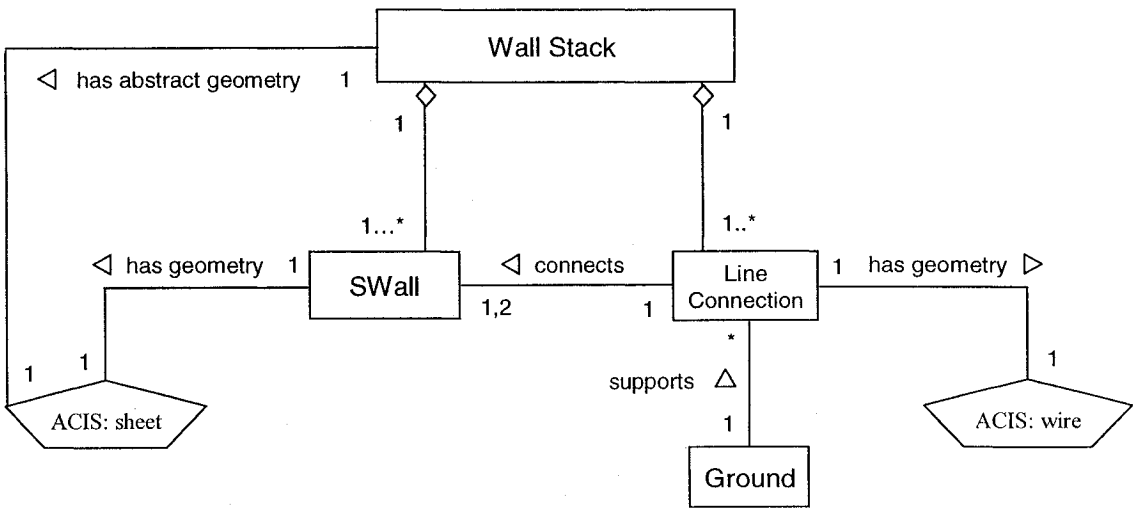


Figure 5.15 Topological description of a **Wall Stack**

A **Wall Stack** assembly is part of the **Vertical Lateral Subsystem** in the direction of the **Wall Stack**, and it is also part of the **Vertical Gravity Subsystem**. As part of these subsystems, it has the responsibility of guaranteeing continuous load paths down to the ground.

5.2.2.5 Frame Assemblies

Three types of **Frame Assemblies** are considered in the representation. These are two-dimensional frames called: **Moment Resistant Frame**, **Single Gravity Frame** and **Braced Frame**. Figure 5.16 shows the topological structure of a typical **Frame Assembly**

with **Wall Stack** and **Column Stack** assemblies. It shows relationships in the plane of the **Frame Assembly** only.

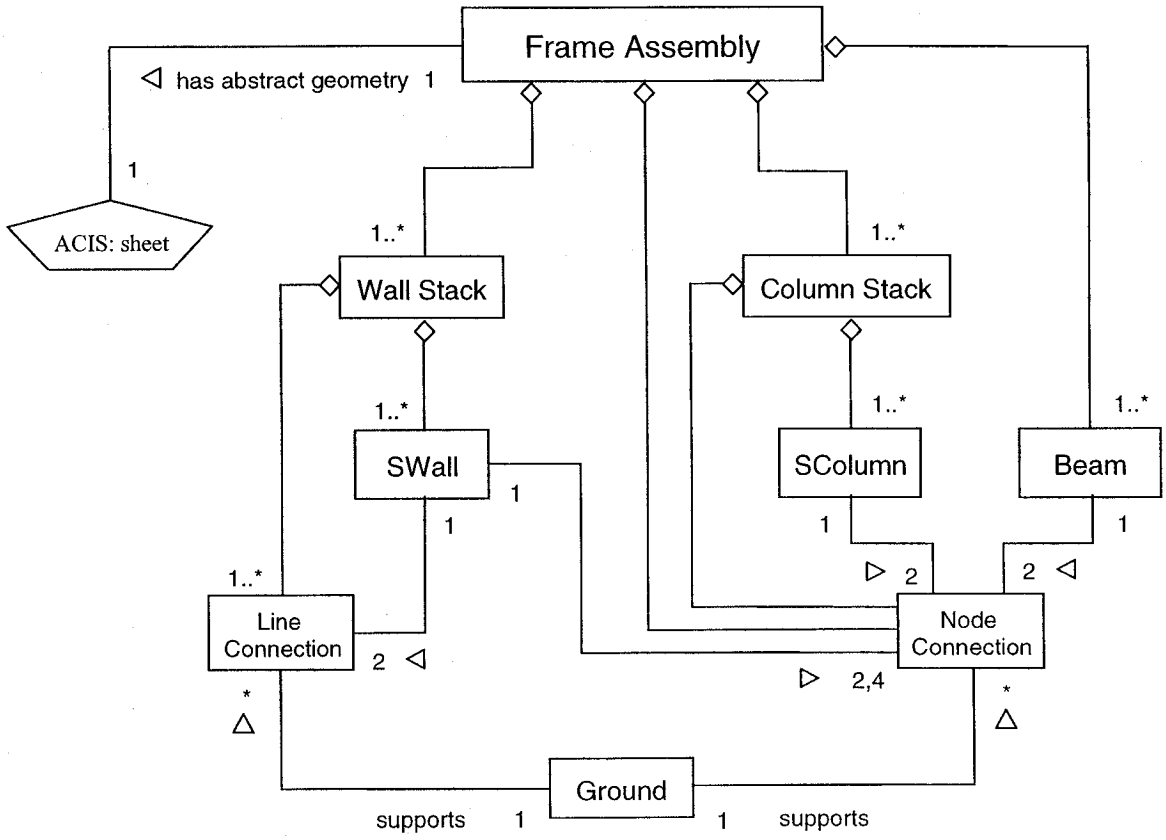


Figure 5.16 Topological description of a **Frame Assembly**

Figure 5.16 illustrates that a **Frame Assembly** is composed of **Wall Stacks**, **Column Stacks**, **Beams**, **Node and Line Connections**. **Node Connections** join **Beams** to **SColumns** and **SWalls**, and **SColumns** to the **Ground**. As shown in the diagram, **SWalls** are integrated to the **Frame Assembly** through **Node Connections**. Thus, **Node Connections** located at the four corners of a **SWall** connect the **SWall** to the **Beams**, except for the **SWall** located at the bottom of the **Wall Stack** that is connected to two **Beams** at the top through **Node Connections**, and to the **Ground** at the bottom through a **Line Connection**. A **Frame Assembly** incorporates methods that guarantee that all its

Structural Elements and subassemblies are always properly connected. As illustrated in Figure 5.16, the abstract geometry of a **Frame Assembly**, which is a vertical plane, is represented as an ACIS® *sheet body*. Figure 5.16 shows relationships in the plane of the frame only. Therefore it does not indicate that **Column Stacks** usually belongs to two orthogonal **Frame Assemblies**. In addition to **Frame Assembly**, **Beams** also belong to **Floor Assemblies**. These relationships are not indicated in Figure 5.16.

Figure 5.17 illustrates the topology of a **Frame Assembly** with a **Wall Stack** (i.e. a shear wall). In the example, all the objects in the model are represented by circles, except for the "ground" object. In the diagram, objects are identified by an initial or initials followed by a number; thus, **SWalls** are labeled with a "w", **Beams** are identified with a "b", **Node Connections** are labeled with "nc", and **Line Connections** are labeled with "lc". Note that only the part of the **Frame Assembly** that is enclosed by a dotted line on the left is represented in the diagram on the right.

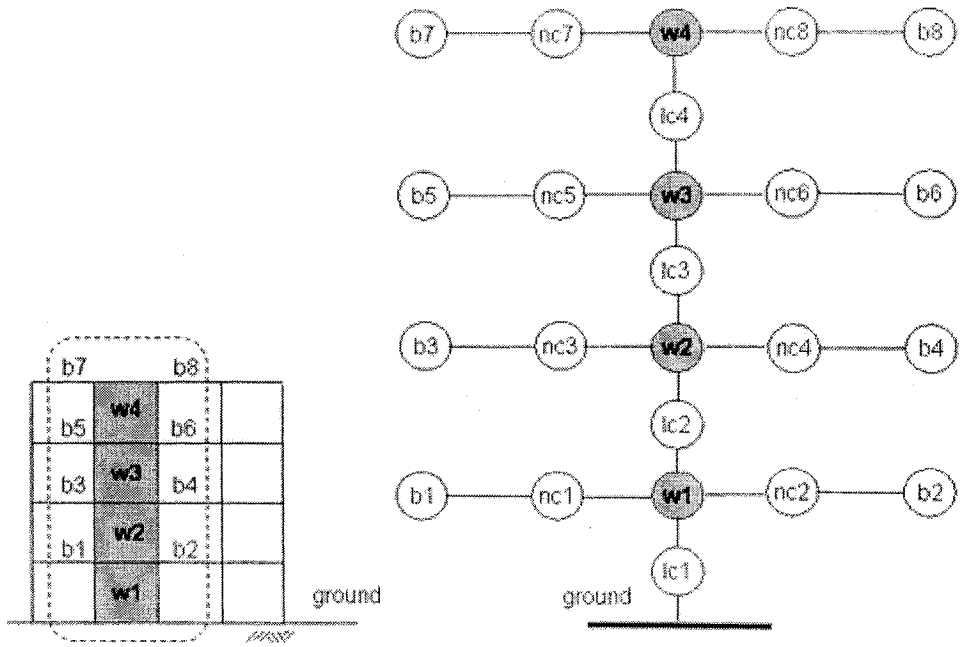


Figure 5.17 Example of the topology of a **Frame Assembly** with a shear wall

5.2.2.6 Floor Assemblies

As illustrated in Figure 5.18, **Floor assemblies** are composed of **Slab Elements**, **Beams**, **Line Connections** and **Node Connections**. Depending on the type of **Floor Assembly**, **Slab Elements** may be supported by **Beams** through **Line Connections** or directly by **Columns** through **Node Connections**. A **Slab Element** may also be supported (through **Line Connections**) by **SWalls** running parallel to its sides.

Three types of **Floor Assemblies** are included in the representation: (1) "one-level" in which a **Slab Element** is directly supported at its vertices by **SColumns**, (2) "two-level slab-on-beam" where a **Slab Element** is supported by perimeter primary **Beams** and these **Beams** are supported by **SColumns**, and (3) "three-level slab-on-beam-on-girder" where a **Slab Element** is supported by **Beams** (secondary **Beams**) that are in turn supported by Girders (primary **Beams**).

Each **Floor Assembly**, depending on its type, defines a hierarchical support organization for its **Structural Elements**. For example, a "one-level" **Floor Assembly** seeks to support **Slab Elements** directly by **SColumns**. However, when other types of supports are available (for example a **Beam** or a **SWall**) they are also used. Similarly, a "two-level" **Floor Assembly** gives priority to perimeter **Beams** to support **Slab Elements** and **SColumns** to support **Beams**. However, when other types of supports are available they are also used with the engineer's consent.

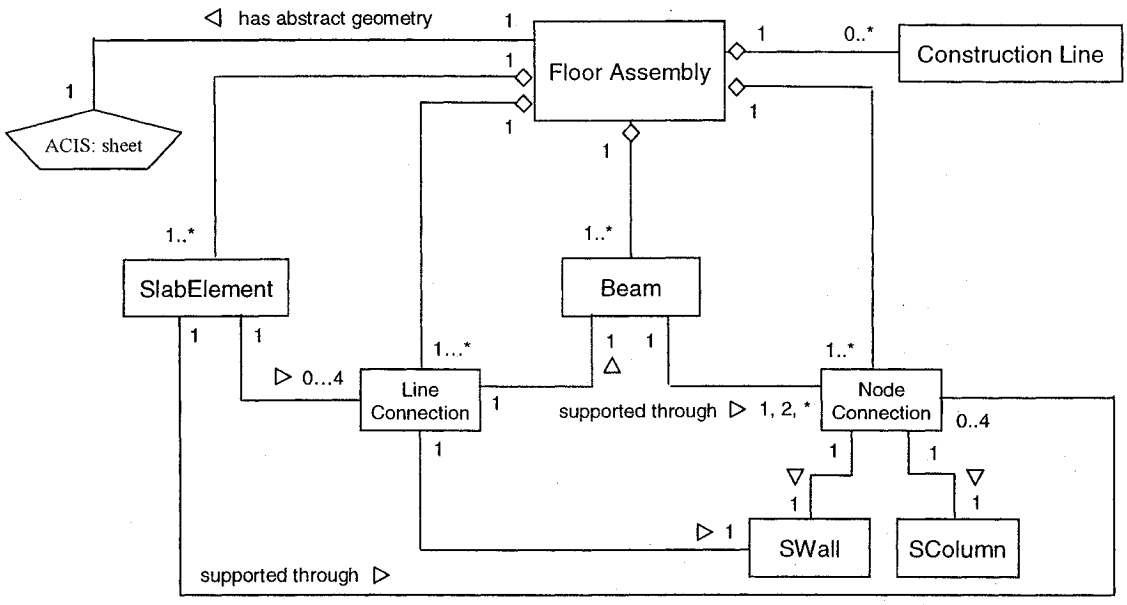


Figure 5.18 Topological description of a **`Floor Assembly`**

As illustrated in Figure 5.18, the abstract geometry of a **`Floor Assembly`**, which is a horizontal plane, is represented as an ACIS® *sheet body*. In addition to the entities that make up its topological structure, Figure 5.18 shows that a **`Floor Assembly`** also may contain reference entities called **`Construction Lines`** which define alignments that are used for assisting the configuration of **`Structural Elements`** within the **`Floor Assembly`**. Within a **`Floor Assembly`**, **`Construction Lines`** are actually grouped in two sets, one for each of the orthogonal framing directions. The application of **`Construction Lines`** is presented in section 5.3.2.

As mentioned in section 5.2.1.2 in the current stage of development, **`Floor Assemblies`** are generic in the sense that they are not associated to any particular construction technology, material(s) and loads. The main focus is in maintaining an adequate topological organization of the **`Structural Elements`** according to the type of **`Structural Assembly`**. More specific **`Structural Assemblies`** need to be included for

better support for structural synthesis. For example, tables and charts have been published including span thresholds for different specific types of **Structural Assemblies** (Schodek 2003). Nevertheless, generic **Floor Assemblies**, as being developed in this research project, incorporate "maximum_span" and "minimum_span" attributes that are used for checking spans based on thresholds entered in advance by the engineer.

5.2.2.7 Structural Layout Constraints

The entities in the architectural model are not always sufficient for conveying the architect's intent to the engineer, particularly concerning architecturally acceptable locations for placing **Columns**. The ***Integrated Representation*** uses **Structural Layout Constraints**, which are defined by the architect for restricting the placement of **Columns** within **Spaces** (see Figure 5.8) and conveying to some degree the architect's intent to the engineer. As shown in Figure 5.19, two main types of **Structural Layout Constraints** have been incorporated in the ***Integrated Representation***, namely: **Column Layout Constraints** and **Beam Layout Constraints**.

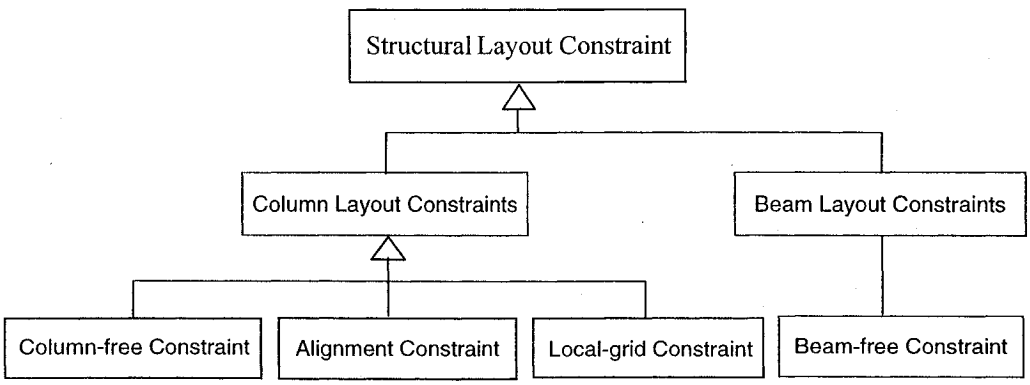


Figure 5.19 Types of **Structural Layout Constraints**

Three types of **Column Layout Constraints** have been created as follows:

- **Column-free Constraint**: is the simplest type of **Structural Layout Constraint**. Its function is not to allow any **SColumn** to be placed within a given **Space**.
- **Alignment Constraint**: specifies architecturally acceptable alignments where the engineer may place **SColumns** within **Spaces**.
- **Local-Grid constraint**: defines a local **Grid** (subset of the **Grid**) inside a given **Space**, which defines "x,y" positions where **SColumns** have to be placed. This constraint is actually used by the architect for placing **AColumns** within a **Space**.

Considering **Beam Layout Constraints**, the only one that has been implemented so far is the **Beam-free Constraint**. This constraint is used for restricting visible **Beams** in **Multi-storey Primary Spaces** (MSPS) as described in section 5.2.1.1. As mentioned above, MSPSs span more than one **Storey** and therefore do not allow the presence of **Floor Assemblies** inside them. However, **Beams** may run between **Floor Assemblies** at intermediate **Stories**. Therefore, it is up to the architect to allow or restrict the presence of these **Beams** within MSPSs.

Structural Layout Constraints are hard constraints since they specify demands or requirements from the architect. However, they are always subject to negotiation between both parties. Any particular **Space** can associate only one **Column-Free Constraint** or one **Local-Grid Constraint**. However, it may have several **Alignment Constraints**. In the future it is expected that more powerful **Structural Layout Constraints** will be implemented. Nevertheless, at the present stage the types of constraints that have been developed provide great versatility in restricting the placement of **SColumns** inside **Spaces**. This is illustrated in the examples shown in Figure 5.20.

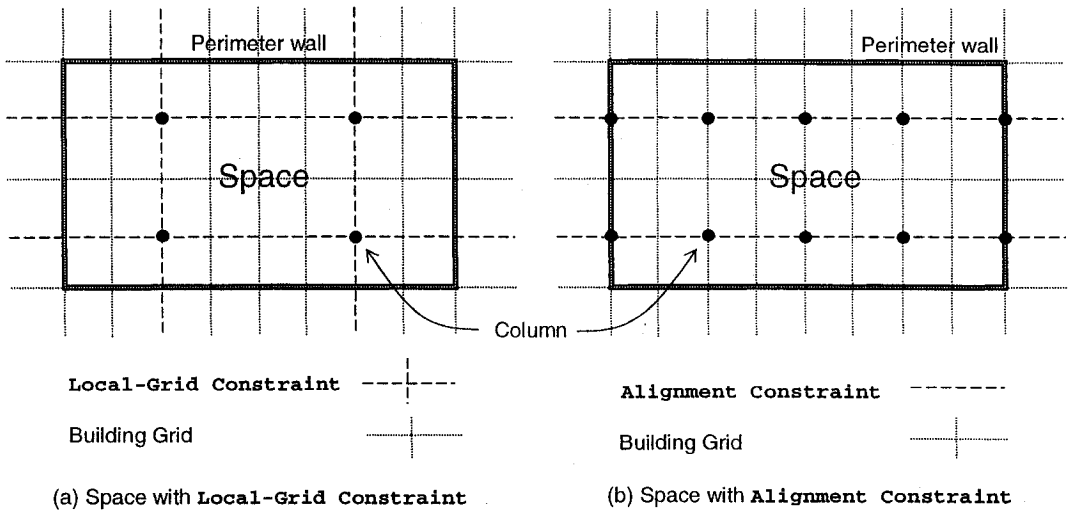


Figure 5.20 Examples of **Structural Layout Constraints**

In Figure 5.20 (a) the architect uses a **Local-Grid Constraint** to place **Columns** inside a **Space**. The **Local-Grid Constraint** defines a **Grid** inside a **Space** which is a subset of the **Building Grid**. In Figure 5.20 (b) the architect uses two **Alignment Constraints** to define architecturally feasible alignments for placing **SColumns**. The engineer is therefore limited to placing **SColumns** along those alignments.

5.2.2.8 The role of the Structural Zones

As described in section 5.2.1.3, when **Spaces** are grouped into **Structural Zones** the functionality of the **Spaces** is translated into structural requirements and constraints that limit the configuration of **Structural Assemblies** and **Structural Elements** within **Structural Zones**. Figure 5.21 shows the main attributes (above the horizontal line) and methods (below the horizontal line) that are incorporated by a **Structural Zone**.

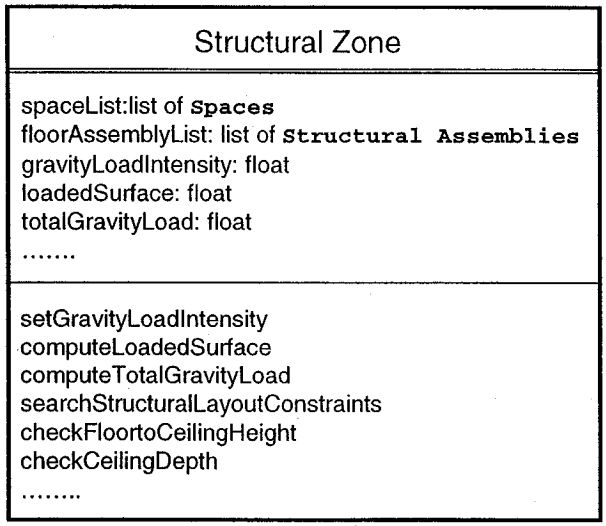


Figure 5.21 Main attributes and methods of a **Structural Zone**

The attributes described in Figure 5.21 are meant to support functional integration within **Structural Zones** (i.e. intra-zone) and not among them (i.e. inter-zone) as described in section 4.3.2.1. The first attribute is a list of grouped **Spaces**. Then, there is a list of **Structural Assemblies** that are in fact **Floor Assemblies** as indicated by the name. As illustrated in Figure 5.8, **Structural Assemblies** may also be associated to **Structural Zones**. However, the relation shown in Figure 5.8 is actually between **Structural Zones** and **Floor Assemblies** (i.e. a type of **Structural Assembly**). This association takes place only whenever there is a **Structural Zone** carrying special functional requirements from the building architecture that force the creation of **Floor Assemblies** for that specific **Structural Zone**. For example, if a **Structural Zone** is a large column-free sports facility within an institutional building it is likely to require a special type of roof and floor **Structural Assemblies** to accommodate the specific functional requirements within the **Spaces**. Then the "*gravityLoadIntensity*" attribute holds an applied gravity load per square meter that results from the occupancy of Spaces. In the current representation, the value for this attribute is entered by the engineer. The

"*loadedSurface*" attribute stores the actual loaded surface area in square meters. This area is calculated using the method "*computeLoadedSurface*". By relying on the algorithms provided by ACIS (Spatial Corp. 2004) ***Geometric Modeling Kernel***, this method computes first the intersections between the 2D abstract geometry of all the **Floor Assemblies** in the **Structural System** and the 3D volume defined by the **Structural Zone**. These intersections are 2D planes. The method considers the **Floor Assembly** at the bottom of the **Structural Zone** but not the one at the top, since the latter carries loads from a **Structural Zone** above. Then, the method calculates the total area from those intersections. Finally, the "*totalGravityLoad*" attribute is calculated using method "*computeTotalGravityLoad*" which multiplies the "*gravityLoadIntensity*" by the "*loadedSurface*". The gravity load is directly applied on **Slab Elements** and from there it propagates to the rest of the **Structural System**. However, this research project does not consider load propagation and load modeling. In the current development, applied gravity loads are assigned manually by the engineer to **Structural Zones**, depending on the functionality of **Spaces**. This feature can be easily implemented through a simple table look-up.

Structural Zones also incorporate a method that searches for **Structural Layout Constraints** within the **Spaces** that it groups. Such constraints are then used for restricting structural layouts by the engineer using ***Configuration Algorithms*** (cf. section 5.3.2). The example in Figure 5.22 shows three alternative **Structural Layout Constraints** assigned by the architect to a **Space** with the subsequent effect on the support hierarchy of the roof, which is a **Floor Assembly**. For the sake of clarity, it considers a **Structural Zone** related to a single **Space**. In the first case (Figure 5.22a),

there is no **Structural Layout Constraint** imposed to the **Space** and therefore, depending on the engineer's intent a "one-level" spanning **Floor assembly** (see Figure 5.6) can be used with **Slab Elements** directly supported by **Columns**, or a "two-level" **Floor Assembly** with **Slab Elements** supported by **Beams** and the **Beams** supported by **Columns**. As shown in Figure 5.22a, the latter **Floor Assembly** was selected. In the second case (Figure 5.22b), the **Space** must be column-free therefore, a "three-level" roof is required, with a slab supported by short **Beams**, which are in turn supported by longer **Girders** resting on **Columns**. In the third case (Figure 5.22c), the **Structural Zone** has an **Alignment Constraint** and therefore, the roof can either be configured as in Figure 5.22b, if the engineer decides not to make use of the constraint, or as illustrated in Figure 5.22c depending on the engineer's intent.

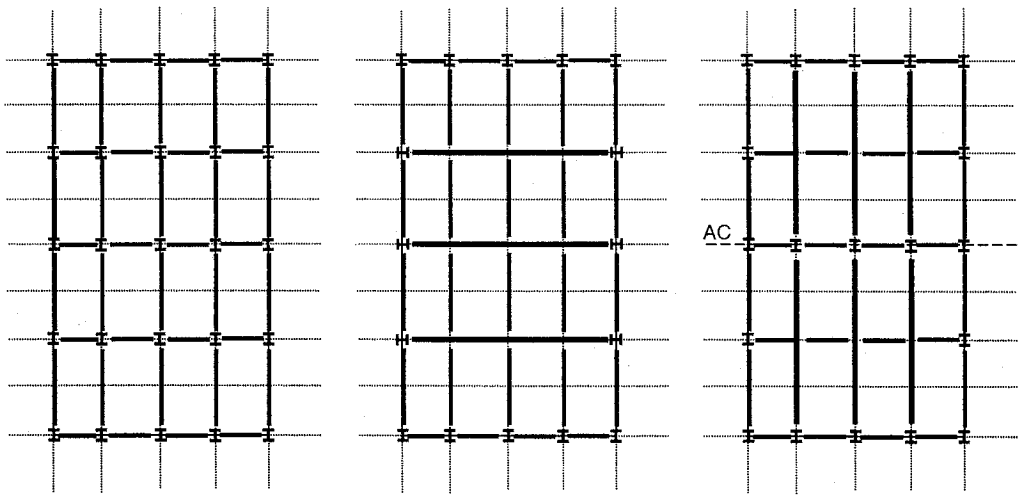


Figure 5.22 Effect of **Structural Layout Constraints** on **Floor Assembly** type

Deciding on the type of **Floor Assembly** depends not only on the amount and location of vertical supports but also on the intensity of the applied load and other architectural restrictions. **Structural Zones** incorporate two more methods for satisfying

architectural restrictions in the selection of **Floor Assemblies**. The method "*checkFloorToCeilingHeight*" is used for verifying the allowable floor-to-ceiling height associated with the function of the **Spaces** grouped within a **Structural Zone**. This height is then subtracted from the height of the corresponding **Storey** (which is a floor-to-floor height) in order to get the available floor depth. Alternatively, the architect may specify the required floor depth, in which case the method "*checkCeilingDepth*" is used. However, in this research project load intensities and allowable **Floor Assembly** depths are not explicitly included in the selection and layout of **Floor Assemblies**. They only provide information to the engineer so that he/she can decide which **Floor Assembly** to use based on his/her own knowledge. Including these factors as explicit knowledge in the ***Integrated Representation*** requires the definition of specific (cf. sections 5.2.1.2 and 5.2.2.6) **Floor Assemblies** with associated heuristic thresholds of applicability.

A main concern during the structural synthesis process that comes from the variation of programmatic requirements in the architecture is the breakdown in the continuity of the **Structural System** that is likely to happen (cf. sections 2.2.3.2 and 4.3.2.1). This affects mostly the vertical continuity of **Wall Stacks** and **Column Stack.** Figure 5.23 shows six possible cases how vertical continuity of **Walls Stacks** and **Column Stacks** could be interrupted. Case (a) shows a **Wall Stack** that is continuous down to an intermediate level where its gravity load is directly taken by **Columns Stacks** underneath; case (b) shows **Column Stacks** that are continuous down to an intermediate level where their gravity load is taken directly by a **Wall Stack** underneath; cases (c) and (d) show **Wall Stacks** and **Column Stacks** that do not reach the roof level due to

column-free and wall-free spaces in the **Stories** above, therefore they start taking loads at an intermediate level; cases (e) and (f) show **Wall Stacks** and **Column Stacks** interrupted at an intermediate level where their gravity load is taken by a **Floor Assembly** that transfers the loads to **Column Stacks** in the lower **Stories**. The ***Integrated Representation*** can describe all cases. However currently, synthesis algorithms can handle only cases (a) through (d). Cases (e) and (f) require the provision for a transfer structure or transfer floor which has not been considered in the present study.

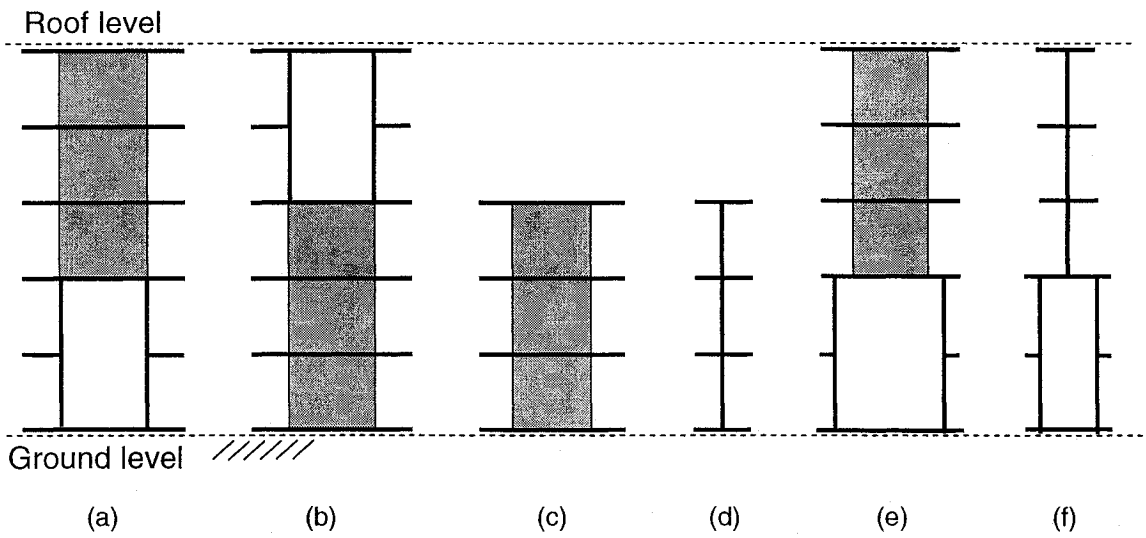


Figure 5.23 Vertical continuity of **Wall Stacks** and **Column Stacks**

5.2.3 Limitations of the *Integrated Representation*

While the organization of a design assistance environment for conceptual structural design focuses on supporting the methodology A/S integration as described in section 4.4.1, it carries inherent limitations that originate from the nature of this research project. These limitations include limitations to the breadth of the scope that restrict its

comprehensiveness for improved support, and limitations to the depth of the scope that relate to features that are essential for providing complete support but are not included due to time constraints. These limitations are described in the following sub-sections.

5.2.3.1 Limitations to the breadth of the scope

The proposed design representation is considered to be an initial step towards the integration of architectural concerns to the process of conceptual structural design. Hence, it has its limitations which have been described in section 1.5.2 and are repeated below for clarity:

- ***The building architecture*** - The representation considers a storey to be bounded at the bottom by a single floor slab and at the top by a single ceiling slab, which is not always the case. In modern architecture, stories are often defined by many floor and ceiling slabs at varying elevations from the ground; these are called multi-level stories. Therefore, the representation does not support the design of multi-level stories.
- ***The structural system*** – The representation considers only structural systems where the structural elements that belong to the vertical support system are continuous down to the ground. Therefore, it does not accept transfer structures where vertical load paths are interrupted at intermediate stories and re-directed to different vertical supports in the lower stories (see Figure 5.23 e and f). Therefore, dissimilar structural patterns are not accepted between adjacent structural zones. As a consequence, level 2 of functional integration between the architecture and the structure (cf. section

4.3.2.1) is not supported by the current design assistance environment for conceptual structural design.

Overcoming the above limitations requires further development work. Nevertheless, it is expected that the proposed ***Integrated Representation*** will simplify the synthesis process by making it more efficient and less error-prone, for the design of most types of building that are being designed nowadays.

5.2.3.2 Limitations to the depth of the scope

Complete support for A/S integration will be provided if the representation incorporates the following features:

- ***Specific structural assemblies*** - The representation should be extended to describe specific structural assemblies that incorporate construction technologies, materials and applied loads in support of decision-making.
- ***Grouping*** - The representation should enable grouping structural elements together based on similarities (e.g. function, material, geometry and position). As discussed in section 2.2.3.4.a) grouping structural elements leads to more efficient structures and simplifies the synthesis process, since it allows the engineer to deal with sets of entities with similar characteristics instead of dealing with the individual entities.

The above limitations imply augmenting the ***Integrated Representation***. Given the object-oriented nature of the representation, incorporating specific structural assemblies and grouping capabilities to it is expected to be a straightforward task.

A knowledge-based component is required for assisting the engineer in selecting specific structural assemblies and grouping structural elements. As discussed in section 3.3.1, the investigation of such a component has been left for future research. The next section describes the main ***Synthesis Algorithms*** that have been developed in this project.

5.3 *Synthesis Algorithms*

Synthesis Algorithms rely on the capabilities of the ***Geometric Modeling Kernel*** and those of the ***Integrated Representation*** for assisting the engineer in reasoning from a geometric model of the building architecture and the structural system and integrating structural solutions. Such algorithms use the partial or complete geometry and topology of the design model for constructing new geometry and topology, and for verifying the model. ***Synthesis Algorithms*** are meant to complement the engineer's own knowledge and experience for architecture/structure (A/S) integration (cf. section 4.3.2) by facilitating the process of devising and configuring continuous load paths to the ground which is the main objective of the conceptual stage of structural design (cf. section 2.2.2.2 a). More specifically, geometric algorithms facilitate the tasks of identifying potential structural problems and opportunities in the building architecture (inspection), laying out structural solutions as part of the building architecture (configuration), and verifying such structural solutions (verification).

Consequently, according to the type of activity to be supported (cf. section 4.3.2), ***Synthesis Algorithms*** have been classified in three groups: (1) inspection algorithms, (2) configuration algorithms, and (3) verification algorithms. The following sections present representative examples of each group. Appendix C presents a list of ***Synthesis***

Algorithms that have been implemented in this research project. These algorithms demonstrate the kind of functionality that is required to assist in A/S integration. However, it is expected that more powerful algorithms can be implemented in the future. For conciseness, only a schematic explanation of the algorithms is given in this chapter. Details are given in the Appendix C where the algorithms are described using flowcharts and pseudo-code. In general, all the ***Synthesis Algorithms*** search for structural supports in some way. Whenever a lack of support is detected by an algorithm, the engineer is notified about the problem. Then, it is up to the engineer to take corrective actions.

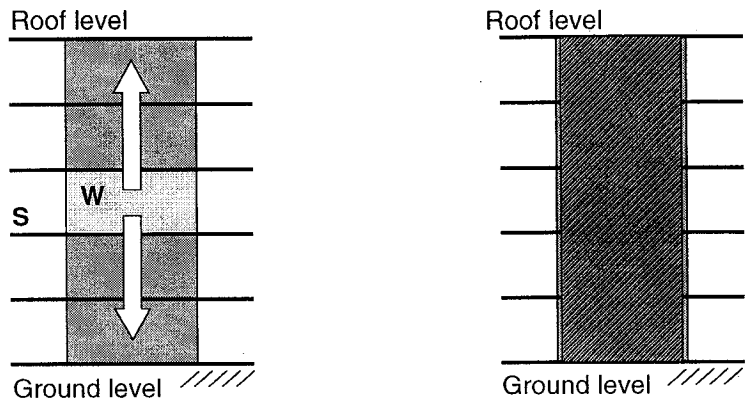
5.3.1 *Inspection Algorithms*

Two representative inspection algorithms have been selected for presentation in this chapter, namely: *verifyWallContinuity* and *findSupportsFromArchitecture*. As indicated by its name, the first algorithm verifies the continuity of a given **`Wall`** from the **`Architectural Model`**. The second algorithm finds potential structural supports by inspecting all the vertical elements (i.e. **`Walls`** and **`Columns`**) in the **`Architectural Model`**. The second algorithm makes use of the first algorithm for finding vertical supports.

5.3.1.1 Algorithm *verifyWallContinuity*

The algorithm *verifyWallContinuity* verifies the vertical continuity of a selected **`Wall`** "w" from any given **`Storey`** "s" within the **`Architectural Model`** (see Figure 5.24 a). In selecting a **`Wall`** from the architecture, the engineer anticipates that the selected **`Wall`** can adequately be integrated within a still envisioned **`Structural System`**. The algorithm

assumes that the **Wall** that has been selected by the engineer can be used for structural purposes (i.e. it is permanent and opaque). The algorithm searches for continuous vertical load paths within the **Stories** above and below the **Storey** "s" where the selected **Wall** "w" is located. The simplest case is where the **Walls** above and below are completely aligned (i.e. having the same geometry and same position in the horizontal plane) with the given **Wall** "w" as illustrated in Figure 5.24 a). Figure 5.24 b) shows the load path with a diagonal pattern. The **Walls** illustrated in Figure 5.25 can also be considered continuous. However, they do not provide a continuous gravity load path to the ground.



(a) Simple vertical **Wall** arrangement (b) Simple load path

Figure 5.24 Simplest case of vertical **Wall** continuity

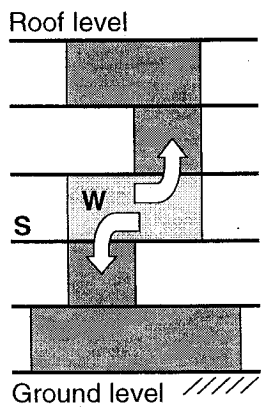
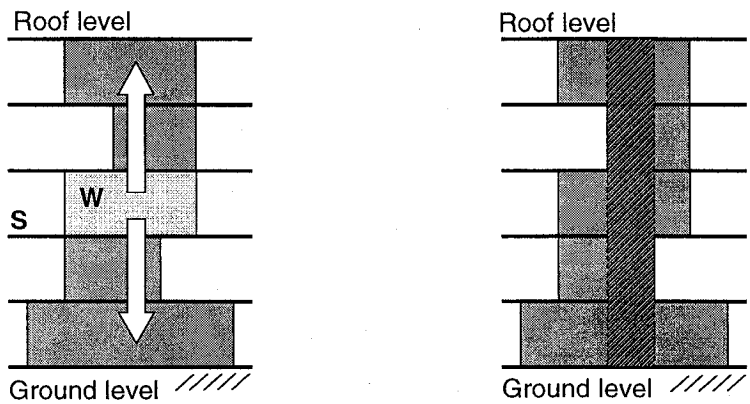


Figure 5.25 Vertical continuity without continuous gravity load paths to the ground

Figure 5.26 illustrates the algorithm *verifyWallContinuity*. It shows a more general vertical Wall arrangement (Figure 5.26 a). From such an arrangement, the algorithm finds a vertical wall-strip that defines a continuous gravity load-path to the ground (Figure 54.26 b). This load-path is basically obtained through a Boolean intersection between the geometry of the given **`Wall`** "w" and the geometry of each intersecting **`Walls`** above and below.



(a) General vertical **`Wall`** arrangement (b) Continuous gravity load path

Figure 5.26 illustration of algorithm *verifyWallContinuity*

The algorithm takes as inputs a wall-segment which is a portion of **`Wall`** "w" that is selected by the engineer for testing vertical continuity, and a list of **`Stories`** from the **`Architectural Model`** sorted in ascending order. A wall-segment is a planar face that is a subset of the geometry of a **`Wall`** which is required because the engineer may want to use only a portion of a **`Wall`** for structural purposes. However, if the engineer decides to select an entire **`Wall`** for continuity testing, then the wall-segment is simply the geometry of the selected **`Wall`**. The algorithm returns a list of continuous **`Walls`** and a continuous gravity load-path, which is represented as a geometric entity called "wall-strip". If the load paths are narrowed in some **`Stories`**, the algorithm also returns a list of these

Stories. The list of **Stories** where the load-path is narrowed is used by the engineer for checking with the architect if the **Wall** can be widened in those **Stories**. If the engineer accepts a load-path, then the computer executes the post-processing algorithm *divideArchitecturalWalls* to get the geometry of **SWalls** from the wall-strip and generate a **Wall Stack**.

The main limitation of the algorithm is that it does not consider **Openings** in **Walls** for continuity. A more general algorithm is required that considers dissimilar **Opening** sizes and locations at each **Storey** within continuous **Walls**.

5.3.1.2 Algorithm *findSupportsFromArchitecture*

Algorithm *findSupportsFromArchitecture* makes use of the algorithm *verifyWallContinuity* (as described above) as well as the algorithm *verifyColumnContinuity* for searching for vertical supports from the **Architectural Model**. The search for vertical supports takes place by inspecting each **Wall** and each **Column** (if placed by the architect) at the lowest **Storey**. Considering **Walls**, the algorithm checks first if a **Wall** being tested for continuity is permanent and opaque; if this is the case, then the algorithm tests its continuity by searching for matches at each consecutive **Storey** until the top of the building is reached or the **Wall** is interrupted at an intermediate **Storey**. In this case, a **Wall** is considered continuous up to that upper **Storey**. Once the search for **Walls** is finished, **Columns** placed by the architect are tested for continuity. While inspecting **Columns**, the algorithm also asks each **Column** whether it is intended to be structural or simply decorative. All the **Columns** that are intended (i.e. by the architect) to be structural are tested for continuity by the algorithm. At the end, for

each continuous **Wall** the algorithm returns a gravity load-path represented by a planar wall-strip, and for each continuous **Column** it returns a **Column** gravity load-path linear column-strip. Both, the wall-strip and the column-strip are geometric entities. Then the engineer can select any load-path at will so that **SWalls** and **SColumns** are generated with their corresponding **Wall Stacks** and **Column Stacks**. In this case, the post-processing algorithm *divideArchitecturalWall* is also executed when applicable for obtaining the geometry of each **SWall** when it is a subset of the geometry of a corresponding **AWall**.

The above inspection algorithms illustrate how the reasoning by the engineer using an **Architectural Model** can be supported while looking for gravity load paths to the ground. More powerful algorithms can be devised for assisting the search not only for gravity but also for lateral load paths to the ground in the building architecture. It is envisioned that these algorithms could check the amount and configuration of potential structural **Walls** for proper lateral resistance. Such algorithms are left for future research.

5.3.2 *Configuration Algorithms*

As described in section 4.3.2, the process for the creation of the integrated model of a **Building** assumes that the architect creates an **Architectural Model** first, and once this model is complete, the engineer integrates the **Structural System** to it. After the structural inspection of the **Architectural model**, ***Configuration Algorithms*** assist the engineer in the configuration of the **Structural System** as part of the building architecture. Most ***Configuration Algorithms*** operate at the global level (i.e. generating configurations that span an entire **Independent Structural Volume**). However, few of them operate at the local level (i.e. generating configurations that span a **Structural**

`Zone`). Algorithms that operate at the global level are executed first, followed by local algorithms that complement the synthesis task. In order to better understand the role of the ***Configuration Algorithms*** in the top-down synthesis process, it is convenient to simplify this process as an instantiation (i.e. creating instances) of entities from the ***Integrated Representation*** in a top-down fashion.

5.3.2.1 Simplified top-down design process

Considering the conceptual design process as a top-down instantiation of entities from the ***Integrated Representation***, the steps of the process are summarized in two views (i.e. the architectural and the structural views) as follows.

Architectural view (see Figure 5.8)

1. The architect defines the `Site` with its name and geometry.
2. The architect defines the `Building` associated to the `Site`, along with the `Architectural Model`. In doing so, the architect defines the building type (office, apartment, mixed-use, etc.) and the number of `Stories`.
3. The architect defines a reference `Grid` for the project.
4. The architect defines each `Storey` with its name and elevation above ground.
5. For each `Storey`, the architect configures `Spaces` (i.e. `Single-storey Primary Spaces` and `Multi-storey Primary Spaces`), draws `AWalls` (with their `Openings`), and places `AColumns` tentatively. The architect may also create typical `Stories` by copying already created ones.

6. When a **Storey** layout is complete the computer generates the geometry of the **ASlabs** (i.e. the architectural view of the **Floor Slabs**).

7. The architect defines **Secondary Spaces** (**SS**) as groupings of **SSPS** and/or **MSPS**

8. The architect poses **Structural-Layout Constraints** in some **Spaces** to restrict structural layouts inside them.

Structural view (see Figure 5.8)

Before initiating the structural configuration process, the engineer inspects first the **Architectural Model** globally and locally in a search for gravity and lateral load-paths, as well as potential structural difficulties. In doing so, s/he uses ***Inspection Algorithms*** to verify continuity and select **Walls** and **Columns** from the **Architectural Model** (cf. section 5.3.1). **Structural Layout Constraints**, as imposed by the architect, are also verified. At this stage, the engineer also may also suggest global and/or local changes to improve the structural properties of the building architecture. Once an agreement has been reached the configuration process is carried out as follows:

1. The engineer defines the **Structural System** associated with the **Building** in terms of material(s) and overall gravity and lateral load resisting strategy (e.g. devise a concrete structure with flat slabs resting on columns for gravity-load resistance and shear walls for lateral-load resistance).

2. The engineer defines **Independent Structural Volumes** by grouping **Primary Spaces** or **Secondary Spaces**.

3. The engineer defines **Structural Zones** by grouping **Secondary Spaces** and/or **Primary Spaces**.

4. The engineer defines **Structural Subsystems**: **Horizontal Subsystem**, **Vertical Gravity Subsystem** and **Vertical Lateral Subsystem**. **Vertical Lateral Subsystems** are given a direction (e.g. North-South, East-West).

5. The engineer selects the types of **Frame Assemblies** to be used in each principal direction (**Moment-Resistant Frame**, **Simple-Gravity Frame**, etc.). Then, he/she lays out **Frame Assemblies** in each principal direction, thus implicitly defining structural gridlines and structural bays.

6. The engineer defines **Floor Assemblies** from **ASlabs**. The engineer may generate various **Floor Assemblies** from a single **ASlab** if required by the **Structural Zones** where they span.

7. The computer defines **Wall Stacks** and **Column Stacks** by using the **Wall** load-paths and **Column** load-paths as produced by algorithms *verifyWallContinuity*, *verifyColumnContinuity* and *findSupportsFromArchitecture*. **Wall Stacks** and **Column Stacks** are automatically integrated to the corresponding **Vertical Lateral** and **Vertical Gravity** subsystems (algorithms: *generateWallStack* and *generateColumnStack*).

8. The computer generates the abstract geometry of each **Frame Assembly** (a vertical plane) which spans the entire **Independent Structural Volume** where it belongs. Each **Frame Assembly** is automatically added to its appropriate **Vertical Lateral** subsystem (algorithm: *generateFrameAbstractGeometry*).

9. The computer generates the abstract geometry of each **Floor Assembly**, which is a horizontal plane that extends either throughout an entire **Independent Structural**

Volume or **Structural Zone** if structurally required. It automatically integrates each **Floor Assembly** to the **HS** (algorithm: *generateFloorAbstractGeometry*).

10. The computer adds **Wall Stacks** and **Column Stacks** to coplanar **Frame Assemblies.** These **Wall Stacks** and **Column Stacks** become sub-assemblies of **Frame Assemblies**. However, not all **Wall Stacks** and **Column Stacks** need to belong to a coplanar **Frame Assembly** (algorithms: *wsIntFrame* and *csIntFrame*).

11. The computer intersects orthogonal frames to produce **Column Stacks**. Then **Column Stacks** become sub-assemblies of **Frame Assemblies**. Before generating **Columns** and **Columns Stacks** the computer verifies if they already exist (i.e. when created by the architect and selected by the engineer after inspection). Also, before generating **Columns** within **Column Stacks**, affected **Structural Zones** are consulted to determine if they carry **Structural Layout Constraints** that may restrict the placement of **Columns** (algorithm: *frameXframe*).

12. The computer intersects **Frame Assemblies** and **Floor Assemblies** to produce primary **Beams**. Primary **Beams** are automatically connected to **Columns** belonging to **Frame Assemblies**. Before creating primary **Beams** the computer checks for span thresholds and conflicting openings in the **Floor Assembly** (algorithm: *frameXfloor*).

13. The computer creates local floor layouts locally for structurally constrained **Structural Zones** where **Structural Layout Constraints** restrict the layout of **Columns** and **Beams** (algorithm: *uniformDepth*).

14. For each **Floor Assembly,** the computer creates **Slab Elements** (algorithm: *generateSlabElements*).

As indicated above, ***Configuration Algorithms*** carry out the steps numbered 7 to 14. In this way as discussed in section 4.4.1, the engineer is in charge of making decisions such as selecting structural system type and material(s), as well as subsystems and assemblies, and positioning them, while the computer takes care of time consuming tasks, such as arranging and connecting elements into assemblies, under the engineer's guidance and supervision. For illustration purposes the last three algorithms are described in detail in the following sub-sections.

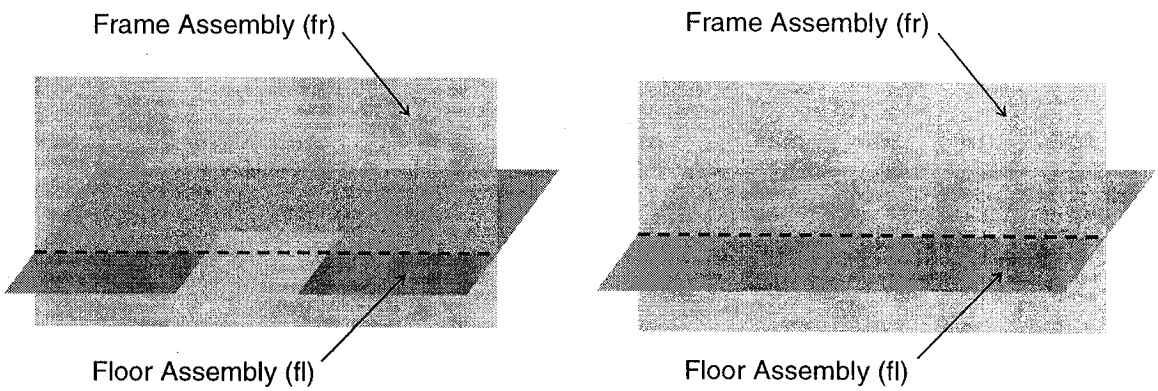
Note that the role of the **`Structural Subsystems`** is not mentioned in the following description of the algorithms for the sake of clarity. However, ***Configuration Algorithms*** always begin the search for **`Structural Assemblies`** and **`Structural Elements`** within the corresponding **`Structural Subsystem(s)`** where they belong. For example, when looking for a **`Floor Assembly`**, algorithms start searching in the **`Horizontal Subsystem`**, while when looking for vertical supports, algorithms start searching in the **`Vertical Gravity Subsystem`**, and when looking for lateral resisting elements, algorithms start searching the **`Vertical Lateral Subsystem`**.

5.3.2.2 Algorithm *frameXfloor*

The algorithm *frameXfloor* receives as inputs a **`Frame Assembly`** "fr" and a **`Floor Assembly`** "fl", intersects their abstract geometries (i.e. a vertical plane for **`Frame Assembly`** "fr" and a horizontal plane for **`Floor Assembly`** "fl") and creates primary **`Beams`**. This algorithm is executed after ***Configuration Algorithms*** *wsIntFrame* (adds **`Wall Stacks`** to **`Frame Assemblies`**), *csIntFrame* (adds **`Column Stacks`** to a **`Frame Assemblies`**), and *frameXframe* (intersects any two **`Frame Assemblies`** to get **`Columns`**).

Therefore before execution, in addition to the abstract geometry, the **Frame Assembly** "fr" includes **Column Stacks** and possibly **Wall Stacks** with **Columns** and **Walls** connected through **Node Connections** and **Line Connections**. By contrast, the **Floor Assembly** "fl" incorporates only an abstract geometry. At the end of this algorithm, newly created primary **Beams** are added to both the **Frame Assembly** "fr" and the **Floor Assembly** "fl".

Any two non-parallel planes always intersect at a line. If the planes have openings, then the intersection may become a collection of line segments. Slab openings are created by vertical ducts, shafts, staircases, lobbies, etc. As discussed in section 5.2.1.1, all these are called **Multi-Storey Primary Spaces** (**MSPSs**). The algorithm *frameXFloor* ignores openings initially, so that the intersection between the abstract geometry of a **Frame Assembly** and that of a **Floor Assembly** is always a single line, which is shown as discontinuous line in Figures 5.27 (a) and 5.27 (b).



(a) **Floor Assembly** with one opening (b) **Floor Assembly** with no openings

Figure 5.27 Intersection between a **Frame Assembly** and a **Floor Assembly**

Since in addition to the abstract geometry **Frame Assemblies** already incorporate **Column Stacks** and **Wall Stacks** with their connections, the algorithm generates **Beams**

along the intersection line between two consecutive **Node Connections**. Once all **Beams** have been created they are added to the intersecting **Frame Assembly** "fr" and **Floor Assembly** "fl". However, before actually creating **Beams** two main conditions are verified.

The algorithm first checks that there is no opening interfering in the layout of the **Beam.** If there is an opening, the algorithm asks the **MSPS** that generated the opening if a **Beam** is accepted in the intended position. Figure 5.28 shows a **Frame Assembly** and a **Floor Assembly** with an opening that interferes at their intersection. The **Frame Assembly** consists of an abstract geometry and **Columns Stacks** (generated with *frameXframe* algorithm). The **Floor Assembly** consists of an abstract geometry only. In Figure 5.28 (a) the opening is generated by the **Multi-Storey Primary Space** called "MSPS1" that does not accept any **Beams** inside. In Figure 5.28 (b) the opening is generated by three **Multi-Storey Primary Spaces**, namely: "MSPS1", "MSPS2" and "MSPS3". "MSPS1" accepts **Beams** inside, "MSPS2" does not accept any **Beams** inside, and "MSPS3" accepts **Beams** inside. In Figure 5.28 (c) there is only one MSPS called "MSPS1" that accepts **Beams** inside.

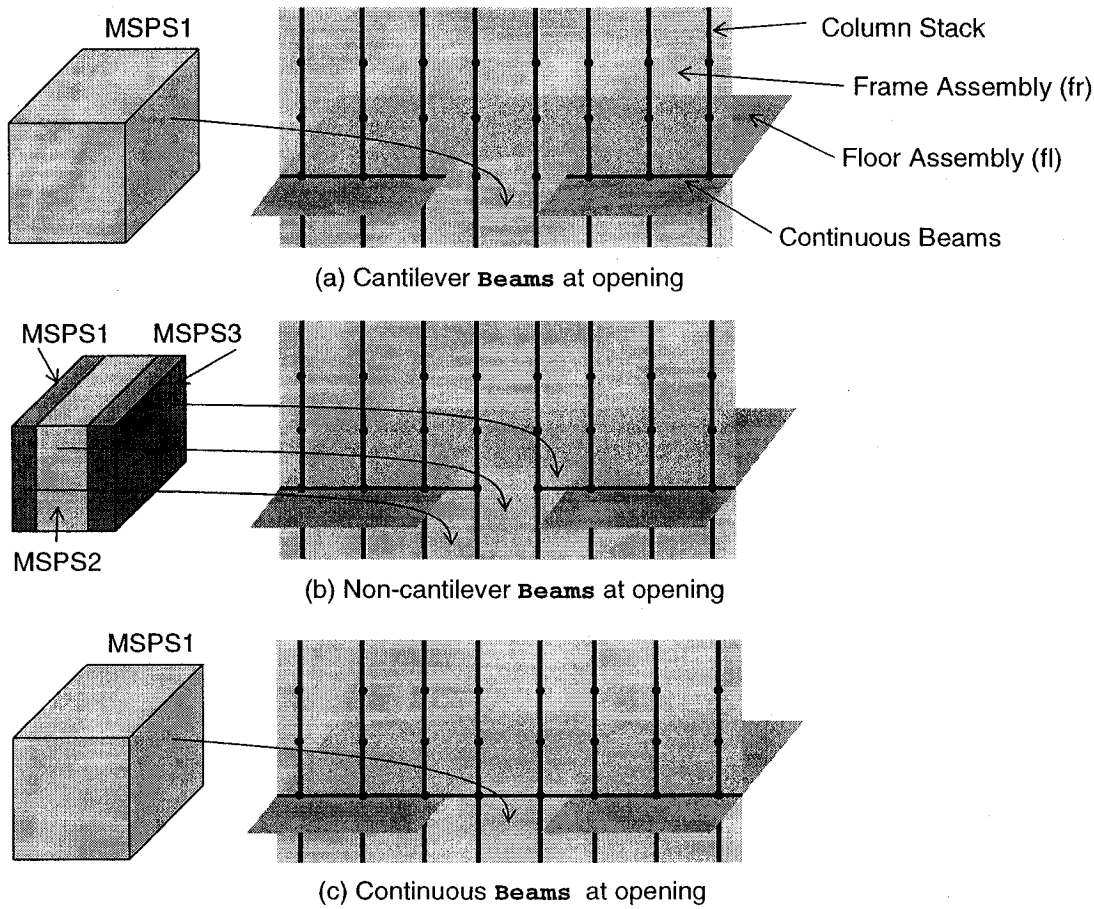


Figure 5.28 Alternatives for **Beam** generation when a slab has openings

The algorithm also verifies if the **Floor Assembly** "fl" is constrained by a **Structural Zone** "sz". In the previous example the **MSPSs** could be defined as **Structural Zones**. However, **Structural Zones** play a broader role in the configuration process since they can be used, for example, to group **Spaces** where vertical supports (e.g. **Columns**) are restricted via **Structural Layout Constraints**. In such cases, the layout and type of the supported **Floor Assemblies** is also affected at the **Structural Zone**. If **Structural Layout Constraints** are present in a **Structural Zone**, the algorithm *frameXfloor* creates **Construction Lines** (cf. sections 5.2.2.1 and 5.2.2.6) spanning the entire **Structural Zone** instead of **Beams**. **Construction Lines** are used later by the

Configuration Algorithm *uniformDepth* (cf. section 5.3.2.3) for generating the layout of a structurally constrained **Floor Assembly** "fl". For example, in Figure 5.22 (a) all **Beams** are created by the algorithm *frameXFloor*. In Figure 5.22 (b) no **Beams** are created by this algorithm, it creates instead **Construction Lines** in the principal directions of the length of the full **Space**. In Figure 5.22 (c) the only **Beams** that are created follow the alignment given by the **Line Constraint,** the rest is filled with **Construction Lines**.

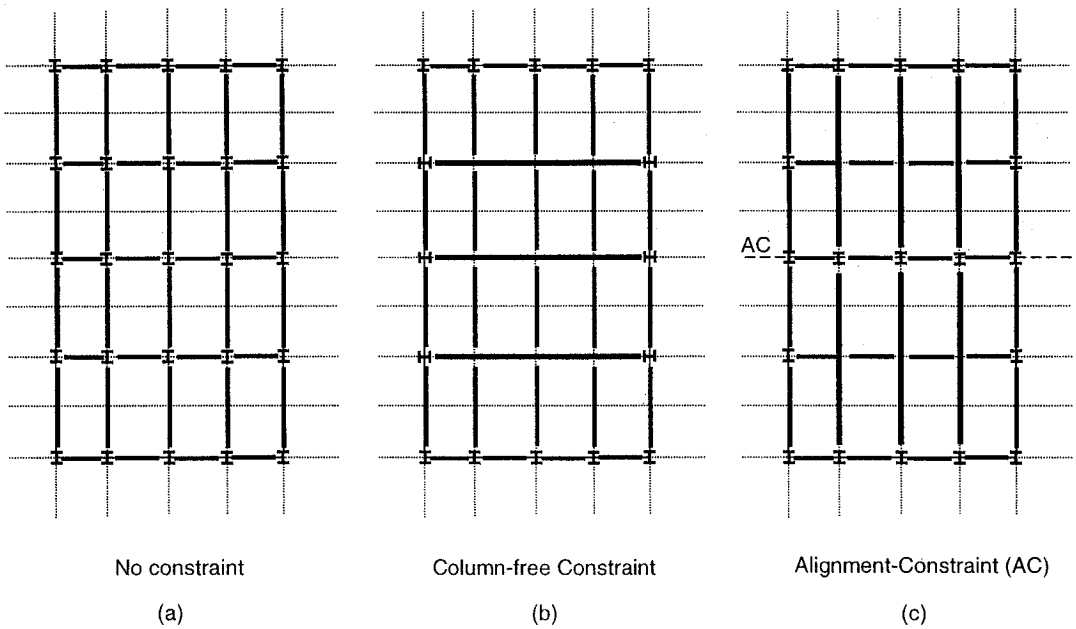


Figure 5.22 (repeated) Effect of **Structural Layout Constraints** on the hierarchy of **Floor Assemblies**

Therefore, when a **Floor Assembly** is inside or at the top of a constrained **Structural Zone**, algorithm *frameXfloor* may result in: (1) **Beams** that are incorporated to the corresponding **Floor Assembly**, (2) **Beams** and **Construction Lines** both passed to **Floor Assembly**, (3) a list of **Construction Lines** which is passed to **Floor Assembly**. **Construction Lines** provide a pattern with tentative locations for primary and secondary **Beams**. Typically, when a **Beam** is added to a **Frame Assembly**, the **Frame Assembly** takes care of connecting it properly to its supports, which may be **Columns**

and/or **Walls**. However, in some situations, for example in large open spaces, **Floor Assemblies** are the ones that take care of generating, configuring and supporting **Beams**. In any case, after a **Beam** is added to a **Floor Assembly**, the **Floor Assembly** verifies if it is properly supported and its length is within the user-specified thresholds. The algorithm *verifyBeamSupports* verifies that a **Beam** has at least two supports or one fixed support (i.e. that restricts rotation about "z" axis) as described in section 5.2.2.2.a, and that its span length is within the thresholds specified by **Floor Assembly** "fl". If a newly incorporated **Beam** is not properly supported, algorithm *findBeamSupports* searches for supporting primary **Beams** within the **Floor Assembly** "fl" or **Columns** and **Walls** within the **Storey** below.

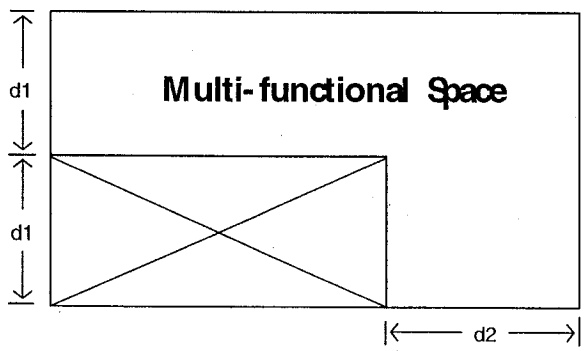
5.3.2.3 Algorithm *uniformDepth*

This algorithm generates floor framing layouts for **Floor Assemblies** located within or above structurally constrained **Structural Zones**. This is a post-processing algorithm executed after the *frameXfloor* algorithm generates **Beams** and/or **Construction Lines** depending on the available supports. As indicated by its name, the algorithm is based on the structural principle that long, lightly loaded secondary **Beams** delivering their loads to shorter primary **Beams**, or **Girders**, leads to uniformity of structural depth (Shaeffer 1998).

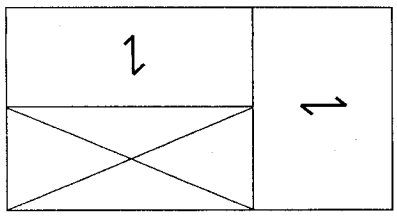
Construction Lines provide a pattern used by the algorithm for generating primary and secondary **Beams**. **Construction Lines** are given in two lists, one for each orthogonal direction. The algorithm first groups the shortest **Construction Lines** and makes them primary **Beams**, then it partitions each **Construction Line** intersecting the smallest lines

and for each partition it creates secondary **Beams**. The newly created **Beams** are incorporated into the **Floor Assembly** "fl" and the corresponding **Frame Assembly** that take care of properly supporting them. Once a **Beam** is created its corresponding **Construction Line** is deleted. Finally, the algorithm recursively calls itself. The recursive call is necessary for the cases when **Alignment Constraints** produce an implicit division of **Spaces** into "sub-spaces" with a resulting division of the **Floor Assembly**. For each primary **Beam** that is created the algorithm finds a **Frame Assembly** where it belongs. This is done through a containment operation between the geometry of the given **Beam** (which is a line) and the abstract geometry of each **Frame Assembly** (which is a vertical plane). If the geometry of a **Beam** is contained within the abstract geometry of a **Frame Assembly** then the **Beam** belongs to that **Frame Assembly**.

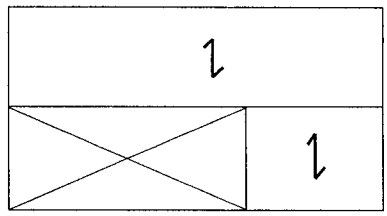
The example illustrated in Figures 5.29 through 5.32 describes how the algorithm *uniformDepth* configures the roof for a multi-functional **Space** with **Structural Layout Constraints** as posted by the architect. The roof framing layout could be done in two ways as illustrated in Figures 5.29 (b) and (c).



(a) A multi-functional Space



(b) Floor framing directions (first option)



(c) Floor framing directions (second option)

Figure 5.29 a **Space** and alternative roof framing options

Figure 5.30 shows the roof framing and the sequence of creation of primary **Beams** and secondary **Beams** when the **Space** has been defined as **Column Free** by the architect. It produces a framing layout with framing directions as illustrated in Figure 5.29 (b) as follows:

1. Select group of smallest **Construction Lines** and create primary **Beams** identified with the number (1).
2. Select **Construction Lines** in the orthogonal direction that cross the **Beams** already created. Partition these **Construction Lines** and create secondary **Beams** identified with number (2).
3. Select the next group of smallest **Construction Lines** and create primary **Beams** (3).
4. Select **Construction Lines** in orthogonal direction, partition them with already created **Beams** and create secondary **Beams** (4).
5. **Floor Assembly** and **Frame Assembly** take care of finding supports for each **Beam** just created.

In this case, the primary **Beam** identified as "B1" is heavily loaded. However, the algorithm has no knowledge of loads and cross-sections, and therefore it is up to the engineer to ascertain that the roof layout proposed is feasible.

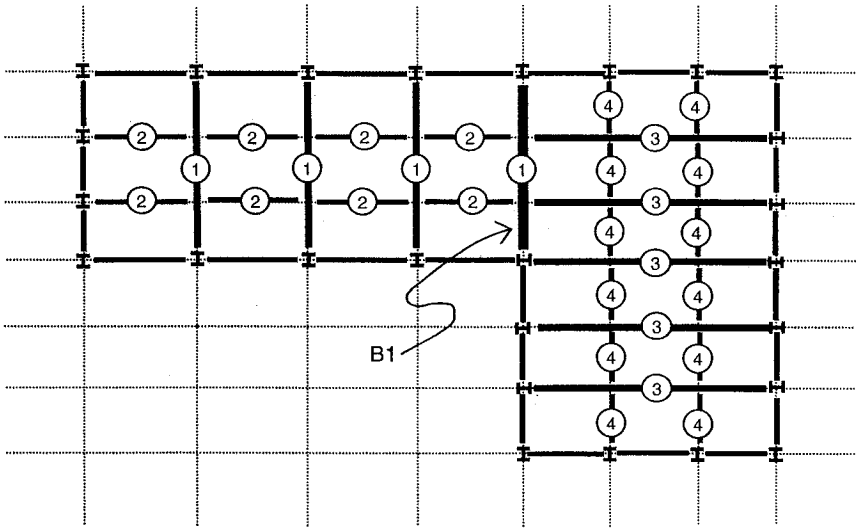


Figure 5.30 Roof framing layout for a **Column-Free Space**

Figure 5.31 shows another roof framing layout that corresponds to the framing directions illustrated in Figure 5.29 (b). The difference between roof layouts proposed in Figs. 5.30 and 5.31 is that an **Alignment Constraint** "AC" has been introduced by the architect in Figure 5.31 that allows **Columns** to be placed only within "AC". Therefore, the **Beams** identified with number (0) are created in advance by the algorithm *frameXfloor*. Then, given these **Beams** and **SColumns** at the alignment given by "AC" the sequence of creation of primary and secondary **Beams** is exactly the same as the one presented in Figure 5.30 but the results are better since heavily loaded **Beam** "B1" (Figure 5.30) is eliminated. The **Alignment Constraint** produces an implicit sub-division of the **Space** into two sub-spaces, however, the entire **Space** is no longer column-free.

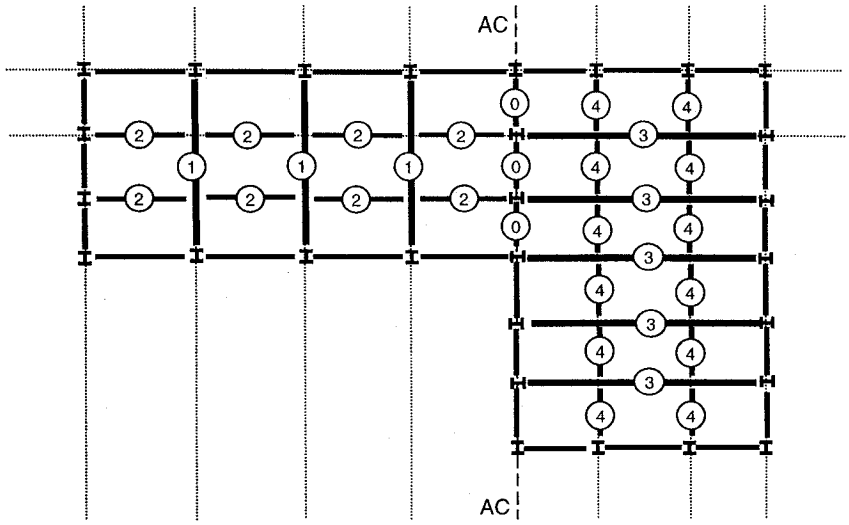


Figure 5.31 Roof framing layout for a **Space** with an **Alignment Constraint** "AC" (option 1)

Figure 5.32 shows a roof layout with framing directions as illustrated in Figure 5.29 (c). This floor framing was obtained by applying an **Alignment Constraint** "AC" in the orthogonal direction. Therefore, following the same sequence of creation the results are indicated in Figure 5.32. As in the previous case, the entire **Space** is no longer column-free.

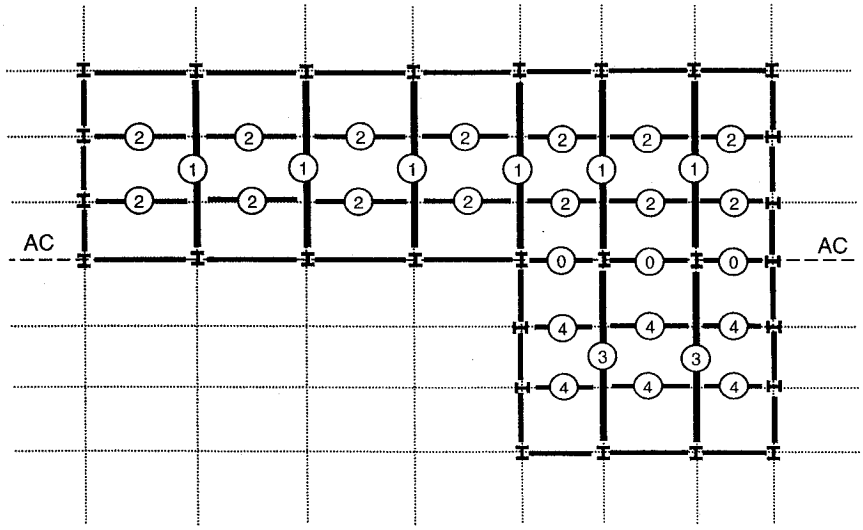


Figure 5.32 Roof framing layout for a **Space** with an **Alignment Constraint** "AC" (option 2)

In both cases illustrated in Figures 5.31 and 5.32 the architect implicitly defines two sub-spaces by specifying an **Alignment Constraint**. In both cases **Columns** at alignments "AC" are placed in anticipation by algorithm *frameXframe* following the intent of the architect. Then, algorithm *uniformDepth* configures roof layouts that incorporate those intents.

Algorithm *uniformDepth* demonstrates how **Floor Assembly** layouts can be generated locally whenever required by the functionality of **Spaces**. However, it needs to be improved in several aspects, as follows.

(1) The algorithm should let the engineer decide the floor framing direction(s). Then, following these directions it is straightforward for the algorithm to perform floor framing layout. For example, considering the dimensions "d1" and "d2" of the multi-functional **Space** dimensions "d1" and "d2" (Figure 5.29), it is likely that the engineer will opt for the framing directions indicated in Figure 5.29 (c) with corresponding layout illustrated in Figure 5.32. This is because this layout results in more uniform **Structural Elements**, (i.e. **Structural Element**s identified with (1) and (3) in Figure 5.32 have the same length).

(2) The algorithm needs to consider other factors that affect the floor framing layout such as the following: loads, material(s), cost, modularity and constructability. In its current state, the algorithm relies entirely on the knowledge and experience from the engineer for deciding if the **Floor Assembly** is able to resist the intended loads.

(3) The algorithm should also consider external factors, such as the layout of mechanical ducts and pipes, for performing floor framing layout.

5.3.2.4 Algorithm *generateSlabElements*

This algorithm receives a **Floor Assembly** "fl" as input and creates its **Slab Elements** by partitioning the abstract geometry of the **Floor Assembly** "fl" using the geometry of its primary **Beams** and **SWalls** from the **Storey** below. It uses any initial primary **Beam** "b" from Floor Assembly "fl" as a "seed" to create a wire-frame graph connecting all the primary **Beams** belonging to "fl". The wire-frame graph is then used to slice the abstract geometry (a horizontal plane) of the **Floor Assembly** "fl" and produce **Slab Elements**. Once **Slab Elements** are created from primary **Beams**, the algorithm searches for **SWalls** in the **Storey** below that may intersect some of the newly created **Slab Elements**. It then uses those **SWalls** to divide further the **Slab Elements** that they intersect and create new **Slab Elements**.

To generate the wire-frame graph from primary Beams the algorithm traverses the topology of Beams from the **Floor Assembly** "fl" and adds, through Boolean union, the geometry of each new **Beam** found to the geometry of the wire-frame graph. Thus, the wire-frame grows as the geometry of each new **Beam** is added. Once **Slab Elements** are created, the algorithm *verifySlabElementSupports* verifies that each **Slab Element** is supported at least by two **Beams** (that have been verified in advance to be properly supported) or a **Beam** and a **Wall**. It a **Slab Element** is not properly supported the algorithm *findSlabElementSupports* searches for proper supports. It looks for supports at the perimeter of the **Slab Element**, such as **Beams**, as well as **Walls** and **Columns** from the **Storey** below. It then creates **Line Connections** and/or **Node Connections** joining each **Slab Element** to its supports.

5.3.3 *Verification Algorithms*

The main requirement for the structural system during conceptual design is the configuration of a feasible and complete system of gravity and lateral load paths from the point of application of the loads down to the ground (cf. 2.2.2.2.a). ***Verification Algorithms*** take care of making sure that all the **`Structural Elements`** are adequately connected to provide continuous gravity and lateral load paths to the ground. In this research project, two main ***Verification Algorithms*** have been developed: algorithm *verifyGravityLoadpaths* and algorithm *verifyLateralLoadpaths*, which are described in the next sections. Further research is required to devise algorithms for quick and simplified verification of the stability of structural configurations.

5.3.3.1 Algorithm *verifyGravityLoadPaths*

The algorithm *verifyGravityLoadPaths* receives as input a **`Building Element`** "be" (cf. section 2.2.1.1.b) such as a **`Wall`**, a **`Column`**, a piece of furniture, or an appliance, and performs a search for supporting **`Structural Elements`**, starting with the **`Slab Element(s)`** that support (i.e. are in contact with) the **`Building Element`** directly, and ending at the **`Ground`**. The algorithm is illustrated in Figure 5.33.

Each **`Structural Element`** that is found in the search path is verified for supports. First, **`Slab Element`** supports are verified using algorithm *verifySlabElementSupports* introduced in section 5.3.2.4. Then, **`Beam`** supports are verified using algorithm *verifyBeamSupports* introduced in section 5.3.2.2. Finally, for support verification for **`Columns`** and **`Walls`** the algorithm simply traverses their topology down to the **`Ground`**. If the **`Ground`** is found, then the **`Building Element`** "be" is supported, otherwise it is not.

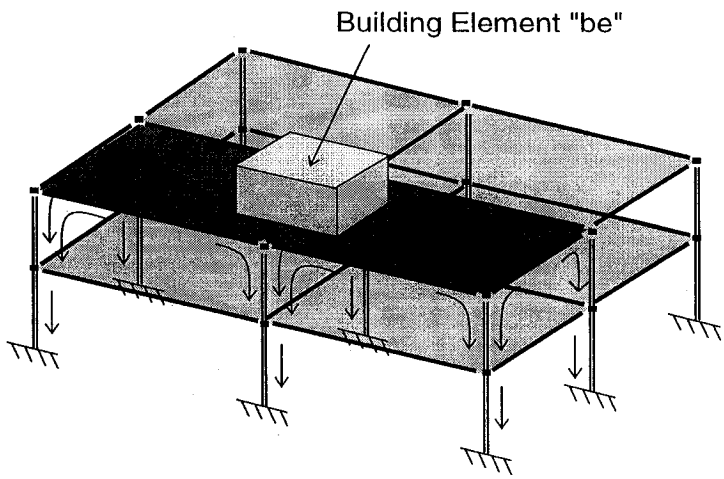


Figure 5.33 Algorithm *verifyGravityLoadPaths*

Note that the algorithm simply verifies load paths through low level geometric operations. Tributary areas are not considered and the actual load distribution in the supports is not considered either because it requires the use of engineering first principles governing redundant systems (unless the system is statically determinate). Nevertheless, existing computer applications for structural engineering already deal with gravity load distribution.

5.3.3.2 Algorithm *verifyLateralLoadPaths*

For lateral load path verification, **`Floor Assemblies`** are treated as rigid "diaphragms" (i.e. having negligible deformation) that transmit horizontal loads to vertical **`Structural Assemblies`** that can be **`Moment-Resistant Frames`**, **`Braced Frames`** or **`Wall Stacks`** or a combination of them (see Figure 5.34). Complete lateral load paths to the ground are provided if all building diaphragms are supported against movements in any two orthogonal directions and rotations about a vertical axis. These conditions are satisfied if all diaphragms are connected to at least three vertical **`Structural Assemblies`** that are

not all parallel and do not meet at a single point (Schodek 2003). In agreement with the ***Integrated Representation***, the algorithm assumes all vertical **`Structural Assemblies`** are always supported on the **`Ground`**.

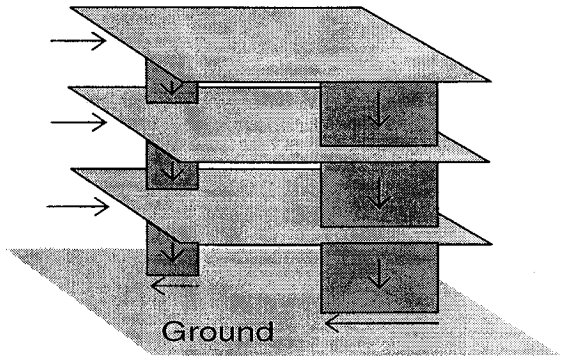


Figure 5.34 Algorithm *verifyLateralLoadPaths*

The algorithm finds first all vertical **`Structural Assemblies`** that can resist lateral load and lie or project within any of the two orthogonal directions. Then it verifies that the number of vertical **`Structural Assemblies`** is greater than three and that they are not parallel and do not meet at a single point. If this is the case, then the algorithm verifies that each **`Floor assembly`** "fl" is connected to the vertical **`Structural Assemblies`** just found. If a **`Floor Assembly`** "fl" is not connected to all the vertical **`Structural Assemblies`**, then the algorithm checks that it is connected to at least three not-parallel and not-intersecting vertical **`Structural Assemblies`**. If this is the case, then the **`Floor Assembly`** "fl" is supported against lateral loads. Otherwise, the **`Floor Assembly`** "fl" is not supported for lateral loads.

The algorithm *verifyLateralLoadPaths* relies exclusively on geometry and topology for verifying lateral load paths to the **`Ground`**. Therefore, it does not consider the actual stiffness of the lateral load resisting elements in the direction of the load. More advanced lateral-load resisting systems demand more complex lateral-load verifications. For

example, a "tube" system (or a building core) acts three-dimensionally by resisting lateral loads in one direction through the action of vertical **Structural Assemblies** in both orthogonal directions. Furthermore, in the "outrigger" system not all **Frame Assemblies** need to be supported for lateral loads at **Ground** level. Secondary **Frame Assemblies** at higher levels can be supported at intermediate levels by floor diaphragms which are in turn supported by primary **Frame Assemblies** connected to the ground. Nevertheless, such advanced lateral-load resisting systems are out of the scope of this research.

5.3.4 Limitations of the *Synthesis Algorithms*

Considering the methodology for architecture/structure integration (cf. section 4.4.1), the ***Synthesis Algorithms*** that have been developed provide support for all the activities as illustrated in Figure 4.8, except for activity 5.b which involves the initial selection of the cross-sectional dimensions of **Structural Elements**. Nevertheless, support for this activity has already been provided using artificial intelligence techniques (cf. section 2.4.1).

It is acknowledged that the ***Synthesis Algorithms*** that have been developed in this research project can be improved and more powerful algorithms developed. In particular, the following limitations need to be addressed:

(1) ***Inspection Algorithms*** need to be developed to verify in advance the amount and distribution of architectural **Walls** for resisting lateral loads, as well as other architectural irregularities that may have a negative impact on the quality of the structural solutions.

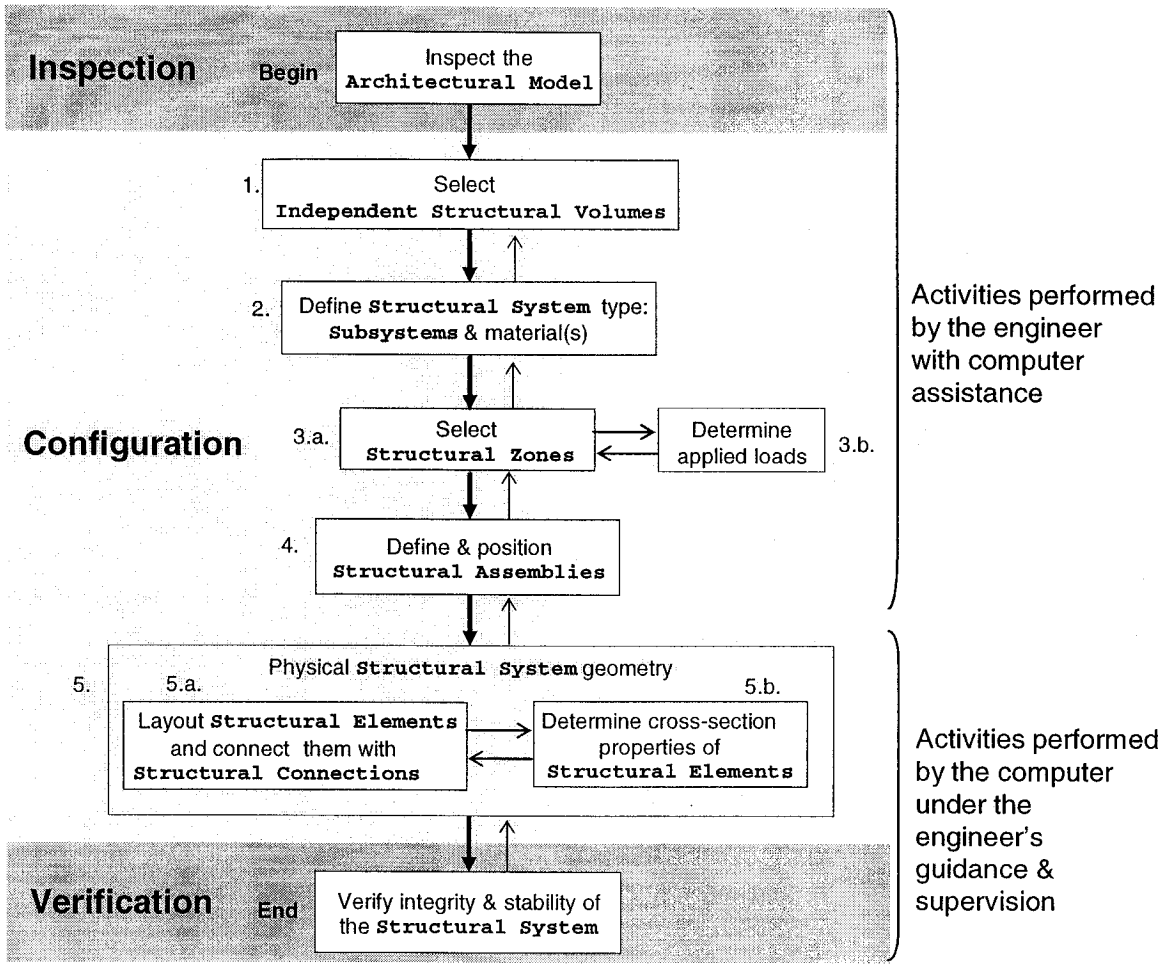


Figure 4.8 (repeated) Methodology for A/S integration during conceptual design

(2) Knowledgeable ***Configuration Algorithms*** need to be developed that combine geometry and topology considerations with knowledge about loads, materials, constructability and existing construction technologies.

(3) ***Verification Algorithms*** are required to verify structural stability and lateral load paths for advanced **Structural Systems** (i.e. the tube system), as well as for complex structural configurations.

5.4 Conclusions

This chapter has presented the planning and organization of two key components of a design assistance environment for conceptual design of building structures. These key components are an ***Integrated Representation*** of the `Structural System` and the `Building Architecture`, and ***Synthesis Algorithms***. First, the ***Integrated Representation*** describes concisely the structural and architectural entities as required for architecture/structure integration during conceptual structural design. Second, ***Synthesis Algorithms*** provide support that is still lacking in current structural engineering software applications, as they facilitate the engineering reasoning using geometry and topology for the inspection, configuration and verification of a model of the `Building Architecture` and the `Structural System`. The main limitations of the design assistance environment for conceptual structural design come from the ***Integrated Representation*** not being able to support level 2 of functional integration (cf. section 4.3.2.1), and the ***Configuration Algorithms*** that do not support levels 3 and 4 of physical integration (cf. section 4.3.2.2). Chapter 6 describes a software prototype that has been implemented to demonstrate the capabilities of the ***Integrated Representation*** and the ***Synthesis Algorithms*** for supporting architecture/structure integration during conceptual structural design.

CHAPTER 6

PROTOTYPE IMPLEMENTATION

A proof-of-concept software prototype has been implemented to demonstrate the validity of the methodology proposed in Chapter 4 and test the components of a design assistance environment for conceptual structural design developed in Chapter 5. The initial idea was to produce a fully interactive prototype to be tested by architects and engineers. However, this idea was later abandoned since developing a fully functional graphical user interface (GUI) would have demanded considerable efforts and resources. This is left for future developments. Therefore, in this chapter the person who executes the prototype is called "the user".

6.1 Prototype system architecture

The prototype has been implemented in Visual C++. The description that follows uses ***Bold, Italic*** typeface to identify the modules of the prototype. It consists of over fifty object-oriented classes developed in over 3000 lines of code. Figure 6.1 presents the prototype system architecture including the following modules:

- ***Alphanumeric User Interface*** – Allows the user to interact with the software prototype for generating the **`Architectural Model`** and integrating the **`Structural System`** into it.
- ***Top-down Design Manager*** – Keeps track of the sequence of design activities for enabling top-down design synthesis.

- ***Integrated Representation*** – As described in Chapter 5, this is the core component of the prototype which is used as a "template" to produce a computer model of the building integrating the **`Architectural Model`** and the **`Structural System`**.
- ***Geometric Modeling Kernel*** - The entire system is built on top of ACIS (Spatial Corp. 2004), which incorporates low-level data structures and algorithms to support mixed-dimensional geometric modeling.
- ***Synthesis Algorithms*** – As described in Chapter 5, these algorithms assist the engineer in synthesizing the **`Structural System`**. This assistance is based on the capabilities of the ***Integrated Representation*** and the ***Geometric Modeling Kernel.***
- ***Visualization interface*** – The commercial application HOOPS (TSA Inc. 2004) is used for 3D visualization.

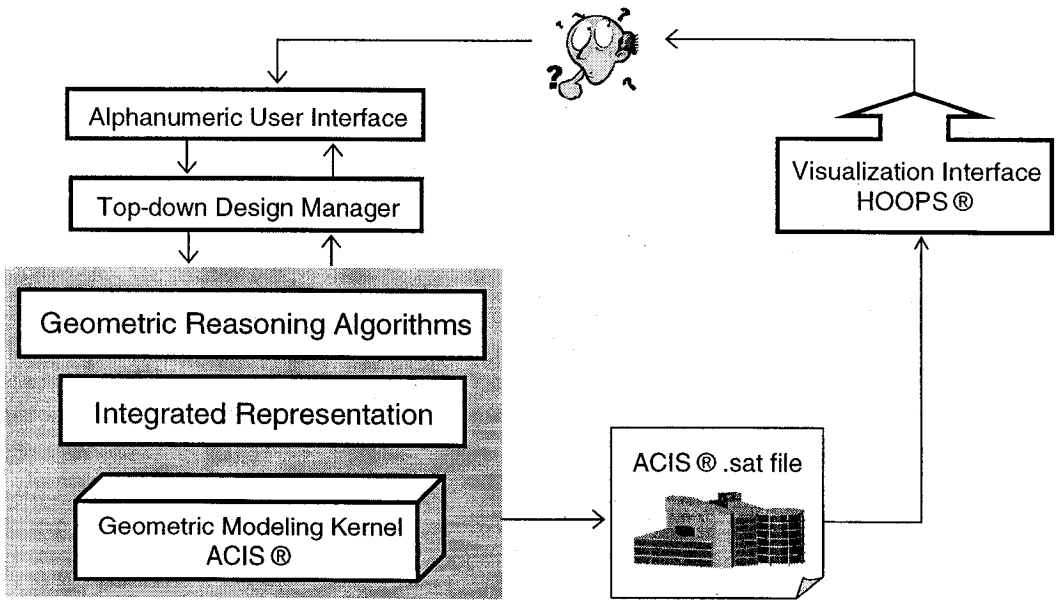


Figure 6.1 Prototype system architecture

Figure 6.1 shows that once a model of the building is created by the user, only its low level geometry (planes, lines, etc.) and topology (faces, edges, etc.) are saved into an

ACIS (Spatial Corp. 2004) text file (which is called a SAT file) that is read by HOOPS (TSA Inc. 2004) for visualization.

6.2 Overall object-oriented class interactions

Figure 6.2 shows a class diagram indicating the main object-oriented classes (i.e. boundary, control and entity classes) that make up the software prototype. Boundary classes make up the interface with the user (e.g. user menus), control classes drive the entire program and entity classes are the ones that are manipulated and have some meaning to the user. On the one hand, the entity classes implemented by the prototype have already been described in Chapter 5. These classes make up the ***Integrated Representation***, as well as the ***Synthesis Algorithms***. Some of them are shown at the bottom of Figure 6.2 to illustrate how they integrate with the other components of the prototype. On the other hand, the managers (Main Manager and Assistant Manager) shown in Figure 6.2 are the control classes that make up the ***Top-Down Design Manager***. The ***Alphanumeric User Interface*** is the only boundary class.

As shown in Figure 6.2, when the user executes the prototype the **`Main Manager`** class is called to drive the design process. The **`Main Manager`** calls the **`Alphanumeric User Interface`** to display the Main Menu. On the user's request from the menu, the **`Main Manager`** may continue driving the program or transfer the control to the **`Assistant Manager`**. The decision depends on the user's preferred way of entering data. If the user prefers using an *Input File* (presented hereafter using *italic* typeface), then the **`Main Manager`** continues driving the program by reading and processing the *Input File*, which has been created in advance. The processing consists in transforming the data from the

Input File into entities of the design model using the ***Integrated Representation*** as a template. If the user decides to enter data directly via C++ code, then the **`Main Manager`** transfers the control to the **`Assistant Manager`**. In this case, the user creates in advance a `drive` function with the input data, i.e. the user needs access to the C++ code of the `drive` function and then compiles before running the program. The drive function is then run by the **`Assistant Manager`** to generate the entities of the model. Therefore, either the *Input File* called by the **`Main Manager`** or the `drive` function from the **`Assistant Manager`** are the current mechanisms for entering data into the program.

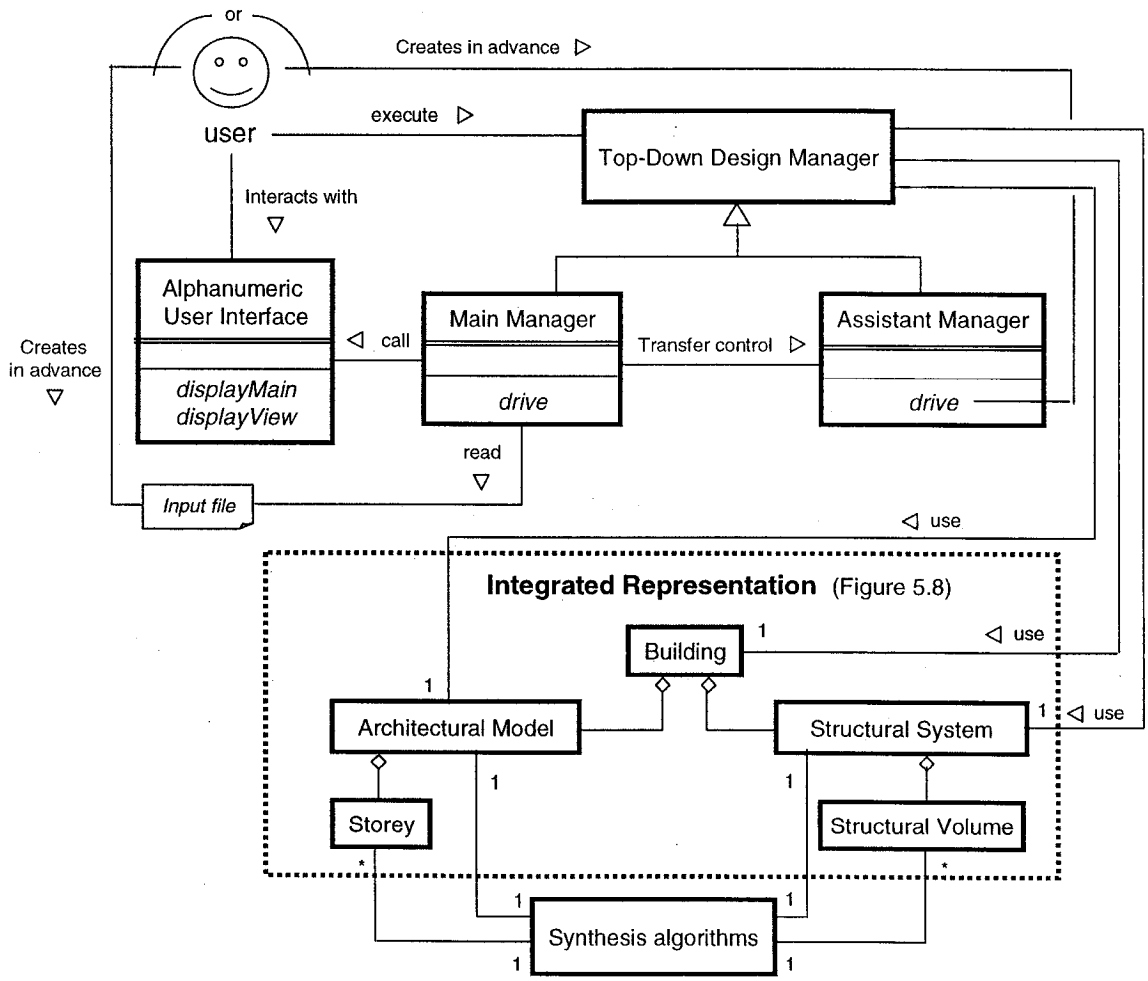


Figure 6.2 Class diagram showing component and user interactions

Note from Figure 6.2 that the manager classes, as well as the ***Synthesis Algorithms*** are related only to the main organizing entities from the ***Integrated Representation***. These links give control classes and ***Synthesis Algorithms*** access to all entities of the model being designed.

6.3 *Alphanumeric User Interface*

This is a single object-oriented boundary class that simplifies the user interaction with the prototype. It has two menus, the MAIN MENU (Figure 6.3) and the VISUALIZATION MENU (Figure 6.5). These menus are not meant for manipulating input data since entering and modifying graphic data using alphanumeric menus is error prone and inefficient. Therefore, as discussed in section 6.2 before running the program all the input data must be created and organized either in the *Input File* or directly in the `drive` function of the **`Assistant Manager`**. Then, the prototype assists the user in entering additional data, making verifications and producing output results.

The input data includes the **`Architectural Model`** and the abstract (i.e. non-physical) **`Structural System`** (described in section 2.1), which includes **`Independent Structural Volumes`**, **`Structural Zones`**, **`Structural Subsystems`** and **`Structural Assemblies`** with overall information (e.g. type(s) and abstract geometry). The input data is then processed by the program that generates the physical **`Structural System`**. Therefore, considering the steps of the simplified top-down synthesis process described in section 5.3.2.1, input data is entered through the initial steps that are carried out by the architect and by the engineer, while the final steps are calculated by the prototype. For simplicity, the abstract **`Structural System`** is called hereafter "partial **`Structural`**

System" because only a partial description of the structure is manipulated initially by the user and this is later completed with the assistance of ***Synthesis Algorithms.***

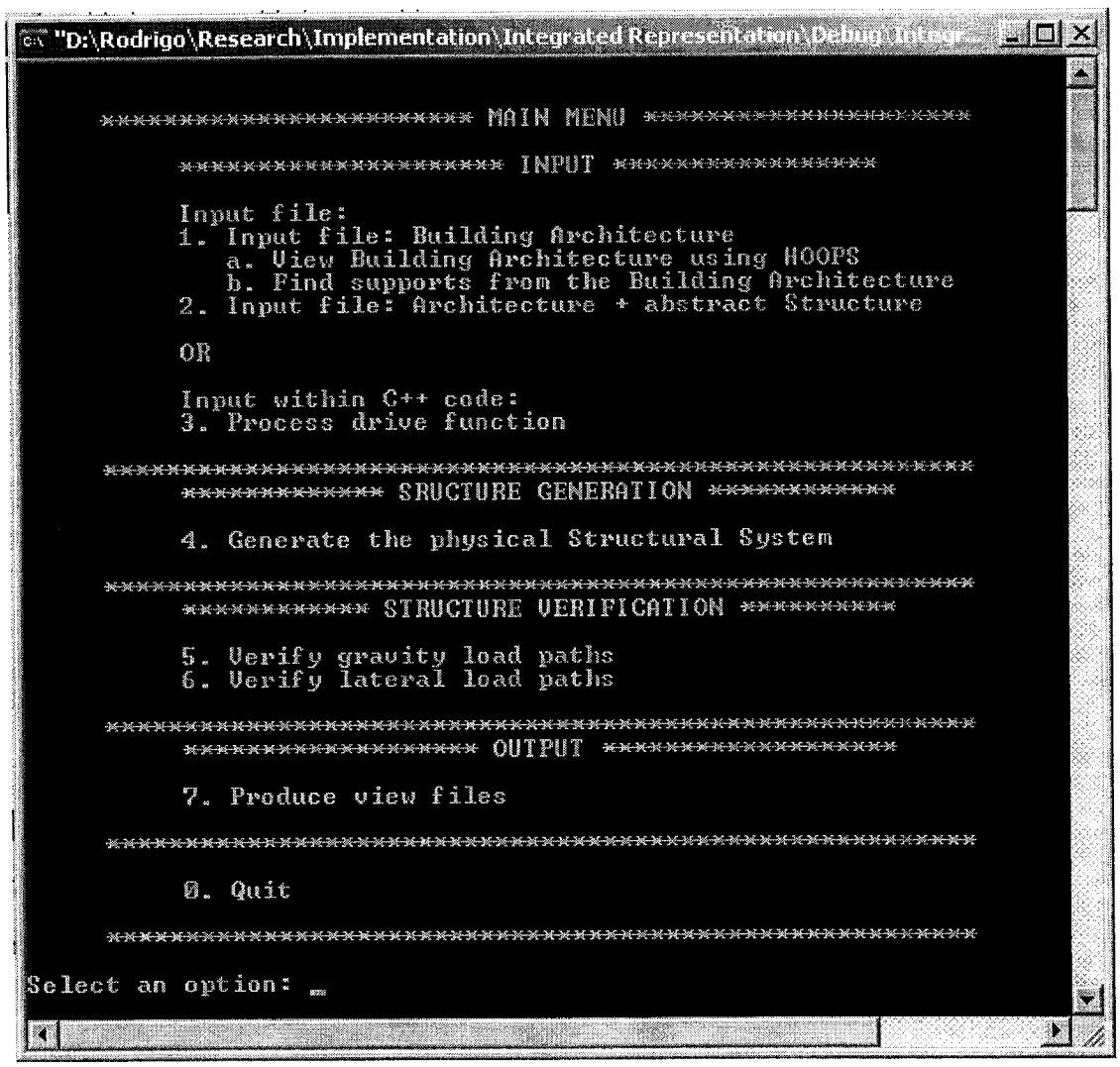
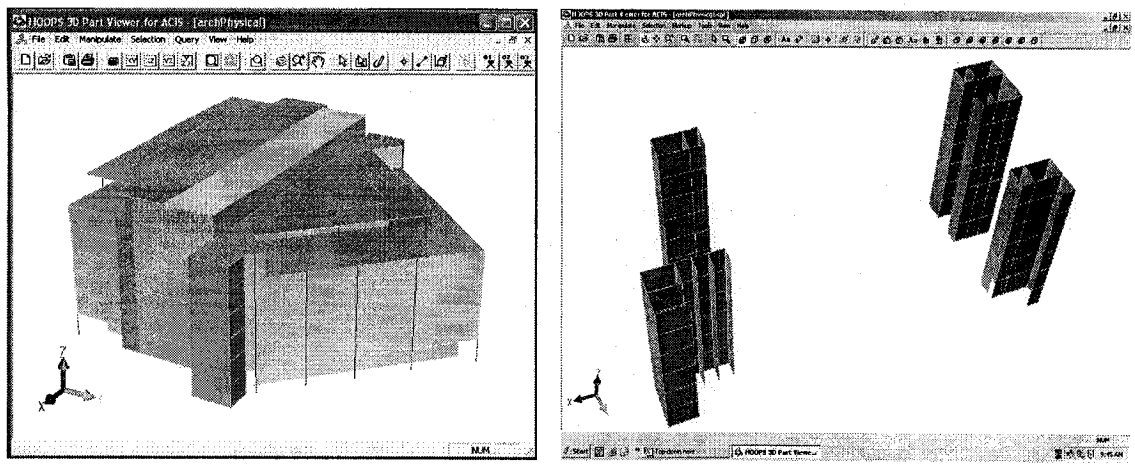


Figure 6.3 Software prototype: MAIN MENU

Figure 6.3 shows that two main alternatives are provided for entering data in the prototype: the first one is through an *Input File* (options "1" and "2" from the menu) and the second one is directly using C++ code (option "3" from the menu). With option "1", the entities from the building architecture are organized by the user in advance within an *Input File*. With option "a" the **Architectural Model** is generated from an *Input File*

(which is described in section 6.4) and an ACIS® SAT file is produced with its geometry. The SAT file is then visualized using HOOPS® (Figure 6.4 a). With option "b" ***Inspection Algorithm*** *findSupportsFromArchitecture* is executed and a list with the names and "x,y" coordinates of **`Walls`** that can become structural is obtained, as well as a SAT file with the geometry of these **`Walls`** (Figure 6.4 b).



(a) An **`Architectural Model`** (b) Continuous core **`Walls`**

Figure 6.4 Inspection of the **`Architectural Model`** using the prototype

After the inspection of the **`Architectural Model`**, the user adds the abstract (i.e. non-physical) entities from the **`Structural System`** at the end of the *Input File* that already incorporates the **`Architectural Model`** and selects option "2" for entering the expanded *Input File* into the prototype. As another alternative for entering data with option "3" all the entities from the **`Architectural Model`** and the abstract entities from the **`Structural System`** are entered directly using C++ code (as described in section 6.5).

Up until this point only a partial model of the **`Structural System`** has been obtained. Option number "4" in the MAIN MENU generates the physical **`Structural System`** using ***Configuration Algorithms*** (section 5.3.2), given the input information provided

either via options "1" and "2", or option "3". Once the physical structure has been generated it can then be verified using options 5 and 6. The verifications are done using the ***Verification Algorithms*** described in section 5.3.3. At any moment during structural generation or verification, if structural requirements are not satisfied (i.e. member supports, continuous load paths and span thresholds), the computer warns the engineer, for example: "Beam located at position (x1, y1, z1), position (x2, y2, z2) is not properly supported". Option 7 opens a VISUALIZATION MENU that provides several alternatives for the visualization of the design model (see Figure 6.5).

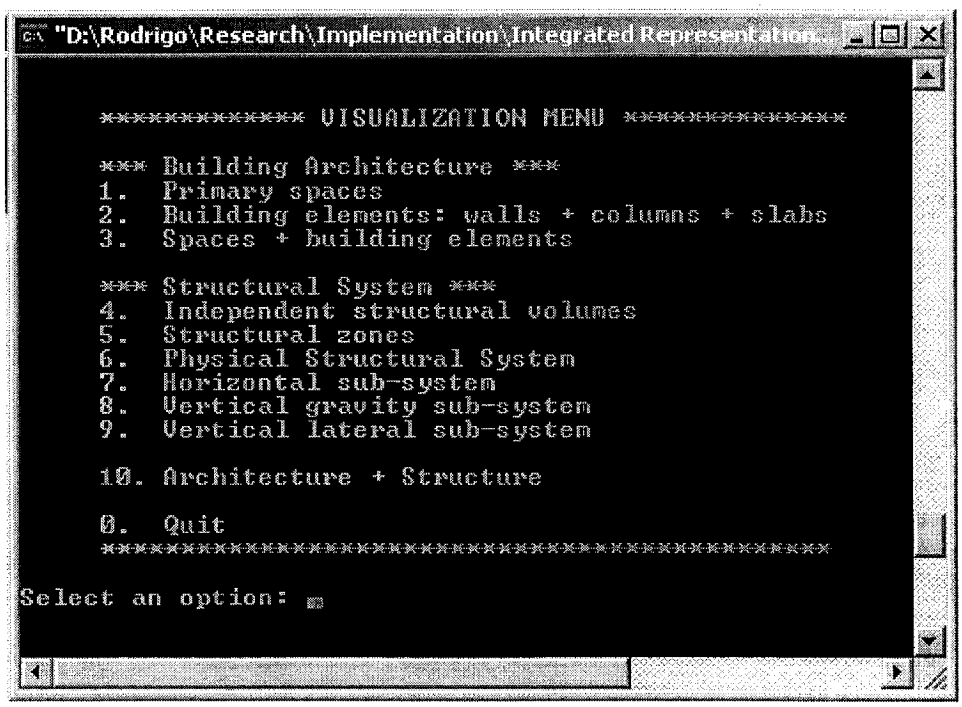


Figure 6.5 Software prototype: VISUALIZATION MENU

Each of the visualization alternatives creates an ACIS® text file (i.e. a SAT file), which is then read by HOOPS® for visualization. In the VISUALIZATION MENU, the first three options allow the visualization of the **`Architectural Model`**. For example, the first option generates a SAT file with the geometry of all **`Primary Spaces`**. The second option

saves a SAT file with the geometry of **Building Elements** only (i.e. **Columns**, **Walls** and **Slabs**). The third option creates a SAT file with the geometry of both **Spaces** and **Building Elements**. Options 4 to 9 allow the visualization of the **Structural System**. Finally option 10 is a combination of options 2 and 6.

6.4 The *Input File*

When the user selects options "1" and "2" from the MAIN MENU, the prototype reads the *Input File* sequentially and builds either the **Architectural Model** only (option "1") or both the **Architectural Model** and the partial **Structural System** (option "2"). The *Input File* is a text file that consists basically of entity types, names and Cartesian coordinates. It is inspired from ACIS text files which are used for storing low-level geometry and topology. Figure 6.6 shows an example of the *Input File* that incorporates both the **Architectural Model** and the partial **Structural System**. Note that all dimensions and coordinates are given in millimeters. For example, in Figure 6.6 the **Site** is entered in the first line by means of "x,y" coordinates of vertices. In the second line the **Building** is given a name and the number of **Stories** is entered. The third line defines the project **Grids** with two directions (x: 1, 0 and y: 0, 1) and a repetitive module in each direction (10000 in x direction and 10000 in y direction). Then the next line indicates that the description of the **Architectural Model** follows in the lines below with each **ASlab** (i.e. the architectural view of a **Slab**) defined first with a name and an elevation above ground. Then, each **Storey** is also defined through its name, and its floor and ceiling **ASlabs**.

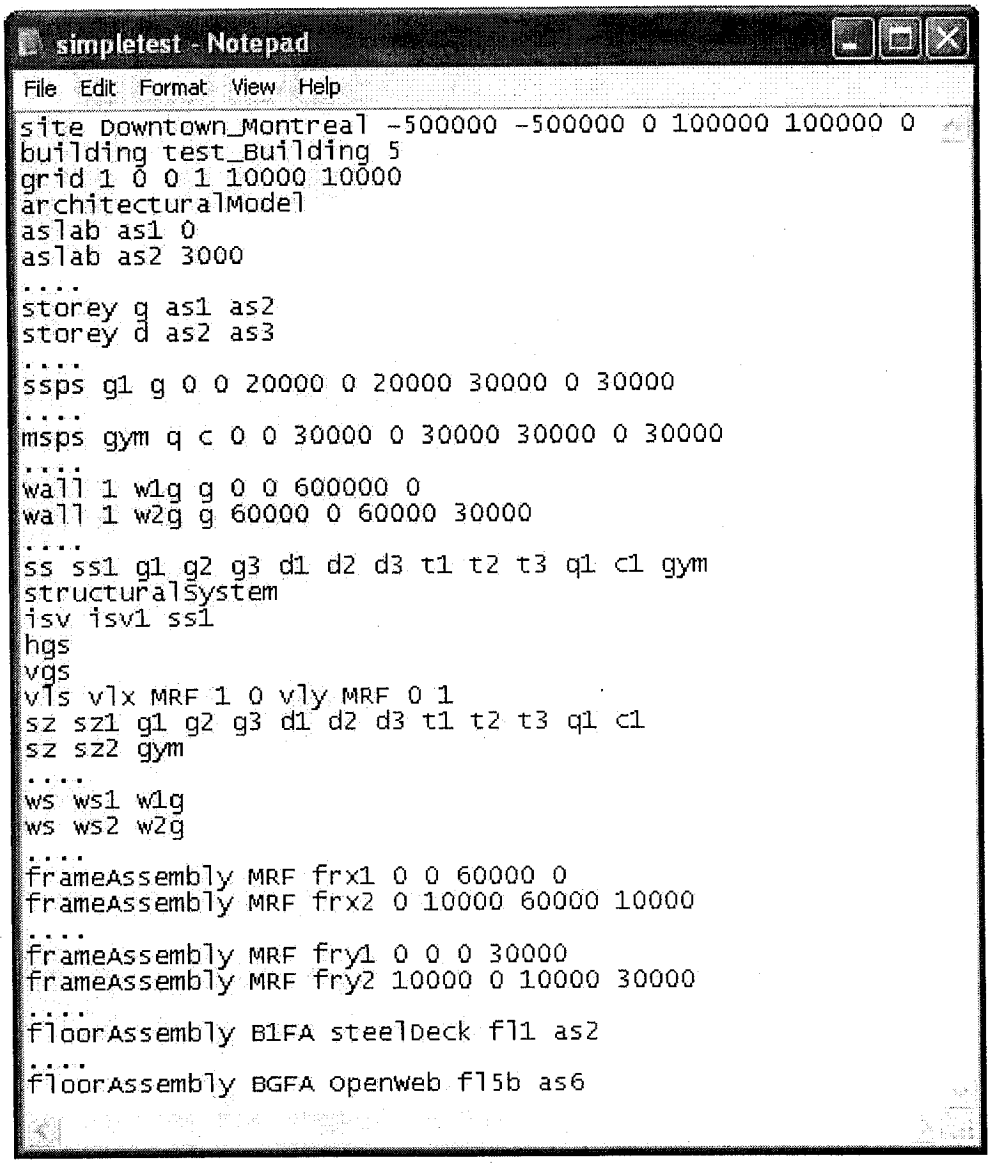


Figure 6.6 *Input File* for entering data to the software prototype

Each **Single Storey Primary Space** (SSPS in the file) is then given a name, a **Storey** and the "x,y" coordinates of its vertices. Note that all coordinates entered are "x,y" Cartesian coordinates. The "z" coordinate is always computed from the elevation and the height of **Stories**. **Multi-Storey Primary Spaces** are given a name, the number of **Stories** they span and the "x,y" coordinates of the vertices that define their geometry in

floor plan. **Secondary Spaces** are given a name and the names of **Spaces** that they aggregate. **Walls** are described by a type (type 1 is interior **Wall**, type 2 is curtain **Wall**, etc.), a name, a **Storey** where they belong, and the "x,y" coordinates of their base in floor plan. Note that four dots in a line "...." are not part of the input data; they are shown in Figure 6.6 to indicate that more entities similar to the ones described above the line follow. For option "1" in the MAIN MENU the *Input File* ends with the description of the **Secondary Spaces** from the **Architectural Model**.

For option "2" in the MAIN MENU the abstract entities from the **Structural System** are incorporated to the *Input File*. As illustrated in Figure 6.6, the next line after initialization of the **Structural System** defines an **Independent Structural Volume** (ISV) with its name and the list of **Spaces** it aggregates. In this case it aggregates only one **Secondary Space** (i.e. ss1). Then all entities that belong to the **Independent Structural Volume** follow after its definition. The **Horizontal Gravity Subsystem** (HGS) and the **Vertical Gravity Subsystem** (VGS) are defined next. Then, the **Vertical Lateral Subsystems** (VLS) are described by providing one or various types (e.g. **Moment Resistant Frame** – MRF with shear **Walls**) and a direction. **Structural Zones** (SZ) are then given their name and the names of the **Spaces** they aggregate. **Wall Stacks** (ws in the file) are also given a name and the name of one of the **Walls** that they aggregate (as obtained with option "a" from the MAIN MENU). **Frame Assemblies** are defined next by means of a type, a name and the "x,y" coordinates of two points that define their "x,y" layout. As explained in Chapter 5 (sections 5.2.2.5 and 5.3.2.1), from an "x,y" alignment, the abstract geometry of a **Frame Assembly** (i.e. a vertical plane) is generated by the computer, which spans the entire **Independent Structural Volume**

where it belongs. Finally, **Floor Assemblies** are defined with a generic type (e.g. **Beam** on **Girder**: BGFA), a specific type (e.g. steel deck), a name and an associated **ASlab**. When a **Floor Assembly** is given a specific type, it gets particular constraints including for example span thresholds, which are later verified. **Structural Layout Constraints** can also be defined in the *Input File* and associated to any particular **Space**. In addition, the user can associate a **Structural Zone** to a **Floor Assembly** whenever he/she realizes that the **Structural Zone** includes constraints that affect the layout of the **Structural System**. This is done in the *Input File* by adding the name of the **Structural Zone** at the end of the **Floor Assembly** line. Finally, the complete description of structural entities must be repeated for each **Independent Structural Volume** in the design project.

6.5 The `drive` function

The `drive` function is an alternative method for entering input data directly using C++ code. Appendix D shows the complete C++ listing of a `drive` function for a sample building that is well documented with comments. Using the drive function directly the user can input the architectural model first, visualize it using HOOPS (TSA Inc. 2003) and inspect it using the algorithms *verifyWallContinuity* or *findSupportsFromArchitecture*. Then he/she can add the abstract structural entities and generate the physical **Structural System**. This section adds a few additional comments about the `drive` function.

There are a number of differences between entering data through the *Input File* and using the `drive` function. One of these differences is that with the `drive` function the user can

copy typical **Stories** using the function *copyStorey*, which is an advantage over the *Input File* where each **Storey** has to be entered. However in the example of Appendix D no typical **Storey** exists so the *copyStorey* function is not used. In addition, with the *drive* function the user manipulates methods from the entity classes directly including ***Inspection Algorithms***. For example referring to the C++ code in Appendix D, ***Inspection Algorithm*** *verifyWallContinuity* is executed in line 457 given the name of a **Wall** and the **Storey** where it belongs. Then, a corresponding **Wall Stack** (WSA) is defined in line 461. Since the drive function is a more flexible way of entering data to the prototype it was used for the case studies described in Chapter 7.

6.6 User-model interaction

In the prototype the interaction between the user and the model is supported through the MAIN MENU and the *Input File* or the *drive* function. This is not an ideal situation and diminishes the capabilities of the software prototype. In this respect, the need for a Graphical User-Interface (GUI) is apparent because the prototype provides the underlying capabilities for user-model interaction through the ***Synthesis Algorithms***. Nevertheless, the current prototype implementation demonstrates the applicability of representative ***Inspection Algorithms*** (option "1.b from the MAIN MENU) and ***Verification Algorithms*** (options 5 and 6 from the MAIN MENU). User-model interaction is also achieved through warning messages that are sent to the engineer during the generation of the physical **Structural System**, for example when supports for a given **Structural Element** are not found.

Considering changes in the design, the prototype allows changes to be made to both the **Architectural Model** and the **Structural System**. These changes have to be done either in the *Input File* or in the `drive` function. For example, if the user wants to move a **Wall**, he/she simply changes its base (i.e. "x,y" coordinates) in the *Input File* or in the `drive` function. As another example, if the engineer wants to change the layout of a **Frame Assembly**, he/she changes the coordinates of the two points that determine its horizontal projection. The prototype is then run again so that the **Structural System** is recomputed.

6.7 Implementation of *Synthesis Algorithms*

Over 20 ***Synthesis Algorithms*** have been implemented (see listing in Appendix C). All such algorithms make use of low-level functions from the ACIS geometric modeler. The most representative algorithms have been described in Chapter 5 (cf. section 5.3). The following examples illustrate the functionalities provided by other ***Synthesis Algorithms***:

- Sort any group of architectural or structural entities in any given direction (e.g. **Spaces**, **Stories** and **Structural Connections**).
- Construct an adjacency graph from a group of entities (e.g. **Spaces** or **Beams** within a **Floor Assembly**).
- Test adjacencies among entities (e.g. **Spaces**, **Structural Elements**).
- Test interferences among entities.
- Test alignment and coplanarity among entities.
- Test containment (e.g. a **Column** within a **Space**).

- Intersect any two entities (e.g. two **`Frame Assemblies`**) in order to generate another entity (e.g. a **`Column-Stack`**).

6.8 ACIS *Geometric Modeling Kernel*

ACIS (Spatial Corp. 2004) is object-oriented and is written in C++. It consists of a set of C++ classes and functions. A developer uses these classes and functions to create an end user application. For a C++ application there are two distinct mechanisms for interfacing with ACIS:

- The direct interface, using the public member functions of ACIS classes directly.
- The API (Application Programing Interface), that consists of a library of functions, which are specifically designed for interfacing with ACIS classes. The advantage of API functions is that they are guaranteed to remain consistent from release to release, regardless of modifications to low-level ACIS data structures and functions.

The software prototype uses the API functions to interact with ACIS®. For illustrative purposes, some examples of ACIS® API functions are presented as follows:

- api_boolean(first Body, second Body, INTERSECTION)
- api_point_in_body(test Point, test Body, result from test)
- api_get_edges(Body, list of edges)
- api_find_face(Body, unit_vector(0,0,-1), Face)

ACIS® is a boundary- representation modeler (B-rep), which means that its data structure defines the boundary between solid material and empty space. ACIS® separately represents the geometry (detailed shape) and the topology (connectivity) of objects. The

root object-oriented class in ACIS® is the ENTITY. All other classes are derived from the ENTITY class. Figure 6.7 shows the topology of objects in ACIS. Bodies are the highest level entities in ACIS Models. Typically, a body is a single solid or a sheet component. A body can also be several disjointed bodies treated as one. Bodies "own" zero or more lumps. A lump represents a bounded, connected region in space. A shell is an entire connected set of faces and/or wires. A face is a portion of a single geometric surface in space (the two dimensional analogue of the body). Zero or more loops of edges constitute the boundary of a face. A wire is a connected collection of edges that are not attached to faces and do not enclose any volume. A coedge records the occurrence of an edge in a loop of a face. The introduction of coedges permits edges to occur in one, two or more faces. An edge is the topology associated with a curve. Finally, a vertex bounds an edge. It is generally the corner of a face or a wire.

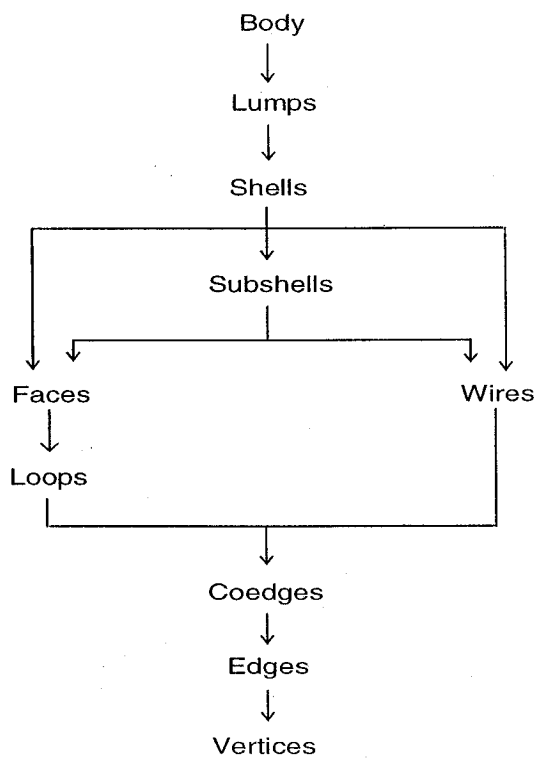


Figure 6.7. ACIS topologic entities (From Spatial Corp. 2004)

As shown in Figure 6.8, the geometry of ACIS objects consists basically of a hierarchical collection of points, curves and surfaces. Thus for example, the geometry of a `Column` is an ACIS® straight curve, while its topology is a wire body. And, the geometry of a `Wall` is an ACIS® plane, while its topology is a sheet body that has one lump, one shell, two faces, two loops, four edges, eight coedges and four vertices.

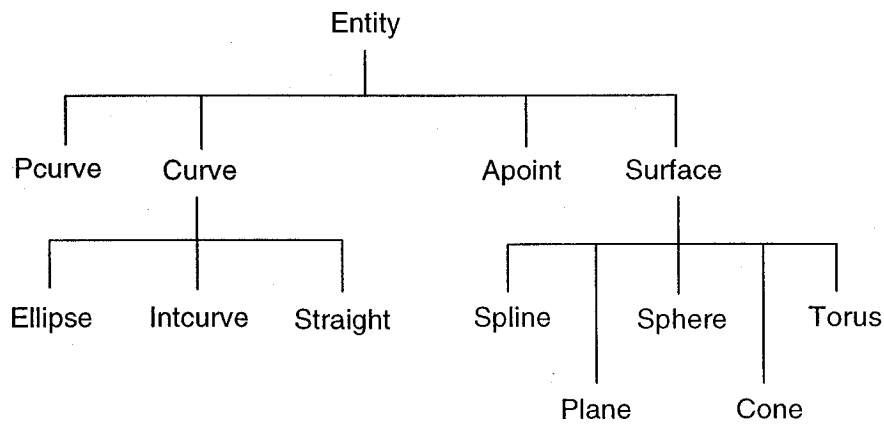


Figure 6.8. ACIS geometric entities (from Spatial Corp. 2004)

ACIS uses a special format to save model information into text files, which are called SAT files (SAT stands for Standard ACIS Text). While SAT files store information regarding the geometry and topology of the model, they do not include any high-level information associated with geometric entities. Thus, when any of the options is selected from the VISUALIZATION MENU in Figure 6.5, the software prototype selects the entities to be visualized, extracts their geometry and accumulates the geometry of each model entity into a list. This list is then used by an API function for creating the SAT file. Figure 6.9 shows the partial view of a SAT file which is presented for illustrative purposes only. In the file, the first line records the version number of ACIS used to generate the file (in this case 6.1). The second line records the number of bodies saved in the file as well as who saved the file and when it was saved. The third line deals with the

numerical precision of the modeler. All subsequent lines describe either a topologic or a geometric entity.

SAT files can be processed (i.e. read and write) by a variety of programs (e.g. AutoCAD by Autodesk and all other packages that incorporate ACIS as the underlying geometric modeler) and therefore they can be used for the exchange of primitive geometric data between different applications.

Figure 6.9. ACIS SAT file

6.9 HOOPS visualization interface

HOOPS (TSA Inc. 2004) is a complete framework for developing 3D applications that has the capability of integrating a variety of geometric modeling kernels, including ACIS. HOOPS allows developers to render geometric modelers interactive. For example, it provides the mechanisms for developing graphical user interfaces that allow users to interact graphically with ACIS entities. This is done through a C++ object-oriented programming environment that is used for developing HOOPS applications. For the current prototype implementation, however, only a ready-to-use visualization component of HOOPS has been employed. Thus, each visualization option in Figure 6.5 produces a SAT file that is read by HOOPS, which recognizes and manipulates geometric and topologic entities only (e.g. plane, body, etc). This means that entities in HOOPS have no actual architectural or structural meaning (e.g. **`Floor Assembly`** and **`Space`**). Figure 6.10 shows the HOOPS visualization of the **`Architectural Model`** of a **`Building`**.

The visualization component of HOOPS® provides standard capabilities for manipulating the geometry of the model. For example, the user can rotate the model, perform zoom in and out, and pan. The user can also select and move individual entities from the model. Figure 6.11 shows a rotated and exploded view of the **`Architectural Model`** using HOOPS.

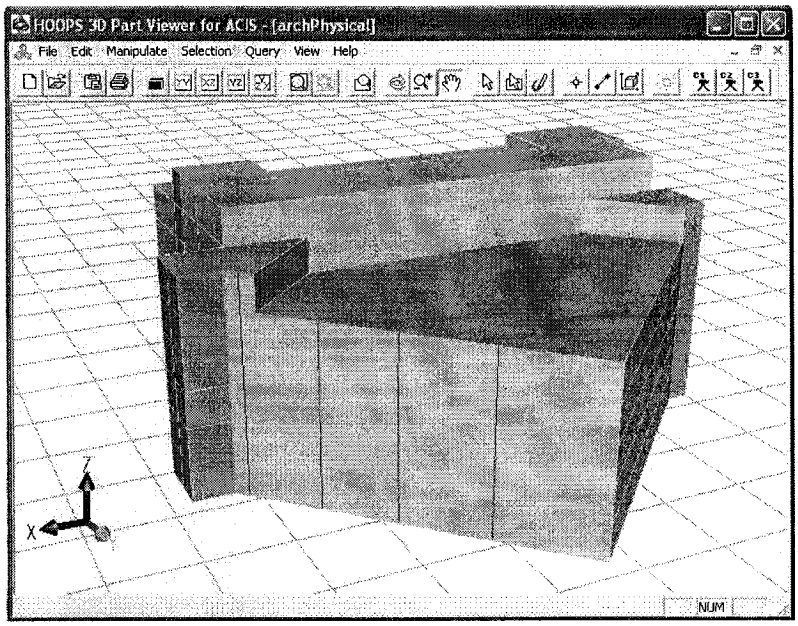


Figure 6.10. HOOPS visualization of an **`Architectural Model`**

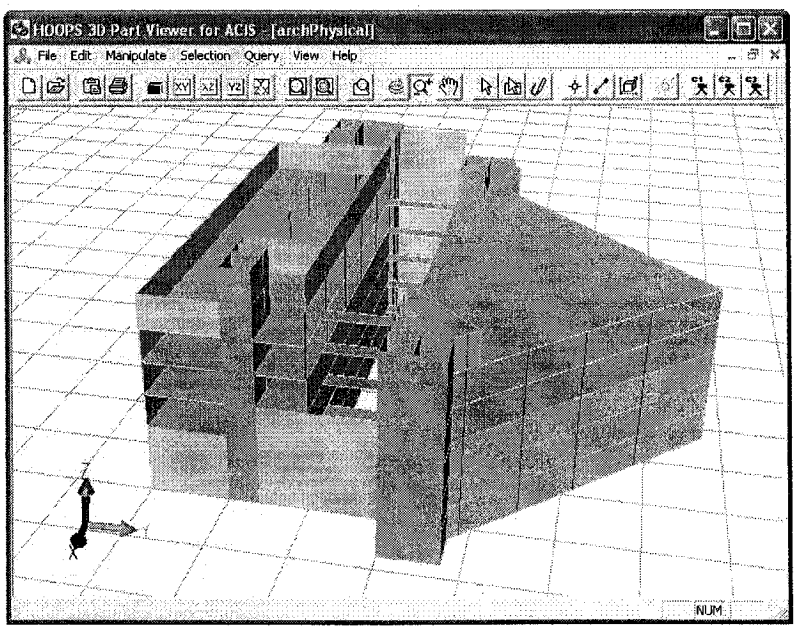


Figure 6.11. Rotated and exploded view of the above **`Architectural Model`** as produced with HOOPS

6.10 Additional considerations

The lack of a graphical user interface (GUI) comes with the obvious result of reduced user-model interactivity and design flexibility. As a consequence in the test cases discussed in Chapter 7 the amount of **`Walls`** and **`Primary Spaces`** included in the **`Architectural Model`** is minimized in order to simplify the tedious and error-prone input of alphanumeric data. The small amount of entities included in the **`Architectural Model`** is however sufficient for defining the shape of the building and its internal configuration. This reduction of entities in the model does not affect the outcome of the structural synthesis process since **`Walls`** and **`Spaces`** that do not play a structural role are simply ignored by the engineer and by the ***Synthesis Algorithms***.

The next obvious step in the development of the software prototype should be to integrate a graphical user interface. Providing the prototype with a user-friendly GUI will enable it to be tested with actual practitioners. Nevertheless as demonstrated in Chapter 7, the current prototype implementation is still sufficiently complete and robust to demonstrate the validity of the proposed conceptual design approach.

CHAPTER 7

TESTING AND VALIDATION

This Chapter presents two test cases that demonstrate improved assistance for collaboration between the architect and the engineer during conceptual structural design following the methodology for Architecture/Structure (A/S) integration proposed in Chapter 4 and using the required components of an environment for conceptual structural design described in Chapter 5. Such improved assistance is demonstrated through the proof-of-concept software prototype that has been presented in Chapter 6.

Even though the prototype was not tested with practitioners, due to the lack of a GUI, it considers the inputs and feedback from eight practicing architects and three structural engineers that were interviewed during the course of this research. In particular, the designers of the first test building provided details about the evolution of the early design of this building. They confirmed that this process followed a traditional sequential approach for collaboration between architects and engineers were early architectural decisions were made without full consideration of the structural implications.

In order to follow the test cases, while appreciating the benefits of the proposed approach over current design practice, it is recalled that current structural engineering packages enable the engineer to work at the physical structural level (i.e. involving elements and connections) give little regard to the building architecture. As explained by structural engineers, generating a 3D computer model of the structural system from architectural floor plans is a time-consuming and error-prone process. This process is often avoided

for simple buildings by breaking down the structure into 2D models that are analyzed semi-independently. For buildings with complex geometries however, the construction of 3D computer models of the structure cannot be avoided. The test cases demonstrate a more efficient an integrated approach for generating those models.

7.1 First test case: ETS building

The first test case is the new building for the *École de Technologie Supérieure* (ETS) in Montreal which opened in the fall of 2004. The building has five stories above ground (which are labeled ground, first, second, third and fourth) occupying an area of 20,400 m^2 and two parking stories below ground. The building houses several activities such as: classrooms, computer rooms, cafeteria, and various administrative services. Figure 7.1 shows an architectural rendering of the building while Figure 7.2 shows the second storey, which is typical. In Figure 7.2 numbered circles indicate staircases, elevator shafts and other shafts that at first glance can be considered good candidates for structural purposes.

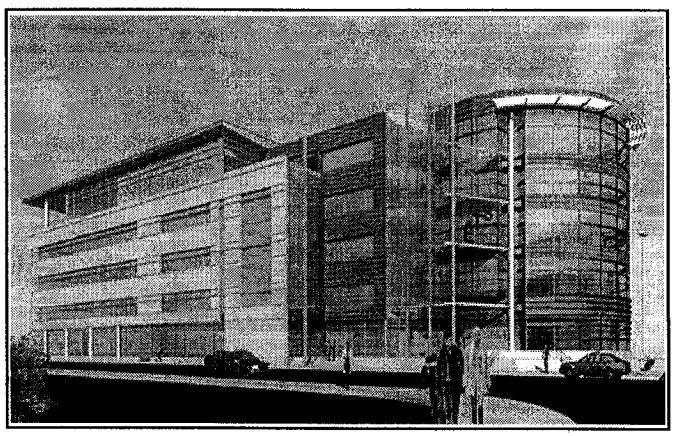


Figure 7.1. Architectural rendering of the ETS building

A gym and a multi-functional sports room are located on the third storey, and span two stories (i.e. the third and the fourth stories). They are large column-free rooms requiring particular structural considerations, which must integrate well with the rest of the structure. Figure 7.3 shows a floor plan of the third storey including the gym and the multi-functional sports room.

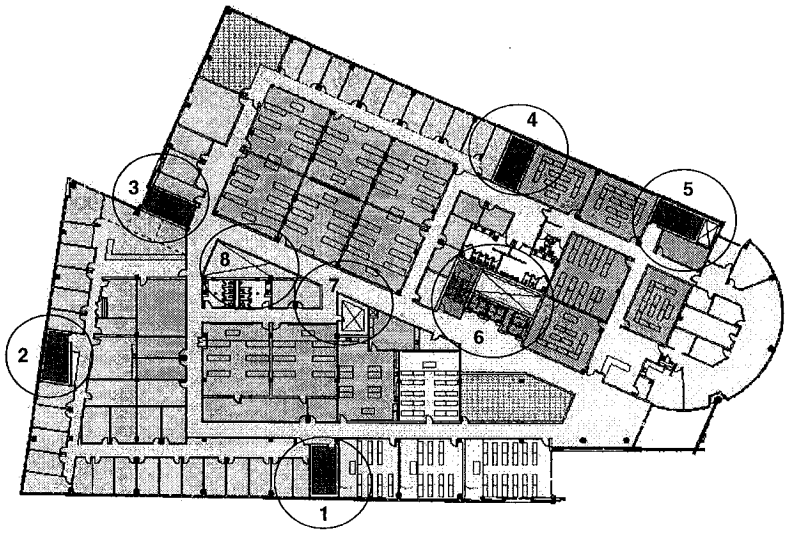


Figure 7.2 Second storey of the ETS building

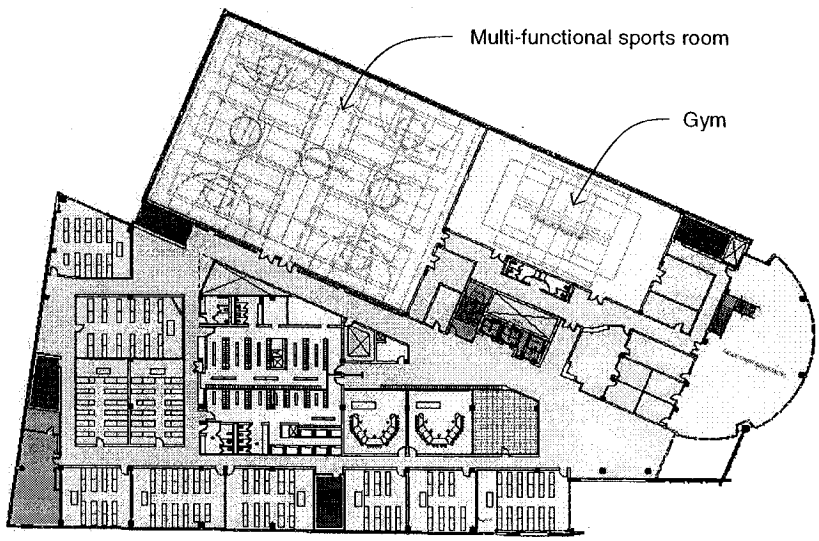


Figure 7.3 Third storey of the ETS building

The structural system is a reinforced concrete structure except for the roof and part of the fourth storey which are made out of steel. All the floor slabs are reinforced-concrete flat slabs with drop panels. The two parking levels below ground are surrounded by perimeter retaining walls. In the actual structure only the first six vertical circulation cores, indicated in Figure 7.2, have been used for structural purposes. Core walls are made out of reinforced concrete and are integrated to the structural system for gravity and lateral load support. The use of concrete for the structure was decided by the architect due to its visual appearance and "solidity".

All the core walls start from the ground at the second basement. However, following architectural functional requirements, some of these walls are interrupted before reaching the last storey. The roof of the building is a steel structure supported by beams and girders, except for the roof of the gym and the multi-functional facility which is supported by long-span trusses. Figure 7.4 shows the structural system of the actual building under construction.

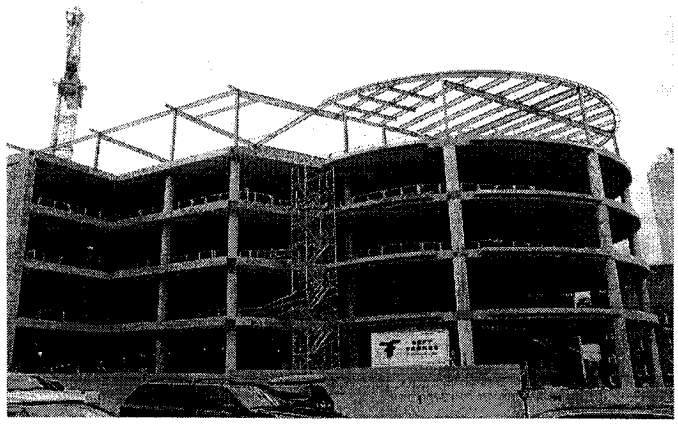


Figure 7.4 ETS building under construction

7.1.1 Pre-processing

As discussed in Chapter 6, before running the prototype, the architectural model is created first alphanumerically using C++ code in the *drive* function. Thus stories are entered with a name and an elevation, followed by **Single-Storey Primary Spaces** and **AWalls** that are associated to **Stories**. **Spaces** are entered with their name and the "x,y" coordinates of their perimeter, and **Walls** are entered with the "x,y" coordinates of their base and other information such as material (translucent/opaque) and function (enclose/partition and permanent/removable). **Multi-storey Primary Spaces** are also entered as well as **Secondary Spaces**. Whenever applicable, **Spaces** are assigned **Structural Layout Constraints**. All these data is entered using the appropriate methods in the *drive* function.

Compared to the actual building, the architectural model had to be slightly modified to fit the requirements of the software prototype. As discussed in section 4.3.2, the formal model of conceptual structural design supports functional integration, as well as physical integration. However, as mentioned in section 5.3.4 support for levels 3 and 4 of physical integration is still incomplete in the design assistance environment for conceptual structural design. As a consequence, it does no accept most architectural and structural elements that fall outside a common architecture-structure grid. This is the case of the ETS building. A floor plan of the second basement of the test building illustrates this point (see Figure 7.5).

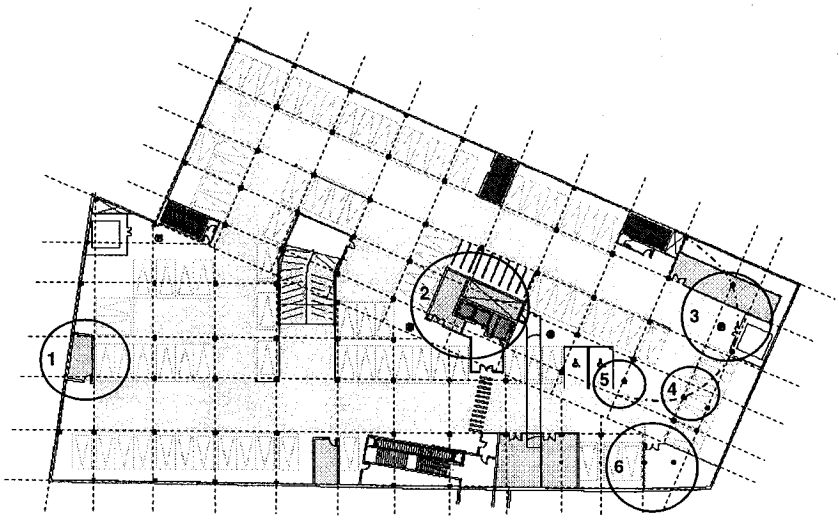


Figure 7.5 ETS building, floor plan of the second basement showing local misfits

Figure 7.5 shows six local "misfits", which are defined as structural entities that do not fit with the overall structural patterns (shown as gridlines). Misfits numbered 1 and 2 consist of core structural **`Walls`** that do not align with **`Frame Assemblies`**. Misfits 3, 4, 5 and 6 are **`Columns`** located outside of the intersection points between gridlines. For example in misfit number 5, a **`Column`** was moved because otherwise it would fall in the middle of the car circulation path. Misfits numbers 3 and 4 are caused by the need for support from the **`Stories`** above, which at these locations have a circular perimeter (see Figures 7.1, 7.2, 7.3 and 7.4). Since the design assistance environment does not support local structural misfits, the affected **`Walls`** and **`Columns`** had to be moved and sometimes rotated to fit within the overall structural pattern.

Another change that had to be made to the building architecture consists of adjusting the shape of the circular façade into straight lines because the software prototype is currently limited to straight line segments. The adjusted **`Architectural Model`** is illustrated in Figure 7.6. The model was constructed alphanumerically using the *`drive`* function of the software prototype.

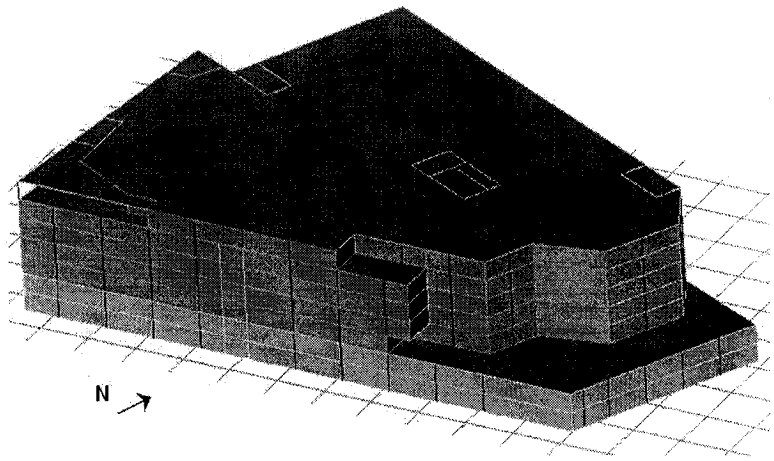


Figure 7.6 **`Architectural Model`** of the ETS as input to the prototype

The next section demonstrates how, given the **`Architectural Model`** shown in Figure 7.6, the engineer uses the software prototype to perform A/S integration.

7.1.2 Computer-assisted Architecture/Structure (A/S) integration

The methodology for A/S integration has been presented in section 4.4.1 and illustrated in Figure 4.8, which is repeated here for convenience. As explained in Chapter 4, the methodology consists of a sequence of activities that are classified either as inspection, configuration or verification activities. Thus, the inspection of the **`Architectural Model`** is carried out first, followed by the top-down configuration of the **`Structural System`**, and ending with the verification of the structural solutions. These activities are carried out with the software prototype as follows.

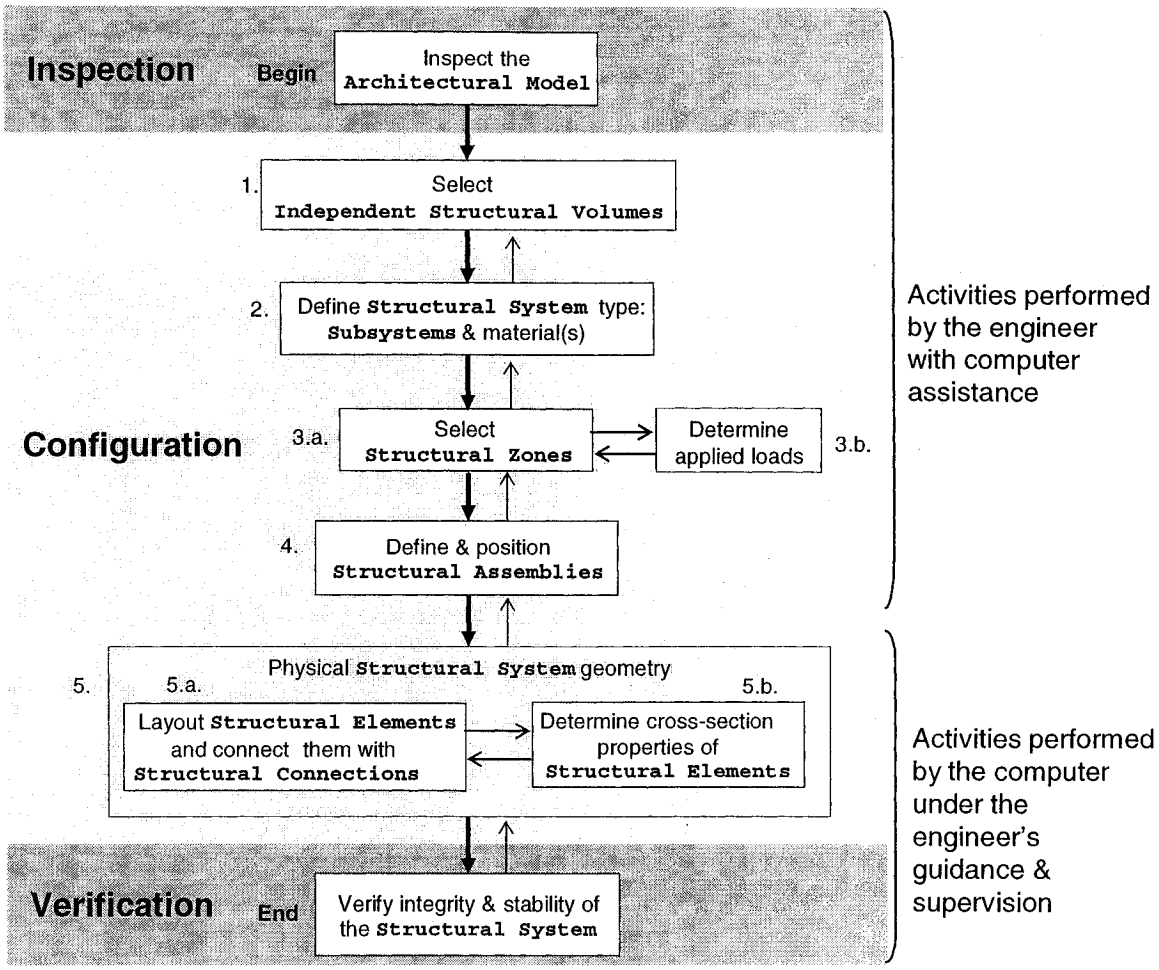


Figure 4.8 (repeated) Methodology for A/S integration during conceptual design

7.1.2.1 Inspection of the `Architectural Model`

The engineer initially inspects the **`Architectural Model`** in search for potential structural opportunities (i.e. patterns and vertical supports), difficulties (e.g. complex architectural configurations) and constraints (e.g. large column-free spaces). In Figure 7.2 eight potential cores and shafts have already been identified. In actual design practice the engineer would have to compare various floor plans to verify the vertical continuity of **`Walls`** and **`Columns`**. For example, the engineer would compare the Figures 7.2, 7.3 and 7.5 while maintaining a reference to the project grids. This task is tedious and error-prone

since some **Walls** that look continuous may in fact be slightly shifted or rotated. In Figure 7.2, this is the case of the shafts in circles numbered 7 and 8 whose **Walls** are slightly rotated on the ground **Storey** and disappear at the parking levels. Assuming for example that the shafts in circles numbered 7 and 8 could play a structural role, before making them structural the engineer would suggest to the architect to extend those **Walls** down to the second basement. If the architect agrees, he/she simply adds the corresponding **Walls** in both basements and then the synthesis process can resume.

Inspection Algorithms facilitate the task of finding continuous supports in the building architecture. With the software prototype, the engineer simply uses the algorithm *findSupportsFromArchitecture* to find out which elements (**Walls** and **Columns**) are continuous down to the second basement, or the algorithm *verifyWallContinuity* to verify only the **Walls** that are of particular interest. Then, the engineer selects the **Walls** and **Columns** that may tentatively become part of the structural system. The algorithm *findSupportsFromArchitecture* considers permanent (i.e. non-removable) and opaque **Walls** only. For example, perimeter **Walls** are not selected either because they are curtain **Walls**, which are glass-made and supported by an aluminium structure. Figures 7.7 and 7.8 show **Spaces** (indicated with different shades of grey) and **Walls** (indicated with lines inside and at the perimeter of **Spaces**) as entered in the prototype. The core **Walls** (within circles) have been identified as continuous down to the second basement by the algorithm *findSupportsFromArchitecture*.

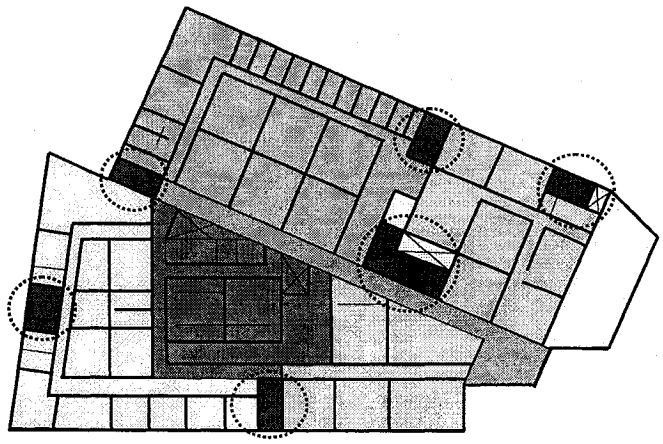


Figure 7.7 Second `Storey` of the ETS building as created using the software prototype

At this point, the engineer also executes the algorithm *verifyStructuralLayoutConstraints* to check for possible constraints that have been placed by the architect and may not be evident. For example, as shown in Figure 7.8, the multi-functional sports room and the gym are intended to be `Column-free`. The engineer also realizes that all the `Walls` at the perimeter of these rooms are opaque and permanent and therefore can house `Structural Elements`, such as `Columns`, as required for supporting the roof. Therefore, the roof of the multi-functional room is required to span across the entire width of the `Space` thus making it a good candidate for long-span light-weight steel trusses.

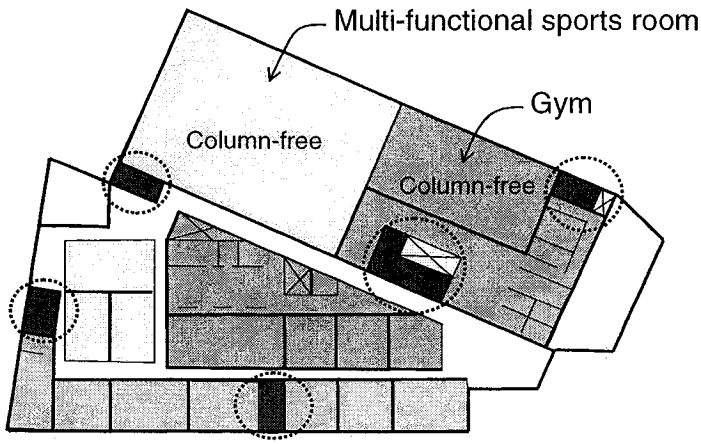


Figure 7.8. Third `Storey` of the ETS building as created using the software prototype

During the initial inspection of the **Architectural Model**, the engineer may also suggest global and/or local changes to improve the structural properties of the building architecture. For example, as discussed with the structural engineer of the project, he may need to verify in advance if the core **Walls** (see Figures 7.7, 7.8 and 7.12) are sufficient for providing lateral resistance and are well distributed to avoid unwanted torsional deformations (this will depend on the vulnerability of the building to be affected by lateral loads, i.e. the seismic region and wind vulnerability). In the event that the core **Walls** are not sufficient then either some concessions may have to be made by the architect to enable additional **Walls** to become structural, or a complimentary mechanism for providing lateral support, such as bracings or rigid frames, may have to be devised by the engineer. These decisions require discussion with the architect. The current prototype provides no support for making such decisions.

7.1.2.2 Configuration of the Structural System

a) Selection of Independent Structural Volumes

After inspecting the **Architectural Model**, the engineer decides that it can be supported by a single structural skeleton. This decision is based on the overall building dimensions, on the geometry of the building, and on the patterns from the building architecture. In this case the size of the building is reasonably small, its geometry is relatively homogeneous and its patterns are more-or-less uniform. Therefore, he/she creates one **Independent Structural Volume** by selecting all the **Spaces** in the **Architectural Model**.

b) Definition of `Structural System` type: `Structural Subsystems` and material(s)

Each **`Structural Subsystem`** is defined initially by specifying material(s) and types of assemblies, elements and connections that are most compatible and appropriate to behave as expected. The **`Vertical Gravity Subsystem`** consists of **`Column Stacks`**, the **`Vertical Lateral Subsystem`** consists of a combination of rigid (moment resistant) **`Frame Assemblies`** and shear **`Wall Stacks`** made out of concrete and the **`Horizontal Subsystem`** consists of "flat-slab-with-drop-panel" **`Floor Assemblies`**. This initial **`Structural Subsystem`** description is used for guiding the subsequent structural configurations since each **`Structural Subsystem`** will be populated later on with constituent **`Structural Assemblies`** and **`Structural Elements`**.

c) Selection of `Structural Zones`

The engineer selects and groups **`Spaces`** having similar structural characteristics into **`Structural Zones`**. **`Structural Layout Constraints`** associated to **`Spaces`** by the architect participate to the definition of **`Structural Zones`**. In Figure 7.9, the **`Independent Structural Volume`** representing the **`Structural System`** at this stage is divided into three **`Structural Zones`** (identified in Figure 7.9 with different gray shadings): a parking zone, a zone with classrooms and administrative services, and a zone that groups the gym and the multi-functional sports space. Then, the engineer assigns an applied load to each **`Structural Zone`** according to the functionality of the enclosed **`Spaces`**. The last **`Structural Zone`** (indicated with lighter shade of grey in Figure 7.9) groups **`Spaces`** that are column-free. Thus, it carries a **`Column-free Structural Layout`**

Constraint as specified by the architect to restrict the layout of the **Vertical Gravity Subsystem**.

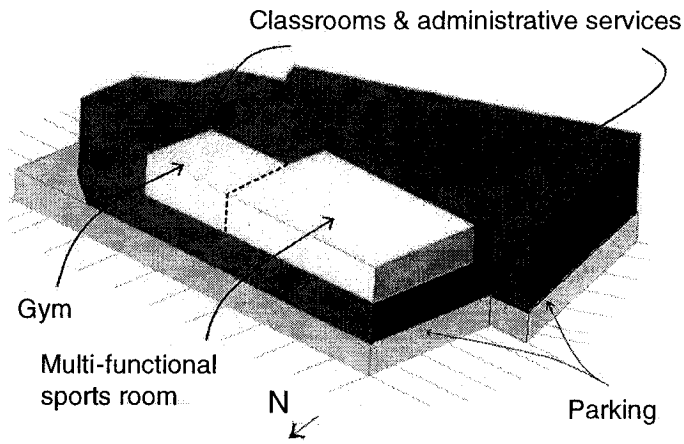


Figure 7.9 ETS building: **Structural Zones**

d) Definition and positioning of Structural Assemblies

Once **Structural Volumes** and **Structural Subsystems** are defined, the engineer proceeds with the configuration of **Structural Assemblies**. Initially, **Structural Assemblies** are spatially represented as abstract 2D planes that determine their layout and extent. Later on in the process, they are populated with **Structural Elements** and **Structural Connections**. Thus, the engineer begins by positioning rigid **Frame Assemblies** by following the project **Grids**. This is done simply by specifying two "x,y" points defining a frame alignment. The abstract geometry of each **Frame Assembly**, which is a vertical plane, is then generated by the computer within the boundaries of its parent **Independent Structural Volume**. By laying out **Frame Assemblies** in orthogonal directions the engineer implicitly defines column lines and structural bays. Once laid out, each **Frame Assembly** is then added to its corresponding **Vertical Lateral Subsystem**. Figure 7.10 shows the abstract geometry of all **Frame Assemblies**

as generated by the software prototype. These **Frame Assemblies** correspond to those of the actual building.

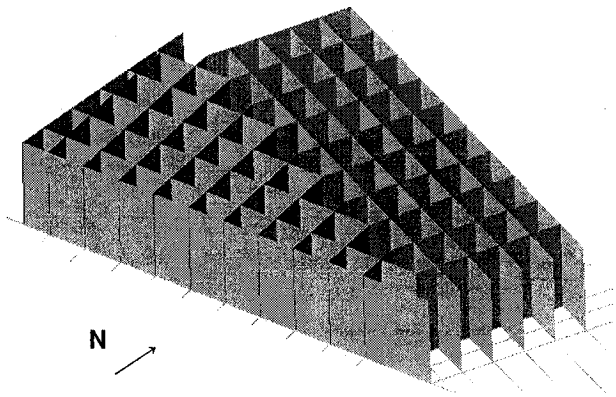


Figure 7.10. ETS building: abstract geometry of **Frame Assemblies**

As illustrated in Figure 7.11, the engineer selects **ASlabs** created by the architect and defines **Floor Assemblies**. Then, the prototype computes the abstract geometry of **Floor Assemblies** (a horizontal plane), which extends throughout the **Independent Structural Volume** except where **Structural Zones** require a special type of **Floor Assembly**. This is the case of the **Structural Zone** that groups the gym and the multi-functional sports room, which are **Column-free** thus requiring a special type of roof system. Each **Floor Assembly** is automatically added to the **Horizontal Subsystem**.

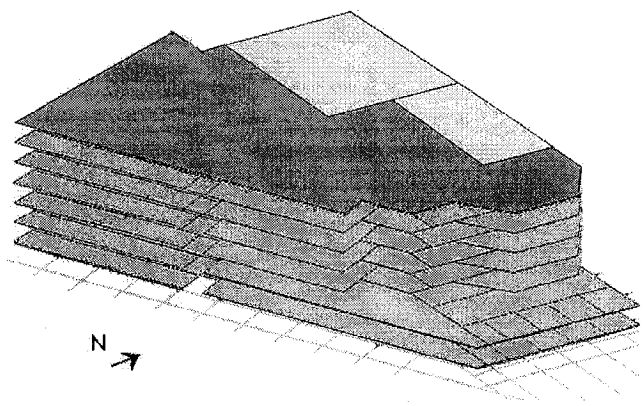


Figure 7.11 ETS building: abstract geometry of **Floor Assemblies**

At this point the engineer decides to add the **AWalls** that have been selected during the inspection of the **Architectural Model** (as indicated in Figures 7.7 and 7.8) to the **Structural System**. These **AWalls** become **SWalls** which are then grouped into **Wall Stacks** (see Figure 7.12). Next, the computer adds each **Wall Stack** to a coplanar **Frame Assembly**, if applicable, and to the corresponding **Vertical Lateral Subsystem** and **Vertical Gravity Subsystem**. Perimeter **AWalls** at the two basement levels also become part of the **Structural System** as retaining **SWalls**. However, these **AWalls** are partitioned by a ***Configuration Algorithm*** *divideArchitecturalWall* so that their counterpart **SWalls** fit within the structural grids.

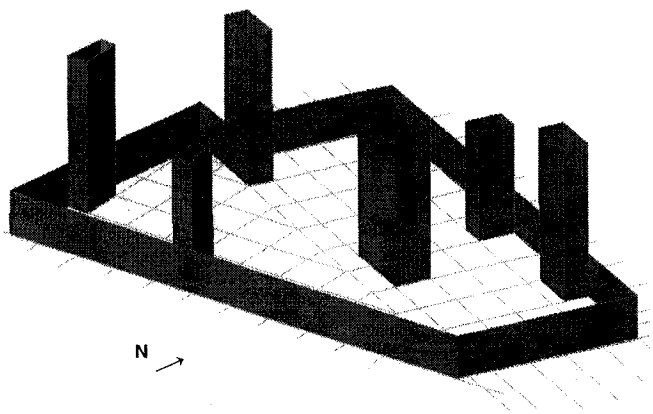


Figure 7.12 ETS building: engineer selects **AWalls** to become **SWalls**

AColumns, if located tentatively by the architect, also participate to the synthesis of the **Structural System**. These **AColumns** become structural provided that they are useful to fulfill structural load-transfer needs (i.e. providing continuous vertical supports and feasible floor spans, while being aligned on the same plane of the **Frame Assemblies** already positioned). If this is the case, **AColumns** become **SColumns** and they are stacked

in **Column Stacks**. In this example, as in the actual building, all **Columns** are laid out by the engineer.

e) Layout Structural Elements and connect them with Structural Connections

The algorithm *frameXframe* finds the intersection between the abstract geometry of **Frame Assemblies** and generates **Column Stacks** (see Figure 7.13). **Column Stacks** are interrupted at the top, in the gym and the multi-functional sports room because these are column-free.

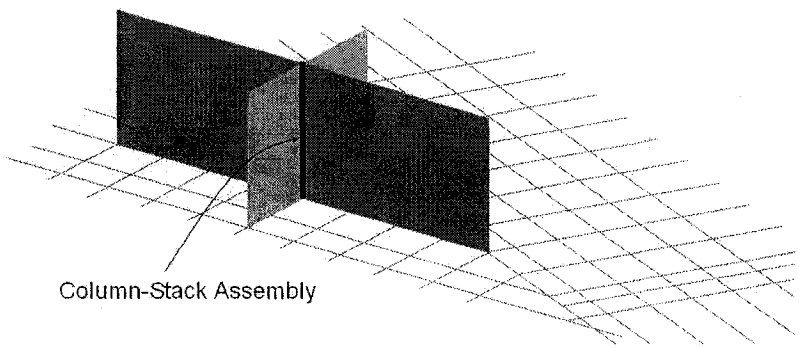


Figure 7.13 ETS building: intersection of two orthogonal **Frame Assemblies** generates **Column Stacks**

The algorithm *frameXfloor* finds the intersection between **Frame Assemblies** and **Floor Assemblies** and generates primary **Beams**. For this building, however, **Construction Lines** (see Figure 7.14) are generated instead of **Beams** because the **Floor Assemblies** are flat slabs. These **Construction Lines** are then used for the generation of **Slab Elements**.

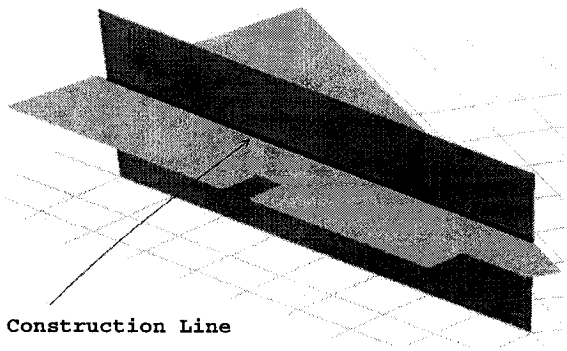


Figure 7.14 ETS building: intersection between **`Frame Assembly`** and **`Floor Assembly`** generates **`Beams`**

Finally, the algorithm *generateSlabElements* generates the **`Slab Elements`** for each **`Floor Assembly`**. **`Slab Elements`** are then connected to supporting **`Columns`** at the vertices. This completes the synthesis process of the **`Structural System`**. The resulting **`Structural System`** is described as a hierarchical organization of structural entities. At the lower hierarchical level, ***Configuration Algorithms*** have generated the physical structure composed of **`Structural Elements`** and **`Structural Connections`** (see Figure 7.15) as directed by the engineer.

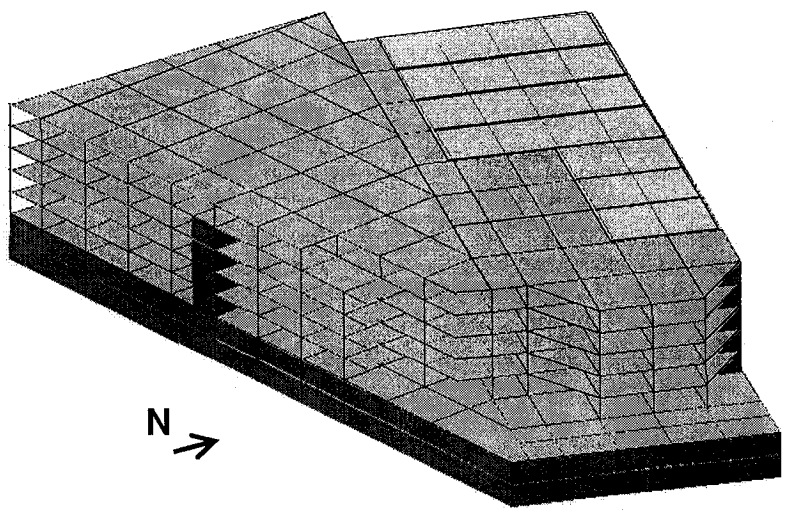


Figure 7.15 ETS building: the **`Structural System`** as generated using the software prototype

7.1.2.3 Verification of the integrity and stability of the `Structural System`

Once the configuration of the **`Structural System`** is completed, verification algorithms are executed to check for the integrity and stability of the structural system. Considering structural integrity, the algorithms *verifyGravityLoadPaths* and *verifyLateralLoadPaths* are executed to check for continuous gravity and lateral load paths to the ground. For this test case the algorithms did not detect any gravity or lateral structural discontinuities. Considering structural stability, the algorithm *verifyLateralLoadPaths* verifies that there are at least three frames connected to floor diaphragms that are not all parallel or intersecting at one single point, however the algorithm does not verify if the frames and shear **`Walls`** have sufficient capacity for providing lateral support while eliminating torsional instabilities. In addition, the stability of **`Columns`** is not verified, especially those spanning more than one **`Storey`**. However, in this case, the only **`Columns`** spanning two **`Stories`** are located at the perimeter of the gym and the multi-functional room (see Figures 7.16 and 7.17). Lateral support for these **`Columns`** is provided by the **`Walls`** that confine them.

Note that the change in structural material for the upper stories did not cause any inconvenient for the synthesis process. This is likely to be the case for most **`Structural Systems`** where light steel is in the upper **`Stories`** and heavy concrete in the lower stories. The reason for this is that lighter steel structures usually require less vertical supports than heavier concrete structures in the **`Stories`** below. Therefore, upper vertical support patterns are usually a subset of the lower support patterns and vertical continuity

is not affected. The result is illustrated in Figure 7.16. Figure 7.17 shows the same part of the structure as generated by the software prototype.

Figure 7.16 ETS building: actual `Structural System` under construction

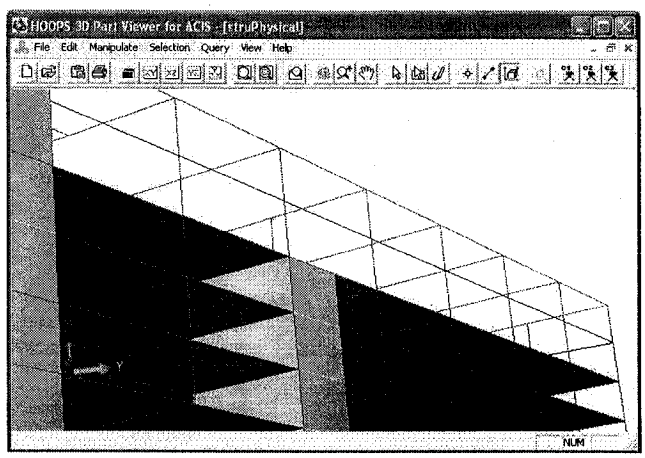


Figure 7.17 ETS building: `Structural System` as generated by the software prototype

7.2. Second test case: office building

The second test case is a hypothetical office building that is inspired from the Expo'98 Headquarters (Expo 98) in Lisbon, Portugal designed by EPR Architects Ltd. The building was chosen because of its geometric complexity. Due to the lack of information about the actual building, a modified version was developed. The modified building

maintains the overall features of the original one (e.g. size and shape). However, some architectural features are not included in the test case and the internal configuration of spaces has been modified. In addition, no structural information is available and therefore the structural system is not expected to be the same as the actual one. With the information available and assumed, the **`Architectural Model`** has been constructed alphanumerically using the `drive` function of the software prototype. Figure 7.18 shows two different views of the **`Architectural Model`** of the building as constructed using the software prototype. The hypothetical building houses several activities, namely: open office spaces, computer rooms, a cafeteria, and a multi-functional room (i.e. for meetings, conferences and social events), which is located on the top storey.

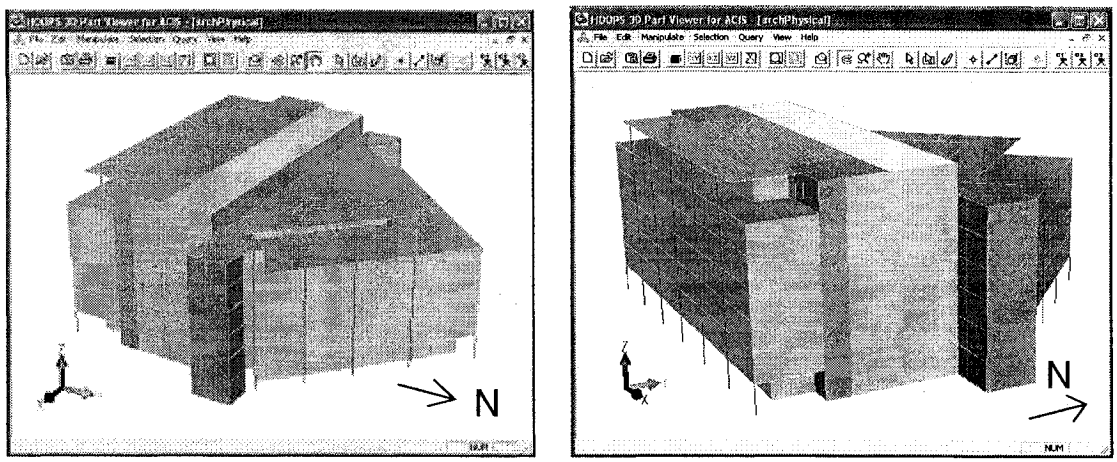


Figure 7.18 Expo98 test building: **`Architectural Model`**

The building actually consists of two semi-independent buildings, the south building and the north building, which are separated by a multi-storey storey atrium. The south building has seven stories and the north building has six. Pedestrian bridges cross the atrium at each level linking both buildings. Figure 7.19 shows an exploded view of the two buildings and the bridges that link them.

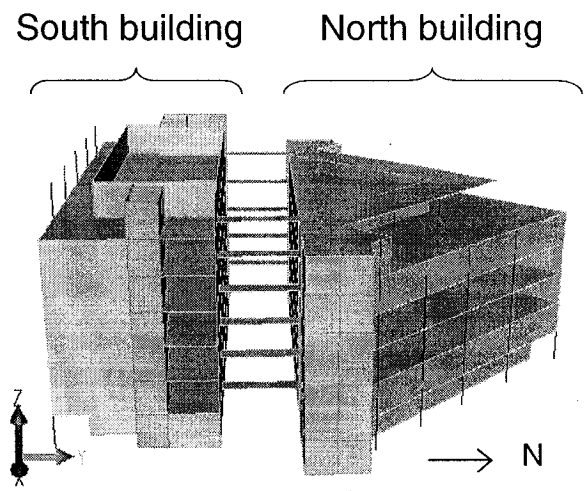


Figure 7.19 Expo98 test building: exploded view

The structural system is a reinforced concrete structure with vertical gravity and lateral subsystems consisting of rigid frames and shear walls, and horizontal subsystem consisting of two-level "slab-on-beam" floor assemblies. The building has four vertical circulation cores, which start from the ground level and are used for structural purposes. The roofs of the north and south buildings are long-span steel structures with cantilevers covering part of the buildings' terraces.

Given the **`Architectural Model`** as shown in Figures 7.18 and 7.19, the engineer can initiate the Architecture/Structure integration process following the methodology proposed in Chapter 4 and illustrated in Figure 4.8. The process is the same as described in the previous case study and therefore each activity is only briefly described in this second case study.

7.2.1 Pre-processing

As with the previous case study, the architectural model is created first alphanumerically using C++ code in the `drive` function. In this case, the architectural model also had to be slightly modified to fit the requirements of the software prototype. For the actual Expo98 building, most of the core **`Walls`** were not aligned with the structural gridlines. Some of these **`Walls`** were slightly moved to the grid lines, while some others left untouched because they were considerably out and rotated from the grid lines. This does not affect the integrity of the structure since for this particular situation the ***Configuration Algorithms*** take care of properly connecting all the rotated **`Walls`** to the rest of the structure.

7.2.2 Computer –assisted Architecture-Structure (A/S) integration

7.2.2.1 Inspection of the `Architectural Model`

The algorithm *findSupportsFromArchitecture* identifies the building core **`Walls`**, which have been circled in Figure 7.20. Office buildings usually have flexible open spaces where cubicles are organized using removable partitions. Space flexibility means that partitions could be moved whenever required and produce different cubicle organizations within the floor. The algorithm *findSupportsFromArchitecture* recognizes this flexibility by searching for permanent **`Walls`** only (i.e. not removable), as defined by the architect. Perimeter **`Walls`** are not selected either because they are curtain **`Walls`**.

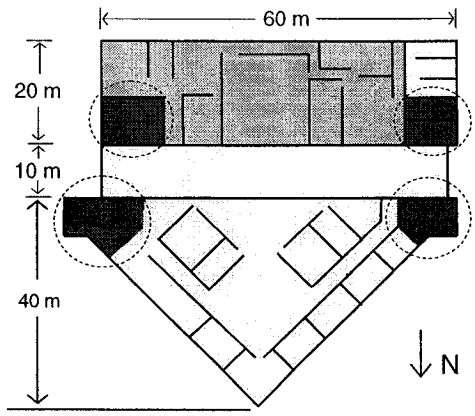


Figure 7.20 Expo98: typical **`Storey`**

A multi-functional room (i.e. for large meetings, conferences and social events) is located in the top **`Storey`** of the south building (see Figure 7.21). This is a large room requiring particular structural considerations, which must integrate well with the rest of the structure. After executing the algorithm *verifyStructuralLayoutConstraint* the engineer realizes that this space is intended to be **`Column-free`** and has a specified **`Floor-to-ceiling-height`** constraint of 4 m. In addition, the engineer also realizes (using the inspection algorithms) that the **`Walls`** at the perimeter of the room, towards the terrace, are curtain **`Walls`** and therefore cannot house **`Structural Elements`**, such as **`Columns`**, as required for supporting the roof. Therefore, the roof of the multi-functional room is required to span across the entire width of the south building thus making it a good candidate for long-span light-weight steel trusses.

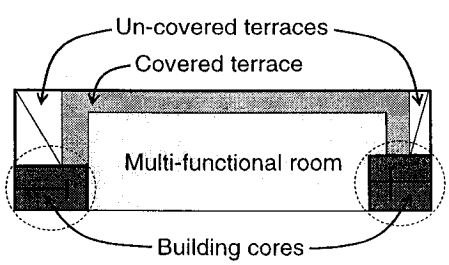


Figure 7.21 Expo98: plan view of the seventh **`Storey`** at the south building

As with the previous test case, the engineer may need to verify in advance if the core **Walls** (see Figure 7.20) are sufficient for providing lateral resistance and are well distributed in the floor plan to avoid unwanted torsional deformations. Potentially problematic geometries, such as the long cantilevers as seen in Figures 7.18 and 7.19, may also need close examination. As discussed with the previous case study, the current prototype is still unable to support those verifications in advance.

7.2.2.2 Configuration of the Structural System

a) Selection of Independent Structural Volumes

The engineer defines two **Independent Structural Volumes**, as shown in Figure 7.22, corresponding to the north and south buildings.

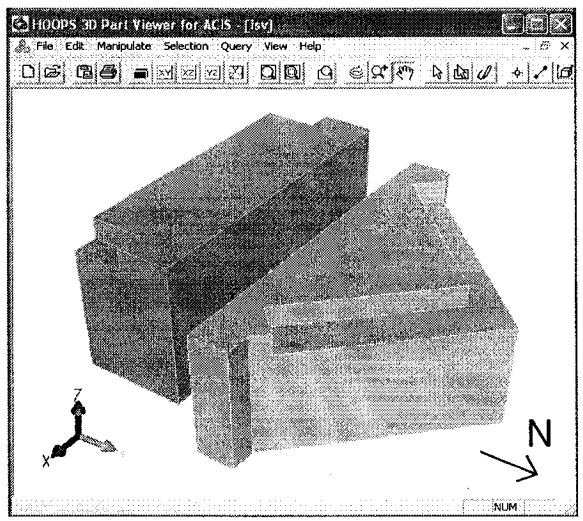


Figure 7.22 Expo98: two **Independent Structural Volumes**

b) Definition of `Structural System` type: `Structural Subsystems` and material(s)

The **`Vertical Gravity Subsystem`** consists of **`Column Stacks`**, the **`Vertical Lateral Subsystem`** consists of a combination of rigid **`Frame Assemblies`** and shear **`Wall Stacks`** made out of concrete and the **`Horizontal Subsystem`** consists of "slab-on-beam" **`Floor Assemblies`**.

c) Selection of `Structural Zones`

The engineer groups **`Spaces`** having similar structural characteristics into **`Structural Zones`**. For the south building, two structural zones have been devised, one grouping the multi-functional room and the covered terrace on the top floor (see Figure 7.21), and the other grouping the rest of the spaces. For simplicity, only one **`Structural Zone`** has been devised for the north building grouping all **`Spaces`**, which are assumed to have similar functionality and load requirements. Each **`Structural Zone`** is then assigned a corresponding applied load.

d) Definition and positioning of `Structural Assemblies`

For each **`Independent Structural Volume`**, the engineer positions **`Frame Assemblies`** in orthogonal directions, leaving 10 m structural bays in the south building, and 10 and 11 m structural bays in the north building. This bay dimensions provide floor spans that are within the limits for "slab-on-beam" **`Floor Assemblies`** (Schodek 2003). Figure 7.23 shows the **`Frame Assemblies`** with their abstract geometry.

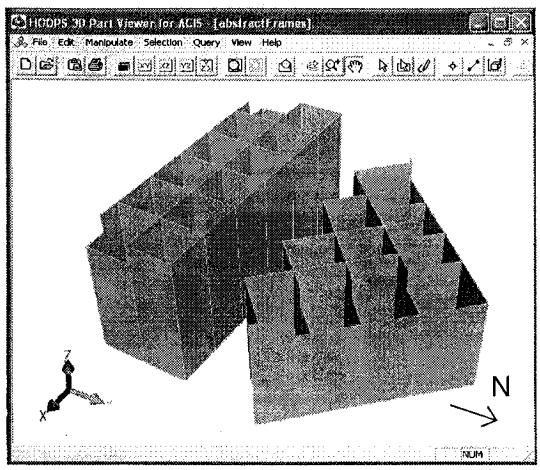


Figure 7.23 Expo98: abstract geometry of **`Frame Assemblies`**

The engineer defines **`Floor Assemblies`** from **`Aslabs`**. **`Independent Structural Volumes`** and **`Structural Zones`** subdivide **`ASlabs`** based on structural considerations. First, each **`ASlab`** is divided by the computer into two **`Floor Assemblies`**, one for each **`Independent Structural Volume`**. Then, the engineer uses the **`Structural Zone`** that contains the **`Column-free`** multi-functional room in the south building to create another **`Floor Assembly`** that becomes the roof of this **`Structural Zone`**. Figure 7.24 shows **`Floor Assemblies`** with their abstract geometry, where the **`Floor Assembly`** corresponding to the roof of the multi-functional room and the terrace is shown darker.

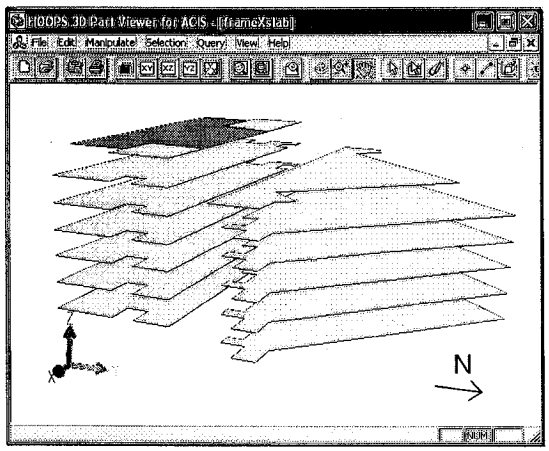


Figure 7.24 Expo98: abstract geometry of **`Floor Assemblies`**

At this point the engineer decides to add all encircled core **Walls** shown in Figure 7.20 to the **Structural System**, so that they become structural **Walls** which are then grouped by the computer into **Wall Stacks** as shown in Figure 7.25. Then, the computer adds each **Wall Stack** to a coplanar **Frame Assembly**, if applicable, and to corresponding **Vertical Lateral Subsystem** and **Vertical Gravity Subsystem**.

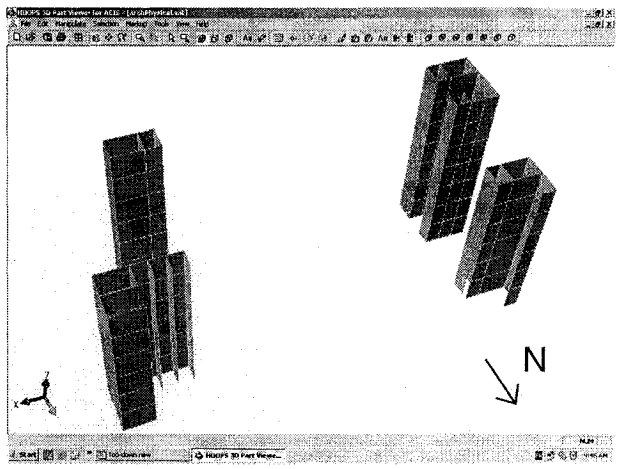


Figure 7.25 Expo98: **Wall Stacks** derived from the **Architectural Model**

e) Layout Structural Elements and connect them with Structural Connections

Configuration Algorithms are then executed to lay out **Structural Elements** following the initial layout provided by the abstract geometry of the **Structural Assemblies**. First, the algorithm *frameXframe* is executed to find intersections among the abstract geometry of orthogonal **Frame Assemblies** and to generate **Column Stacks** (see Figure 7.26).

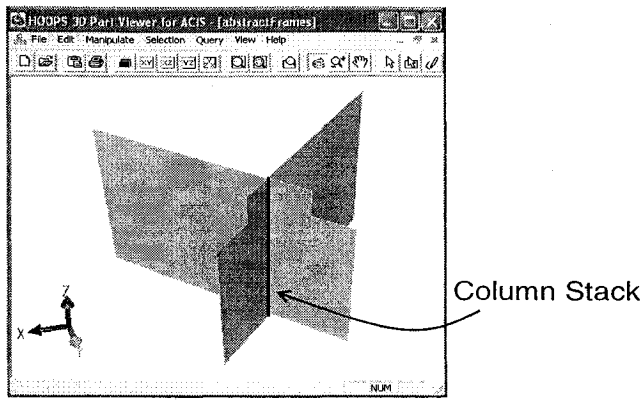


Figure 7.26 Expo98: the *frameXframe* algorithm generates a **`Column Stack`**

Then, as illustrated in Figure 7.27, algorithm *frameXfloor* finds intersection among the abstract planes of **`Frame Assemblies`** and **`Floor Assemblies`**, thus generating primary **`Beams`**.

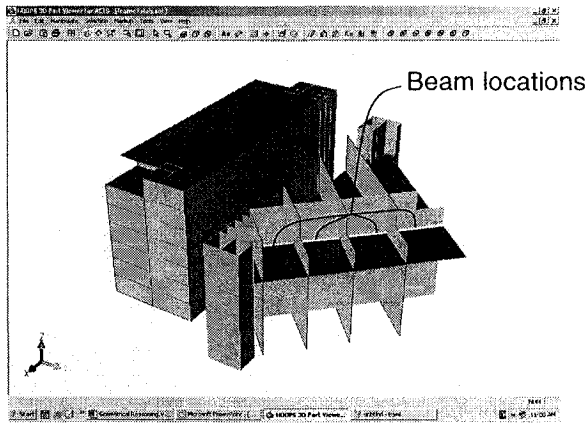


Figure 7.27 Expo98: the *frameXfloor* algorithm generates **`Beams`**

Note that since the roof of the **`Structural Zone`** containing the multi-functional room and the covered terrace is a long span structure, its beams are not laid out by the algorithm *frameXfloor*. Instead, the algorithm *uniformDepth* is executed automatically to generate the three-level (slab-on-beam-on-girder) roof structure (see Figure 7.28). At this point, a check may be required of the spans of the **`Floor Assemblies`** that have been laid out. An approximate depth is also likely to be required for each **`Floor Assembly`**

according to material and type, so that the **Floor_to_ceiling_height** constraint can checked by subtracting the depth of the upper floor minus the floor-to-floor height of the corresponding **Storey.**

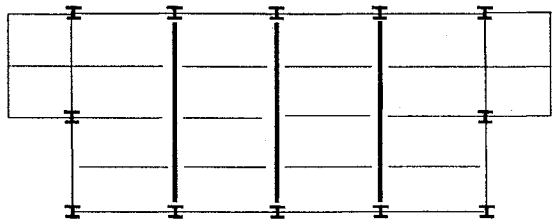


Figure 7.28 Expo98: three-level roof structure for the multi-functional room and the terrace

Finally, the algorithm *generateSlabElements* generates **Slab Elements** for each **Floor Assembly. Slab Elements** are then connected to supporting **Beams**. This completes the A/S integration process.

7.2.2.3 Verification of the integrity and stability of the Structural System

The algorithms *verifyGravityLoadPaths* and *verifyLateralStability* do not find structural discontinuities or lacks of supports. As with the previous test case, the above algorithms do not verify if the frames and shear **Walls** have sufficient capacity for providing lateral support while eliminating torsional instabilities. Figure 7.29 shows the populated **Vertical Gravity Subsystem** and Figure 7.30 shows the populated **Vertical Lateral Subsystem** in one direction, both for the north building.

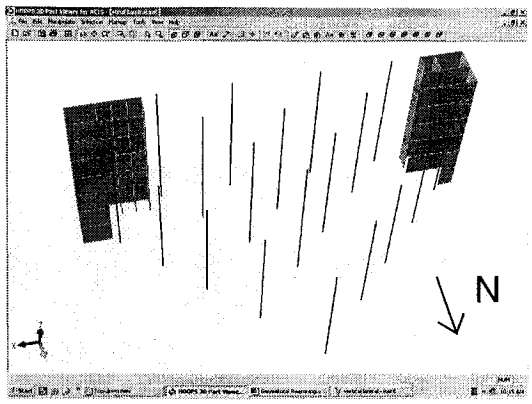


Figure 7.29 Expo98: the **`Vertical Gravity Subsystem`** for the north building

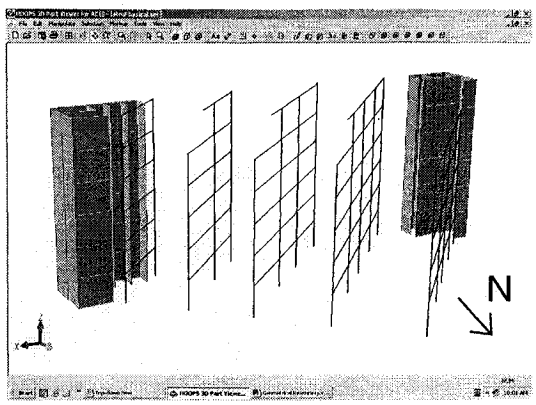


Figure 7.30 Expo98: the **`Vertical Lateral Subsystem`** in one direction for the north building

Figure 7.31 shows the entire **`Structural System`**. Note the long cantilevers in the last **`Storey`** of the north building (see also Figure 7.19). A simple heuristic rule from design practice (Eng-tips 2005) is used by the algorithm *verifyLengthOfCantilevers* (see Appendix C) to verify their lengths. This rule of thumb states that the length of a cantilever should not exceed 1/5 (for concrete) or 1/3 (for steel) of the length of the un-cantilevered span of a continuous **`Beam`**. In this case, the engineer is warned that some of these cantilevers exceed 1/3 of the un-cantilevered length. Then it is up to the engineer to communicate and convince the architect about this potential structural problem and/or devise a special roof system that should be compatible with the constraints (i.e. roof

depth and clear height of the **Space**) imposed by the architect for the **Space(s)** that it shelters. However, in order to convince the architect (and possibly the client) about this potential structural problem, a rule of thumb may not be sufficient and simplified analysis may be required to show **Beam** deflections.

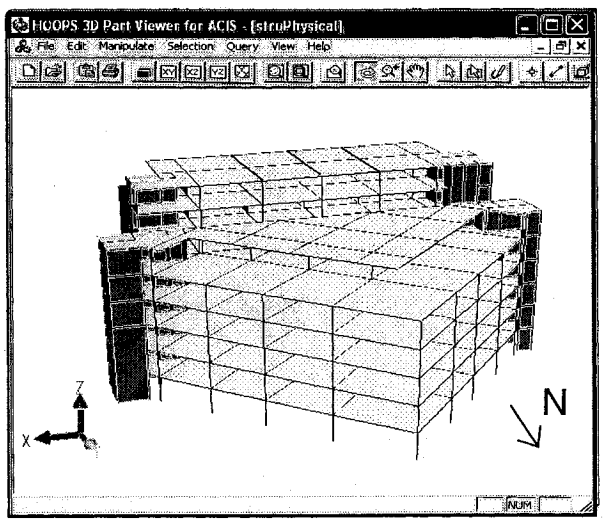


Figure 7.31 Expo98: the **Structural System** as generated with the prototype

7.3 Evaluation

The test cases demonstrate that the software prototype achieves its goal, which is to facilitate collaboration between architects and engineers during conceptual structural design for better integration between the architecture and the structure.

7.3.1 Advantages

The advantages of the proposed approach over current practice, as demonstrated by these test cases, can now be listed as follows:

- The top-down structural configuration process is a more efficient way of synthesizing the **Structural System**. Unlike current structural CAD applications, the prototype lets the engineer make decisions and lay out overall structural schemes at "higher-levels" in the structural hierarchy, which facilitates the subsequent generation of the physical **Structural System** by the software prototype.
- In addition, the top-down design approach guarantees compatibility among the different levels of the structural hierarchy. For example, in the test cases this is illustrated when a **Structural Zone** aggregating large **Column-free Spaces** imposes conditions on the vertical supports, as well as the type of roof system required. As another example, each **Structural Assembly** makes sure that the geometry and support conditions of each of constituent **Structural Elements** are consistent with its own type and geometry. In both test cases, the three-level (e.g. slab/beam/girder) **Floor Assemblies** at the roof of **Column-free Spaces** accept supports according to their support hierarchy.
- Geometric modeling and reasoning techniques provide support at different levels of design refinement and throughout the entire synthesis process via ***Synthesis Algorithms***. By contrast, current commercial structural engineering packages provide some capabilities for geometrical reasoning at the detailed level only (i.e. dealing with detailed geometry of **Structural Elements** and **Connections**). The software prototype enables geometric reasoning by providing the engineer with the algorithms for performing the following activities:
    - inspection of the building architecture,
    - selection of relevant architectural functional and physical entities,

- association of structural entities to architectural entities at different levels of the structural hierarchy,
- configuration of the structural system within the building architecture, and
- verification of the structural system.

This is in sharp contrast with most commercially available structural engineering applications that provide capabilities only for generation and verification of the physical structural system in isolation from the architecture.

- Since architectural functional requirements become explicit, they cannot be overlooked and therefore are guaranteed to be taken into consideration by the engineer during the structural synthesis process. This is achieved through the **`Structural-Layout Constraints`** that are associated with **`Structural Zones`**. In both test cases **`Structural Layout Constraints`** help the engineer in defining **`Structural Zones`**. In addition, the software prototype takes into consideration these constraints automatically for laying out the structural system locally, as illustrated in the test cases using the algorithm *uniformDepth*.
- The test cases demonstrate the impact of internal and external physical constraints (cf. section 4.3.2.2) in the structural synthesis process and applicable solution technologies. For example, in both cases a simple generative algorithm was appropriate for generating room framing solutions for large spaces with no internal physical constraints (i.e. the multi-functional sports rooms).
- The proposed methodology makes the engineer more aware of the structural opportunities in the building architecture. This is achieved by facilitating the

engineer's inspection of the building architecture. For example during the verification of the vertical continuity of **Walls**, the algorithms take into consideration their type.

- As a consequence of the above, the proposed methodology minimizes communication and coordination errors between architects and engineers. These errors may occur due to subtle architectural variations among floor plans that may be undetected by the engineer. Minimization of errors is provided by the following mechanisms: (1) providing tools enabling the inspection of the building architecture. For example, small shifts and rotations in **Walls** among different floor plans are not easily perceived by the engineer but are easily detected by synthesis algorithms like *verifyWallContinuity*, (2) enabling the direct integration of the **Structural System** into the **Building Architecture**. In this way, the engineer makes direct use of physical elements from **Building Architecture**. For example, if the engineer mistakenly selects a **Wall** that is not continuous down to the ground, the load path verification the prototype will detect the discontinuity and warn the engineer. By contrast in current design practice, if the engineer mistakenly assumes full **Wall** continuity the **Wall** is entered as such into the structural engineering application. As a result, the **Structural System** does not correspond to the actual architectural design. This error will only become apparent at later stages of the design process when design changes are more costly and difficult to make. In some cases, such errors may go undetected until construction time, which may then have serious consequences on the entire project budget and schedule.

It should also be realized that since for the test cases every operation has been carefully explained, this may give the impression that involved and time-consuming engineer participation is required to perform structural synthesis. However, the proposed A/S integration process consists of simple steps that involve inspecting the architecture, selecting few architectural entities (mainly `Walls` and `Spaces`), selecting overall structural properties, laying out structural assemblies, and verifying structural solutions.

7.3.2 Limitations

The test cases also uncover shortcomings of the prototype and the proposed approach that will have to be solved in order to support more effectively the synthesis process. These shortcomings are listed below:

- The first limitation is related to the degree of support that is provided by the design assistance environment for all the levels of functional and physical integration as described in sections 4.3.2.1 and 4.3.2.2. This limitation is related to the complexity of the building geometry. For simple building geometries, both, functional and physical integration are expected to be achieved straightforwardly. For complex building geometries, such as the ones of the test cases, the integration is not expected to be complete.
    - Considering the two levels of functional integration (i.e. intra-zone and inter-zone), only the first level (intra-zone) is currently being supported by the prototype. Nevertheless, in both test cases inter-zone integration was rather smooth since the patterns among structural zones were perfectly

matched in advance. The exercise of matching architectural and structural patterns is therefore not demonstrated with the test cases.

- Considering the four levels of physical integration, the design assistance environment supports the first two levels; that is where all structural elements fall within a structural grid. Support for levels 3 and 4, where structural elements fall outside the structural grid, is partially provided as demonstrated through the integration of skewed walls to the structure in the second test case.

- ***Inspection Algorithms*** are useful for identifying structural opportunities and constraints from the architecture. However, they do not facilitate the identification and assessment of potential structural problems coming from irregularities in the architecture. In both test cases it is not a straightforward task for the engineer to verify in advance if the core **`Walls`** from the architecture are sufficient for providing lateral support or if they are well distributed in the floor plan to avoid unwanted torsional deformations. In the second case study long cantilevers are present in the last **`Storey`**. In this case, the engineer has to wait until the model of the **`Structural System`** is built to verify their length. Due to the infinite variation of irregularities that can exist in the architecture, it has been found impractical to define parameters and rational rules to detect and quantify such irregularities in advance (Arnold and Reitherman 1982).

- In both test cases, ***Configuration Algorithms*** performed well. However, the lack of knowledge is noticeable when the structural layouts need to be verified. In the current prototype, the engineer should perform such verifications using his/her own

knowledge about loads, materials, constructability and available construction technologies. In the test cases, the prototype has no means for verifying floor spans and floor depths. The floor spans need to be within the thresholds specified by the specific **`Structural Assembly`** being laid out, while considering the applied loads. Floor depths also need to be verified to see if they provide the clear floor-to-ceiling height as specified by the architect.

- Considering ***Verification Algorithms***, in both test cases it is shown that a simplified method is required to verify if the frames and shear **`Walls`** have sufficient capacity for providing lateral support in any given direction while eliminating torsional instabilities. As discussed with the engineer of the first test case, in actual design practice the engineer has to construct the 3D model of the structure (i.e. incrementally using a time-consuming bottom-up approach) and run a time-consuming dynamic analysis to perform such verifications.

- In both test cases A/S integration was achieved. However, in neither of these cases feedback to the architect was fully demonstrated for two reasons:

    - First, the test cases are already refined buildings where the majority of the conceptual design problems have been solved. Unfortunately, as discussed in section 3.3.5 it is difficult to find architects and engineers that are willing to test a software prototype with actual conceptual design problems.

    - Second, in addition to geometry and topology, feedback to the architect must be based on structural engineering knowledge. Some limited

knowledge is included in the algorithms; for example, the algorithm *verifyLenghtOfCantilevers* incorporates a simple rule of thumb for the verification the length of cantilevers. However, for more complete support a knowledge-based component is required.

- Other limitations mentioned in Chapter 1, such as enabling changes and the management and evaluation of design alternatives were not included as part of this research and therefore they are not evaluated with the test cases.

7.4 Conclusions

In this Chapter, two test cases were presented that demonstrate the integration between the architectural design and the structural system at the conceptual stage. On the one hand, the methodology for A/S integration makes the engineer more aware of functional and physical constraints, as well as sources of difficulties and opportunities in the building architecture. When using traditional structural design tools structural engineers may overlook architectural concerns and subtle difficulties during early structural design stages. Thus, compared to the traditional design practice the proposed methodology proves to be more effective since it brings early structural design decisions to a higher level of resolution. On the other hand, the top-down approach allows two levels of integration of the **`Structural System`** to the **`Architectural Model`**, namely: integration at the functional and at the physical level. In addition, compared to traditional design practice, this is a more efficient approach for structural synthesis since high level decisions always guide low level actions. The approach also ensures compatibility among the different levels of the structural hierarchy.

Considering the shortcomings of the prototype, the test cases have shown that achieving A/S integration for complex architectural geometries still poses some difficulties that need to be solved for complete support. Particularly, in both test cases the prototype has not been able to provide satisfactory support locally, when inconsistencies exist between structural patterns and architectural design features. Therefore, for complex architectural geometries complete functional and physical integration is still not achieved. Another limitation comes from the lack of a knowledge-based module to provide knowledgeable support for decision making during inspection, configuration and verification of structural solutions. It should be mentioned that using already refined test cases diminished the capabilities of the prototype in supporting conceptual structural design. In particular, since conceptual design problems had already been solved in both test cases, feedback to the architect has not been required.

In spite of the above limitations, the discussions with practicing architects and engineers confirm that compared to current design practice the prototype shows improved computer assistance for early collaboration and structural model generation. The approach minimizes communication and coordination errors (that usually come from misinterpretation of architectural information by the engineer) while enabling a quicker generation of the 3D model of the structure. Therefore, the proposed design assistance environment for conceptual structural design fulfills its goal: to provide a more integrated design approach for improved communication and understanding between architects and engineers so that, not only errors and inconsistencies are minimized, but also problems detected, opportunities identified and constraints respected. This will hopefully lead to an improved overall building performance.

CHAPTER 8

SUMMARY, CONTRIBUTIONS AND FUTURE WORK

8.1 Summary

The goal of the engineer during conceptual structural design is to devise feasible structural systems that transfer loads to the ground safely and efficiently while acknowledging the form and function of the building architecture, i.e. responding to the design intents of the owner and the architect. A feasible structural system layout provides continuous load paths to the ground in a way that makes it structurally stable and realizable. The emphasis during conceptual structural design is on the synthesis of structural solutions (i.e. form and function). Consequently, during conceptual design the engineer makes decisions regarding the form of the structure by reasoning about its geometry and topology, while accounting for its function.

Current computer support for conceptual structural design is still inadequate. Commercial structural engineering applications rely on a constructive bottom-up approach for structural synthesis in which a structural model is built element by element in isolation from the building architecture and the other building engineering systems. This approach works well for small buildings. However, it becomes tedious and error-prone as building size and complexity increase. Some integrated structural applications enable the bottom-up construction of a structural model from architectural models. However, they reduce the process to the selection of few architectural elements and the construction of a structural model without consideration of architectural concerns. In addition, such

applications provide no assistance to the engineer to reason with architectural and structural models while looking for continuous load paths to the ground.

Considering research projects, most previous efforts have minimized the impact of architectural design on the structural synthesis process. With few exceptions, they focus on devising problem solving techniques rather than representational aspects. The main criticism comes from the simplification of the building architecture to fit into a problem-solving paradigm. By contrast, standardization efforts in building design and construction have been geared towards devising representations for supporting detailed design stages as well as manufacturing, construction and operation of building facilities. Providing support for conceptual design is still in its infancy.

The main premise of this research project is that in order to support properly the conceptual design stage of building structures, and enable timely engineering feedback to the architect, computers must allow engineers to perform design within a building architectural context without interrupting the creative workflow of the architect. To support this premise, interactions between architects and engineers during early building design have been investigated. Two stages of the early architectural design have been identified: a first stage where architectural representations are ambiguous, informal and vague (i.e. often private and not meant to be communicated to engineers); and a second stage where a first unambiguous computer representation of the building architecture is produced. Consequently, a distinction has been made between two sub-processes that are carried out by the engineer in response to the abovementioned stages of refinement of the architectural design, namely: structural layout planning and architecture/structure (A/S)

integration. This distinction is required because the input, the processing and reasoning, and the output of these two sub-processes are different. The computer requirements are also expected to be different.

During the structural layout planning stage, the engineer relies on overall building information (e.g. location and type of building, number of stories, approximate area of each storey, shape of the land, and the client's budget) and possibly sketches to give feedback to the architect. For the actual A/S integration stage a computer model of the building architecture must be available to the engineer. Three main activities have been identified for engineers while integrating the structural system to the building architecture: (1) inspection of the architectural model in which the engineer looks for problems, opportunities (i.e. load paths) and constraints within the architecture, (2) configuration of the structural system, and (3) verification of the model integrating the building architecture and the structural system. In addition, two interrelated types of constraints for A/S integration have been identified, namely: functional constraints imposed by the use of spaces, and physical constraints imposed by the construction elements that give shape to the building architecture. Physical constraints coming from the building architecture are classified either as internal constraints (i.e. restricting the internal geometry and topology of the structural system) or external constraints (i.e. affecting only the external shape of the structural system).

The above constraint definitions enable the decomposition of the structural synthesis process into global and local synthesis activities and the identification of computer technologies to support each activity, depending on the type and stringency of architectural constraints involved. Thus based on architectural considerations geometric

modeling and reasoning (GM/R) techniques have been identified as key for facilitating the inspection, configuration and verification (ICV) of a design model combining the building architecture and the structural system. GM/R techniques rely essentially on geometric (e.g. coplanar, collinear, parallel, etc.) and topologic (e.g. adjacent, overlapping, contained, etc.) relationships, as well as operations (e.g. Boolean union, difference and intersection) among entities in a geometric model to provide feedback about the model and enable its refinement.

As a result from the study of the interactions between architects and engineers during early building design, a formal model of the process of conceptual structural design has been devised that aims at describing the early collaboration between architects and structural engineers. This model describes the activities and constraints between architects and engineers during the structural layout planning and A/S integration stages. As part of this formal model, a methodology for A/S integration has been developed including the three general activities required for A/S integration: inspection of the building architecture, configuration of the structural system, and verification of the integrated model. Structural configuration is carried out following a top-down hierarchical refinement approach where the engineer defines independent structural volumes and structural zones first, followed by structural sub-systems, assemblies and elements, which are connected together through structural connections.

Based on the above methodology of A/S integration, a design assistance environment for conceptual structural design has been developed. Three components are required for such development: (1) a representation of the structural system and the building architecture,

(2) a geometric modeling and reasoning component, and (3) a knowledge-based reasoning component. Only the first two components have been investigated in this research project. Considering the first component, a semantically explicit representation is proposed that integrates standard concepts from the building architecture and the structural system. Considering the second component, synthesis algorithms are developed that rely on geometric modeling techniques and on domain-specific constraints encapsulated in the entities of the computer representation to allow the engineer to reason based on geometry and topology of the model being created.

A proof-of-concept software prototype has been implemented to demonstrate the A/S integration methodology and to test the components of a design assistance environment for conceptual structural design. The prototype relies on a commercial geometric modeling kernel for representing and manipulating geometry and uses another commercial interface for visualization.

Two test cases demonstrate improved assistance for A/S integration during conceptual structural design following the proposed A/S integration methodology, and using the components of an environment for conceptual structural design. The test cases demonstrate that the proposed approach is a more effective and integrated way of assisting engineers in the synthesis of structural solutions while considering the architect's design intentions. The test cases also show that there is still work to be done for providing improved assistance to the engineer, such as supporting complex architectural geometries while including knowledge about construction technologies, costs and materials.

8.2 Main research contributions

Architects and engineers think differently, however they also have much knowledge in common that become relevant during conceptual building design. This research project proposes a new approach to conceptual structural design that takes advantage of this common knowledge between architects and structural engineers. Most previous research efforts in the field aimed at making the engineer more aware of the architect's concerns. However, they did so by bringing those concerns into the engineer's design environment. By contrast, instead of bringing the architect closer to the engineer, this research project aims at bringing the engineer closer to the architect, thus following actual practice whereby the architect requests structural advice, and not the other way around (i.e. the engineer requests architectural advice). Therefore, this research is an initial attempt in understanding the architect's work and acknowledging his/her active involvement in the conceptual design of building structures. Consequently, the contributions of this research project are divided in the following sub-sections.

8.2.1 Identify and organize factors for architecture/structure integration

The following factors for architecture/structure integration have been identified and organized:

- Two main stages of conceptual structural design have been identified in response to the building architectural design, namely: structural layout planning and architecture/structure (A/S) integration. The communication, processing and

reasoning, and feedback provided by the engineer for these two stages are different. This research project has focused on supporting the A/S integration stage.

- Two degrees of integration between the structural system and the building architecture have been identified: functional and physical integration. The distinction between the two types of integration is essential since each type poses different constraints for structural synthesis.
- Two levels of functional integration have been identified (cf. section 4.3.2.1). These levels account for the architectural functionality that affects the structural integration. The first level has been implemented and demonstrated with the test cases (cf. section 7.3.2).
- Four levels of physical integration have been identified using set theory (cf. section 4.3.2.2). These levels provide a clear description of the opportunities for integration, as well as the difficulties faced by engineers and developers of applications aimed at supporting the structural synthesis process. The first two levels have been implemented and demonstrated in the test cases (cf. section 7.3.2).
- Two generic types of constraints (i.e. internal and external) posed by the building architecture have been identified. As demonstrated with the test cases and discussed in section 7.3.1, the presence and stringency of these constraints ultimately determine suitable problem solving approaches for structural synthesis.
- Based on the architectural constraints, the problems faced by engineers during the structural synthesis process have been decomposed into global and local sub-problems or sub-activities (cf. section 4.3.3). Suitable computer technologies have been identified for providing support for each activity (cf. section 4.3.4). This point

has been illustrated with the test cases in Chapter 7 since in both cases a simplified generative technique was suitable for floor framing layout of large multi-functional spaces (i.e. local-cells), which are free from internal physical constraints. Previous research projects have concentrated on solving specific conceptual design sub-problems without regard to the larger context of the problem that cannot be solved with a single computer technology.

8.2.2 Devise a formal model of conceptual structural design

The formal model of conceptual structural design has the following innovative features:

- It is a first attempt in defining a model of two-way interactions between architects and engineers during conceptual structural design. The test cases demonstrate how these interactions can take place.
- It takes into consideration the different architectural representations that can be made available to the engineer, such as qualitative and quantitative parametric descriptions, sketches and digital building models. However, the focus of this research has been in supporting structural synthesis from digital building models of the architecture.
- It defines activities to be performed by the structural engineer that respond to the architect's own design workflow of activities. Consequently, it identifies when and how informed feedback can be provided to the architect depending on the architectural representations available. Due to a lack of a knowledge-based

component, as well as analysis and evaluation methods, engineering feedback has not been fully demonstrated with the test cases.

8.2.3 Devise a methodology for architecture/structure integration

The methodology for A/S integration brings the following advantages over existing and already proposed approaches for conceptual structural design:

- The methodology tackles the entire A/S integration process, as opposed to concentrating on a particular sub-activity or hierarchical level. This is in contrast to previous research projects for conceptual and preliminary structural design that describe and support only one sub-activity of the process.
- It incorporates three main activities: inspection, configuration and verification. The first activity acknowledges that the A/S integration process for structural synthesis begins with a throughout inspection of the building architecture in order to find potential structural opportunities, detect structural problems and uncover architectural constraints, as well as the owner and the architect's intent in the design.
- Following a top-down approach for structural configuration, the engineer is in charge of making major decisions such as selecting structural system type and material(s), as well as subsystems and assemblies, and positioning them, while the computer takes care of time-consuming tasks, such as arranging and connecting elements into assemblies, under the engineer's instruction.. This is in sharp contrast with current commercial applications that enable an incremental (i.e. bottom-up) approach for structural synthesis

- Thus, through structural problem decomposition, the methodology lends itself to the configuration of global and local structural solutions that respond to architectural functional and physical constraints.
- It enables two-way interactions between architects and engineers at each stage of design refinement. Engineering feedback to the architect can be provided as a result of the inspection, configuration and verification of the integrated model. However, as discussed previously due to a lack of a knowledge-based component, as well as evaluation and analysis methods, engineering feedback has not been fully demonstrated with the test cases.

8.2.4 Develop the main components of a design assistance environment for conceptual structural design

The following two components have been developed and implemented to provide a design assistance environment for conceptual structural design.

a) ***Integrated Representation*** - The semantically explicit representation developed in this research project has the following innovative features that make it suitable for supporting conceptual structural design.

- It is tailored-made for conceptual design of building structures. This brings simplicity to the representation by eliminating redundancies usually associated with unnecessary concepts, entities and relationships that are required only at later building lifecycle stages. Representational simplicity also facilitates design refinement and exploration.

- The semantic concepts that are incorporated in the representation are standard concepts that are well defined in the architectural and structural engineering domains, and therefore can be explicitly represented without limiting the flexibility and extensibility of the representation.

- It supports design refinement. This is achieved by including purely functional building entities (i.e. that are required for considering the intentions of the designer, but do not have physical representations) such as structural massings, structural subsystems and structural assemblies. In the test cases it has been demonstrated how these purely functional entities effectively assist engineers in making major structural decisions while following the methodology for A/S integration. In addition, these entities play the key role of ensuring compatibility between these major decisions and the resulting physical layouts.

- It includes basic design intents by the architect that become apparent to the engineer. This is achieved by including structural layout constraints associated to functional spaces (e.g. column-free, floor-to-ceiling-height) and providing wall types that are taken into consideration by the problem-solving algorithms.

b) Synthesis Algorithms - Through these algorithms it has been demonstrated that reasoning based on geometry and topology, while considering architectural and structural constraints, can be successfully applied to develop computer environments for conceptual structural design assistance. The test cases have illustrated how these algorithms help the engineer in accomplishing the main goal of conceptual structural design, which is to lay out feasible structural systems that transfer loads to the ground efficiently, while acknowledging the form and function of the building

architecture. Specifically, this is done by assisting the engineer in the following tasks.

- Looking for continuous load paths to the ground and detecting potential structural problems within the building architecture. With the test cases, however, it was found that providing assistance in detecting structural problems in the architecture is difficult due to the infinite variation of irregularities that can exist in the architecture.
- Configuring structural solutions efficiently while respecting structural layout constraints imposed by the architect.
- Verifying the integrity and stability of structural solutions. The test cases demonstrate integrity and simple stability checks; however, simplified methods are still lacking for verifying structural safety and stability, particularly for irregular buildings for which the only solution so far is to resort to "expensive" 3D dynamic analysis.

8.3 Directions for future research

A number of research ideas were identified during the course of this work as extensions to this research project. Some of these ideas originated from the premises and requirements that were identified at the beginning of this project, while others emerged during the later part of this research. The overall directions for future research projects are briefly discussed below.

- ***Facilitate design changes*** – During conceptual design, changes are the norm rather than the exception. The topic of enabling changes was not studied in this research project. Nowadays, parametric techniques are the main mechanism for enabling changes with computers. However, as design products become more complex, parametric associations may over-constrain or threaten the integrity of the design model. For example, most products integrate various subsystems (horizontal integration), which are in turn described by different levels of abstraction/decomposition (vertical integration). For such products, changes need to be propagated in five dimensions, namely: the three-dimensional Euclidean space, the horizontal integration space and the vertical integration space. Design changes are likely to affect the geometry and/or the topology of the model being designed. When only the geometry is affected the propagation of changes becomes simpler. However, when the model topology is affected by changes, entities need to be added and/or deleted from the model. Thus, the integrity of the design model may be affected and the negative effects of changes propagated either towards related subsystems (i.e. horizontally) and/or other levels of abstraction/decomposition (i.e. vertically).
- ***Modeling knowledge*** - Synthesis algorithms currently provide assistance to the engineer, based mostly on geometry and topology concerns. However, in this research project it has been found that structural engineering knowledge is also necessary during the synthesis process for providing more competent communication between the engineer and the computer, thus enabling more informed decision-making. A knowledge-based reasoning framework should therefore be developed that

incorporates as well knowledge regarding construction technologies, materials and cost.

- ***Management, analysis and evaluation of design alternatives*** – A mechanism needs to be developed for managing architectural and structural design alternatives. Simplified analysis methods such as the ones proposed by Lin and Stotesbury (1988) need to be studied to help evaluate structural problems, at different levels of the structural hierarchy, coming from irregularities in the architectural form, as well as the form and layout of spaces and space establishing elements. Evaluation mechanisms also need to be examined that include not only structural factors, but also qualitative architectural factors that reflect how the structural system respects architectural constraints and responds to the architect's intent.
- ***Facilitate structural synthesis for more complex building geometries*** - Further research is required to assist the engineer in devising structural solutions for even more complex building geometries. Such research should include a study of the mechanisms that can let the engineer assess structural problems and inconsistencies derived from irregularities in the architecture, and help the architect and the engineer find feasible solutions. For example, helping architects and engineers in efficiently finding the most suitable combination of structural patterns among dissimilar stories and architectural functional zones is an interesting future extension of this research project. A related extension is to investigate more effective support for the synthesis of structural solutions for buildings such as airports, museums and other types of symbolic buildings for which the structural solutions are more challenging. This involves improving the integrated representation that has been proposed in this

research, and developing more powerful synthesis algorithms for assessing the structural feasibility of complex building architectures and enabling the configuration of advanced structural systems.

- ***Consider other engineering domains*** – In this research project, the only two building domains that have been considered for integrated conceptual structural design are the architectural and the structural domains. Efforts should be devoted to study the effects of major decisions regarding the envelope, mechanical, plumbing, electrical, and communication systems in the selection and layout of the structural system during conceptual structural design.

The above directions for future research show that there is still much research to be done in developing computer applications for effectively assisting the conceptual design of building structures. The results of this research project constitute one step forward in a lengthy process that will hopefully lead to improved conceptual structural design practice, hence to better building in the long term.

References

Augenbroe G. (1992). "Integrated Building Performance Evaluation in the Early Design Stages", *Building and Environment*, Elsevier Science Ltd., Vol. 27, No. 2, pp. 149-160.

Arnold C. and Reitherman R. (1982). *Building Configuration and Seismic Design,* John Wiley & Sons, New York, 296 p.

Bachmann H. (2003). Seismic Conceptual Design of Buildings- Basic Principles for Engineers, Architects, Building Owners and Authorities, Swiss Federal Office of water and Geology, BBL, Vertrieb Publikationen, CH-3003 Bern, 81 p.

Bailey S.F. and Smith I.F., (1994). "Case-Based Preliminary Building Design", *Journal of Computing in Civil Engineering*, ASCE, Vol. 8, No. 4, pp. 454-467.

Bédard C. and Gowri K. (1990). "Automating Building Design Process with KBES". *Journal of Computing in Civil Engineering*, Vol. 4, No. 2, ASCE, pp. 69-83.

Bédard C. and Ravi M. (1991). "Knowledge-Based Approach to Overall Configuration of Multistory Office Buildings". *Journal of Computing in Civil Engineering*, Vol. 5, No. 4, ASCE, pp. 336-353.

Biedermann, J. and Grierson, D. (1995). "A Generic Model of Building Design", *Engineering with Computers*, Vol. 11, No. 3, Springer-Verlag, pp. 173-184.

Björk B.C. (1992). "A Conceptual Model of Spaces, Space Boundaries and Enclosing Structures", *Automation in Construction*, Elsevier Science, Vol. 1, No 3, pp. 193-214.

Booch G., Rumbaugh J. and Jacobson I. (1999), *The Unified Modeling Language User Guide*, Addison-Wesley Longman Inc, Reading, MA, 482 p.

Brown, K. M. and Charles, C. B. (1995). *Computers in the Professional Practice of Design*, McGraw Hill, New York, 227 p.

Cambridge Dictionary (2005). Cambridge Online Dictionary, Internet Website: http://dictionary.cambridge.org/, [last consulted: March 2004].

Christianson P. (2000). "Properties of the virtual building", *Artificial Intelligence in Design* '00, J.S. Gero (ed.), pp. 2909-2919.

Cross N. (1989). *Engineering Design Methods*, 1st Ed., John Wiley & Sons Ltd, New York, USA, 159 p.

Datta S. and Woodbury R. (2004). "Feature nodes: An interactive construct for sharing initiative in design exploration", *Design Computing and Cognition '04*, J.S. Gero (ed.), pp. 23-36.

Dias W.P.S. (1996). "Multidisciplinary Product Modeling of Buildings", *J. Comput. Civ. Eng.*, Vol. 10, No. 1, ASCE, pp. 78-86.

Eng-tips (2005). Eng-Tips Forums, Internet Website: http://www.eng-tips.com/, [last consulted: January 2005].

Expo 98 (2004). EPR Architects Ltd, Internet Website: http://www.epr.co.uk/architect_offices_expo98hq.html [last consulted: December 2004]

Fenves S.J., Choi Y, Gurumoorthy B., Mocko G. and Sriram R. (2004). "Master Product Model for the Support of Tighter Integration of Spatial and Functional Design", *report NISTIR 7004*, National Institute of Standards and Technology (NIST), Technology Administration, U.S. Department of Commerce.

Fenves S.J., Rivard H. and Gomez N., (2000). "SEED-Config: a Tool for Conceptual Structural Design in a Collaborative Design Environment", *Artificial Intelligence in Engineering*, Elsevier Science, Vol. 14, pp. 233-247.

Fenves S.J., Flemming U., Hendrickson C., Maher M.L., Quadrel R., Terk M. and Woodbury R. (1994). *Concurrent Computer-Integrated Building Design*, 1st Ed., PTR Prentice Hall, Englewood Cliffs, NJ, 242 p.

Flemming U. (1990). "Knowledge Representation and Acquisition in the LOOS System", *Building and Environment*, Vol. 25, No. 3, pp. 209-219.

Froese T. (1994). "Information Standards in the AEC Industry", *Canadian Civil Engineer*, Vol. 11, No 6, September 1994, pp. 3-5.

Fuyama H., Law K.H., Krawinkler H., (1997). "An Interactive Computer Assisted System for Conceptual Structural Design of Steel Buildings", *Computers and structures*, Elsevier Science, Vol. 63, No. 4, pp. 647-662.

Gero, J. S. (1986). "An overview of knowledge engineering and its relevance to CAAD", *in* A. Pipes (ed.), *CAAD Futures*, Butterworths, Guildford, pp. 107-119

Gero J.S. (1990). "Design prototypes: a knowledge representation schema for design", *AI Magazine*, (11) 4, pp. 26-36.

Gero, J.S. (2004). *Foreword for the First International Conference on Design Computing and Cognition (DCC'04)*, MIT, Cambridge, USA, Internet Website: http://www.arch.usyd.edu.au/kcdc/conferences/dcc04/

Giarratano J. and Riley G. (1998). *Expert Systems: Principles and Programming*, 3 Ed., PWS Publishing Company, Boston, MA, USA, 597 p.

Grierson D.E. and Park K.-W. (1993)."Optimal Sizing, Geometrical and Topological Design using a Genetic Algorithm", *Structural Optimization*, Vol. 6, Springer, pp. 115-159.

Grierson D.E. and Khajehpour S. (2002). "Method for Conceptual Design Applied to Office Buildings", *Journal of Computing in Civil Engineering*, ASCE, pp. 83-102.

Gursoz E.L., Choi Y. and Prinz F.B., (1991). "Boolean Set Operations on Non-Manifold Boundary Representation Objects", *Computer-Aided Design*, Elsevier Science, Vol. 24, No. 1, pp. 33-38.

Haymaker J., Fischer M., Burke K. and McDonough W. (2004). "Tools to Signal C2C Design Intention", *W78 Workshop*, Integrated IT to Support Sustainable Construction, Toronto, Canada, May 7-8, 2004, 12 p.

Heisserman J. (1994). "Generative Geometric Design", *IEEE Computer Graphics and Applications*, (14) 2, pp. 37-45.

Heisserman J., Callahan S. and Mattikali R, (2000). "A Design Representation to Support Automated Design Generation", *Artificial Intelligence in Design* '00, J.S. Gero (ed.), pp. 545-566.

Holgate A. (1986). *The Art in Structural Design: An Introduction and Sourcebook*, Oxford University Press, New York, 337 p.

Huey S. M., North T. L. and Birchler A. (2000). "A Whole Experience", *Civil Engineering Journal,* ASCE, August 2000, pp. 58-65.

IAI (2004). International Alliance for Interoperability, IAI, Internet Website: http://ce.vtt.fi/iaiIFCprojects/ [last consulted: March 2004].

ISO-STEP (2004). ISO Standard 10303: Standard for the Exchange of Product Model Data, Internet Website: http://www.steptools.com/library/standard/, [last consulted: February 2005].

Jain D., Krawinkler H. and Law K.H., (1991). "Logic-Based Conceptual Structural Design of Steel Office Buildings", *Technical Report Number 49*, Center for Integrated Facility Engineering, Stanford University, 148 p.

Khemlani L., Timerman A., Benne B and Kalay Y. (1998). "Intelligent Representation for Computer-aided Building Design", *Automation in Construction*, Elsevier Science, Vol. 8, No. 1, pp. 49-71.

Kirkpatrick S., Gelatt Jr, C.D. and Vecchi M.P. (1983). "Optimization by Simulated Annealing", Science, (220) 4598, pp. 671-679.

Krishnamourthy C.S. and Rajeev S. (1996). *Artificial Intelligence and Expert Systems for Engineers*, Boca Raton : CRC Press, 297 p.

Kumar B. and Raphael B. (1997). "CADREM: A Case-based System for Conceptual Structural Design", *Engineering with Computers*, 13(3), pp. 153-164.

Law, K., Barsalou, T. and Wiederhold, G. (1990). "Management of Complex Structural Engineering Objects in a Relational Framework", *Engineering with Computers*, Vol. 6, Springer-Verlag, pp. 81-92.

Leclercq P. (1996). *Environnement de Conception Architecturale Préintegrée : Eléments d'une plate-forme d'assistance basée sur une représentation sémantique*, Ph.D. thesis, Faculté des Sciences Appliquées, Université de Liège, Belgium, 243 p.

Lin T.Y. and Stotesbury S.D (1988). *Structural Concepts and Systems for Architects and Engineers*, 2nd Ed. Van Nostrand Reinhold, New York, 507 p.

Lipson H. and Shpitlani M. (2000). "Conceptual design and analysis by sketching", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 14, pp. 391-401.

Luger G. F. and Stubblefield U.A. (1989). *Artificial Intelligence and the Design of Expert Systems,* Redwood City, Calif. : Benjamin/Cummings Pub. Co., 660 p

Luth, G. P., Krawlinker H. and Law K., (1991). "Representation and Reasoning for Integrated Structural Design", *Technical Report Number 55*, Center for Integrated Facility Engineering, Stanford University, 259 p

Maher M.L. (1985). *HI-RISE: A Knowledge-Based Expert System for the Preliminary Structural Design of High Rise Buildings,* Ph.D. Thesis, Department of Civil Engineering, Carnegie Institute of Technology, Carnegie Mellon University, Pittsburgh, PA., p. 125.

Mantyla M. (1988). *An Introduction to Solid Modeling*, Computer Science Press, 401 p

Martini K. and Powell G.H. (1990). "Geometric Modeling Requirements for Structural Design", *Engineering with Computers*, Vol. 6, Springer-Verlag, pp. 93-102.

Meniru K., Rivard H. and Bédard C. (2002) "Specifications for Computer-Aided Conceptual Building Design", *International Journal of Design Studies*, Elsevier Science Ltd, Vol. 24, No. 1, pp. 51-71.

Merriam-Webster Dictionary (2005). Merriam-Webster Online Dictionary, Internet Website: http://www.m-w.com, [last consulted: March 2004].

Meyer S. and Fenves S.J., (1993). "Structural Design of Tall Buildings Knowledge Acquisition Study Report", Engineering Design Research Center, Department of Civil Engineering, Carnegie Mellon University, PI, PA, 52 p

Meyer S. (1995). *A Description of the Structural Design of Tall Buildings Through the Grammar Paradigm*, Ph.D. Thesis, The Engineering Design Research Center, Dept. of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA, 159 p

Mokhtar A., Bédard C. and Fazio P., (1998). "Information Model for Managing Design Changes in a Collaborative Environment". *Journal of Computing in Civil Engineering*, ASCE, Vol. 12, No. 2, pp. 82-92.

Pillai S.V., Kirk D.W. and Erki M.A. (1999). *Reinforced Concrete Design*, 3rd Edition, McGraw-Hill Ryerson Limited, 630 p.

Rafiq M.Y., Mathews J.D. and Bullock G.N. (2003). "Conceptual Building Design – Evolutionary Approach", *Journal of Computing in Civil Engineering*, ASCE, Vol. 17, No. 3, pp. 150-158.

Raphael B. and Smith I.F.C. (2003). *Fundamentals of Computer-Aided Engineering*, John Wiley & Sons Ltd, England, 306 p.

Ravi M., (1998). *Knowledge-Based System Approach to Integrate the Design of Multistorey Office Buildings at the Preliminary Stage*, Ph.D. Thesis, Centre for Building Studies, Department of Civil, Building and Env. Engineering, Concordia University, Montreal, QC, Canada.

Rivard H. and Fenves S.J. (2000a) "A Representation for Conceptual Design of Buildings", *Journal of Computing in Civil Engineering,* ASCE, Vol. 14, No. 3, pp. 151-159.

Rivard H. and Fenves S.J. (2000b). "SEED-Config: A Case-based Reasoning System for Conceptual Building Design", *Artificial Intelligence in Engineering Design, Analysis and Manufacturing*, Cambridge University Press, Vol. 14, pp. 415-430.

Rivard H., Mora R. and Bedard C., (2000c). "Geometrical Reasoning and the Conceptual Design of Building Structures", 8th International Conference in Computing in Civil and Building Engineering, ICCCBE-VII, Stanford University, Stanford, California, USA.

Rivard H., Bédard C., Fazio P. and Ha K.H., (1995). "Functional Analysis of the Preliminary Building Envelope Design Process". *Building and Environment*, Elsevier Science Ltd., Vol. 30, No. 3, pp. 391-401.

Rosenman, M. A. and Gero, J. S. (1996). "Modeling Multiple Views of Design Objects in a Collaborative CAD Environment", *Computer-Aided Design*, Vol. 28, No. 3, Elsevier Science, pp. 193-205.

Rossignac J. and O'Connor M. (1990). "SGC: A Dimensions Independent Model for Point sets with Internal Structures and Incomplete Boundaries", Wozny MJ, Turner JU, Preiss K, editors, *Geometric Modeling for Product Engineering*. North-Holland.

Rossignac J.R. and Requicha A.A., (1991). "Constructive Non-Regularized Geometry", *Computer-Aided Design*, Elsevier Science, Vol. 23, No. 1, January/February 1991, pp. 21-31.

Sacks R. and Warszaiwki A. (1997). "A Project Model for an Automated Building System: Design and Planning Phases", *Automation in Construction*, Vol. 7, No. 1, Elsevier Science, pp. 21-34.

Sacks R., Eastman C.M. and Lee G. (2004). "Parametric 3D Modeling in Building Construction with Examples from Precast Concrete", *Automation in Construction*, Vol. 13, No. 2, Elsevier Science, pp. 291-312.

Sauce, R., Martini, K. and Powell G.H. (1992). "Object-Oriented Approaches for Integrated Engineering Design Systems", *J. Comput. Civ. Eng.*, Vol. 6, No. 3, ASCE, pp. 248-265.

Scherer, R.J. and Gehre A. (2000). "Approach to a Knowledge-based Design Assistant System for Conceptual Structural System Design", *Product and Process Modeling in Building and Construction", Proc.* ECPPM 2000, Lisbon, Portugal, pp. 229-238

SCI (2004). The Steel Construction Institute, CIMsteel Integration Standards release 2, Second Edition, Internet Website: http://www.cis2.org/index.htm, [last consulted: March 2004].

Schaeffer, R. E. (1998). *Elementary Structures for Architects and Builders*, 3rd ed., Prentice Hall, Upper Saddle River, New Jersey, 449 p.

Shai O. (2001). "Combinatorial Representations in Structural Analysis", *Journal of Computing in Civil Engineering*, ASCE, Vol. 15, No. 3, pp. 193-207.

Shea K. and Cagan J. (1998). "The Design of Novel Roof Trusses with Shape Annealing: Assessing the Ability of a Computational Method in Aiding Structural Designers with Varying Design Intent", *Design Studies*, 20, pp. 3-23.

Shea K. (2004). "Explorations in using an Aperiodic Spatial Tiling as a Design Generator", *Design Computing and Cognition'04*, J.S. Gero (Ed.), pp. 137-156.

Schmitt G. (1988). *Microcomputer Aided Design for Architects and Designers*, 1st Ed., John Wiley and Sons Inc., USA, 208 p.

Schodek, D.L. (2003). *Structures*, 2nd ed. Prentice Hall, Inc., Englewood Clifs, New Jersey, 585 p.

Simon H.A., (1996). *The Sciences of the Artificial,* 3 Ed., The MIT Press, Cambridge, Massachusetts, 231 p.

Sisk G.M., Miles J.C. and Moore C.J. (2003). "Designer Centered Development of GA-Based DSS for Conceptual Design of Buildings", *Journal of Computing in Civil Engineering*, ASCE, Vol. 17, No. 3, pp. 159-166.

Soibelman L. and Peña-Mora F. (2000). "Distributed Multi-Reasoning Mechanism to Support Conceptual Structural Design", *Journal of Structural Engineering*, ASCE, pp. 733-742.

Spatial Corp. (2004). Spatial Corp. by Dassault-Systems, France, Internet Website: http://www.spatial.com/ [last consulted: March 2004].

Sriram R.D., Wong A. and He L, (1995). "GNOMES: An Object-Oriented Non-Manifold Geometric Engine", *Computer-Aided Design*, Elsevier Science, Vol. 27, No. 11, pp. 853-868.

Stiny, G, (1980). "Introduction to Shape and Shape Grammars." *Environment and Planning B: Planning and Design* 7, 343-351.

Stouffs R. and Krishnamurti R. (2004). "Data Views, Data Recognition, Design Queries and Design Rules: Representational Flexibility for Design", *Design Computing and Cognition'04*, JS Gero (Ed), Kluwer Academic Publishers, pp. 219-238.

Taranath B. S. (1998). *Steel, Concrete, and Composite Design of Tall Buildings.* 2nd Ed., Mc Graw-Hill, New York, USA, 998 p.

Thiel P. (1963). Notes on *Environmental Space and Elementary Space Notation*, College of Architecture and Urban Planning, University of Washington, USA.

TSA Inc. (2004). Tech Soft America, Internet Website: http://www.hoops3d.com/about/hoops_scene/hoops_scene.htm

Turk Z. (2001). "Phenomenological Foundations of Conceptual Product Modeling in Architecture, Engineering and Construction", *J. of Artificial Intelligence in Engineering*, Elsevier Science, No 15, pp. 83-92.

Unigraphics (2004). UGS, Internet Website: http://www.ugs.com/products/open/parasolid/, [last consulted: February 2005].

Winters N. (1997). *Architecture is Elementary: Visual Thinking through Architectural Concepts*. Gibbs Smith, Publisher, Salt Lake City, USA, 255 p.

Woodbury R. and Damski C. (1997). "Geometric Representation and Reasoning in Design", *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Guest Editorial, Cambridge University Press, Vol. 11, pp. 243-244.

Zamanian M.K., (1992). *Modeling and Communicating Spatial and Functional Information about Constructed Facilities*, Ph.D. Thesis, A National Science Foundation Engineering Research Centrer, Carnegie Mellon University, Pittsburgh, PA., 153 p.

APPENDIX A

LIST OF DESIGN SOFTWARE APPLICATIONS

A.1 Architectural CAD Software (cf. section 2.3.1)

Name	Developer	Website
Architectural Studio	Autodesk	http://www.autodesk.com/
SketchUp	@Last Software	http://www.sketchup.com/about.php

Table A.1 Architectural CAD software for conceptual design

Name	Developer	Website
ArchiCAD	Graphisoft	http://www.graphisoft.com/products/archicad/
Architectural Desktop	Autodesk	http://www.autodesk.com/
Microstation Triforma	Bentley	http://www.bentley.com/
Revit	Autodesk	http://www.autodesk.com/

Table A.2 Architectural CAD software for preliminary and detailed design in 3D

Name	Developer	Website
AutoCAD	Autodesk	http://www.autodesk.com/siteselect.htm
Microstation	Bentley	http://www.bentley.com/en-US/Products/

Table A.3 Traditional general purpose CAD programs

Name	Developer	Website
Autodesk VIZ	Autodesk	http://www.autodesk.com/siteselect.htm
form•Z	auto•des•sys	http://www.formz.com/

Table A.4 General purpose 3D modeling programs

A.2 Structural Engineering Software (cf. section 2.3.2)

Software type

1a Structural model generation using traditional template approach

1b Structural model generation using geometric modeling techniques

1c Structural model generation from architecture

2 Integrated analysis and design

3 Component analysis and design

4 Detailing and drafting

Name	Developer	Website	Type(s)					
			1a	1b	1c	2	3	4
Advance	Graitec	http://www.graitec.com/		x	x			
Allplan/AllPlot	Nemetschek	http://www.nemetschek.com/		x	x	x	x	
Arche, Effel, Melody	Graitec	http://www.graitec.com/				x	x	
Beam 2D	ORAND Systems Inc	http://www.orandsystems.com/					x	
Bocad	GmbH	http://www.bocad.de/						x
Etabs, SAP, etc.	Computers & Structures	http://www.csiberkeley.com/		x		x	x	
Floor2	SCS Inc	http://www.concsoft.com/default.htm					x	
GT-Strudl	Georgia-Tech	http://www.gtstrudl.gatech.edu/	x			x	x	
IdeCAD	ideYAPI Ltd	http://www.idecad.com/			x	x	x	
Bentley Structural	Bentley	http://www.bentley.com/		x	x	x	x	
P-Frame & S-Frame	CSC	http://www.cscworld.com/				x	x	
RAM Structural	RAM International	http://www.ramint.com/index1024.jsp		x		x	x	
Risa	Risa Technologies	http://www.risatech.com/		x		x	x	
Safi	Safi	http://www.safi.com/	x			x	x	
SDS/2	Design Data	http://www.sds2.com/						x
STAAD etc.	Research Engineers Inc.	http://www.reiworld.com/		x		x	x	
Structures	Tekla	http://www.tekla.com/		x		x	x	x

Table A.5 List of structural engineering software applications

A.3 Mechanical and aerospace 3D modeling software

Name	Developer	Website
CATIA	Dassault Systems	http://www.3ds.com/home
Pro/Engineer	PTC the Product Development Company	http://www.ptc.com/
keyCreator	Kubotek Corporation	http://www.kubotekusa.com/
SolidWorks	Dassault Systems	http://www.solidworks.com/
SolidEdge	Unigraphics UGS	http://www.solidedge.com/

Table A.6 List of mechanical and aerospace 3D modeling applications

APPENDIX B

GEOMETRIC MODELING AND REASONING TECHNIQUES

This appendix is divided in two main parts. The first part briefly presents the evolution of geometric modeling applications behind CAD packages from purely graphical drafting tools to complex spatial representations that narrow the semantic gap between software and designers, allow designers to interrogate the model, and help them manipulate their designs consistently. At the end of the first part of this Appendix a comparative of geometric modeling kernels (GMK) is made to help decide which kernel suits the needs of this research project. The second part describes the main mechanisms that enable geometrical reasoning. Such mechanisms are grounded on the different types of relationships between entities in a model and on advanced geometric modeling techniques.

B.1 Graphical models

Over the last three decades a fast evolution of CAD software has taken place, with the aim of providing the most efficient and complete design support focusing mainly on the geometry of the artifact been designed. Graphical models were first developed with the purpose of describing drawings of objects rather than the objects themselves (Mantyla 1988). Early CAD applications incorporated two-dimensional (2-D) drafting commands for drawing low-level entities such as lines, circles, ellipses, etc. Low-level entities could in turn be grouped into more meaningful 2-D entities, such as blocks, bricks, doors,

appliances, etc. Furthermore, blocks could include limited alphanumeric attributes. Manipulation commands were also provided such as move, copy, rotate, fillet, etc; as well as utility tools for better presentation and design specification such as dimensions, hatch, line styles, etc. However, these early CAD applications were meant to support design drafting rather than design itself, i.e. only limited design information could be derived from the graphic model (only distances and areas). For instance, topology was not defined; therefore, after any dimensional design change "apparent" member connectivity was automatically lost. On top of that, since no means for checking design integrity were provided, ambiguous and inconsistent design information could be represented at any time. Furthermore, design changes made in any orthographic projection needed to be performed manually in the other affected projections, thus increasing the risk for producing inconsistent representations.

B.2 Wire-Frame models

Graphical models were later enhanced with the wire-frame representation for three-dimensional objects. Wire-frame models represent objects as collections of edges. The main disadvantages of wire-frame models are (1) the lack of expressiveness and (2) the ambiguity when visualizing wire-frame objects in the screen. (1) Wire-frame models cannot show the faces or surfaces of solids, nor can they be used to compute areas, volumes, or spaces. This is because their geometric data structures are limited to lines, curves, and vertices. (2) Ambiguity comes from the fact that since no boundary faces can be represented, hidden lines or faces cannot be hidden. However, wire-frame models may include topology in their geometric data structures. Therefore, it would be possible to stretch the model and keep its elements (i.e. arcs, lines, and vertices) tied together.

Nowadays, wire-frame representations are used in all CAD systems as auxiliary method for more advanced representations; thus, saving design time and effort when better model visualization or more complete information are not required.

B.3 Geometric models

CAD researchers and developers realized that they needed more precise geometric descriptions of real objects, rather than simple drawings. In aerospace, automobile, and other manufacturing industries, they began exploring mathematical ways for accurately representing complex surfaces (i.e. surface modeling). Soon after, they followed the object-based modeling trend, where an object consists on a non-geometric part and a purely geometric part called the geometric model. A geometric model is, therefore, a subset of the object-based model (Mantyla 1988), which is supported by a solid mathematical foundation. A geometric model is basically composed of two interwoven representations: a geometrical representation describing the algebraic equations of planes, curves and, surfaces that bound the object, and a topological representation describing the connectivity network between entities that make-up the object. It also includes the geometric algorithms for manipulating the object's geometry and maintaining its integrity. The geometry and topology of the model must be mathematically sound in order to perform accurately the tasks required by the designer.

B.4 Solid models

Geometric modeling was specialized into solid modeling in order to make emphasis on the support for the design of solid objects. Solid modeling is the branch of geometric modeling that aims at the construction and processing of "informationally complete"

representations of three-dimensional solid objects (Mantyla 1988). A complete representation of a physical object allows the model to provide answers to arbitrary geometric questions algorithmically such as providing the object's volume, its weight, its centroid, its surface area, and its connectivity. This in turn allows the model to answer more meaningful design questions such as: is the object strong enough to carry this load? Although all solid modelers internally manipulate lower dimensional entities such as curves and surfaces, typically a user cannot access directly such entities, or use them to construct non-solid objects (Rossignac and O'Connor 1990). Solid modelers provide, therefore, higher-level entities (e.g. spheres, cylinders, cubes, extruded solids, etc) to be manipulated by the user for the construction of the solid model. This manipulation is performed with the use of high-level operators.

The traditional spatial representation schemes for solid objects are (Mantyla 1988): (1) primitive instancing, (2) decomposition models, (3) constructive models, (4) boundary models, and (5) hybrid models.

(1) Primitive Instancing: this is the simplest approach for describing solid objects. Using primitive instancing, spatial models are generated by "gluing" together instances of primitive objects, which are taken from a library, in a constrained (non-overlapping way). Examples of primitives are: a cube, a pyramid, a cylinder, or a valid combination (i.e. through attachment by their faces) of them. The instantiation requires a few descriptive parameters, such as component type, location, orientation, and dimensions.

(2) Decomposition models: are not used to build geometric models. Instead, once the model is build, using any other representation, it can be decomposed into non-

overlapping cubes of uniform size and orientation. Therefore, they are auxiliary representations for volume subdivision and volume calculations.

(3) Constructive models: are the most popular representations in use by most CAD systems nowadays. The most widely used constructive representation is called constructive solid geometry (CSG). CSG models are built by combining primitives (the prism, the sphere, the cylinder, the cone, and the torus) using Boolean set operations. However, unlike primitive instancing, the primitives are not "glued" together; intersection, union, substraction, and containment operations are allowed between primitives. Using CSG, solids can also be constructed by extruding closed 2D shapes along a given path. CSG models are represented by a CSG tree, in which the leaves represent primitive solids and the interior nodes either rigid motions or Boolean operations; they are mathematically supported by the theory of point-set topology. CSG models represent the object's boundary implicitly. This means explicit data structures are not provided to describe the boundary of objects. For visualization purposes evaluation algorithms check the boundaries of the primitives and compute the visible surfaces along with their vertices, edges, and curves. This boundary information is displayed but not stored; therefore, the amount of memory is limited to the primitives and the operations in the CSG tree.

(4) Boundary (B-rep) models: represent a solid indirectly through representation of its bounding surfaces. Therefore, B-rep explicitly represent, in a complex data structure, surface geometry and topology of the solid. Integrity of B-rep models is mathematically supported by the theory of algebraic surface topology.

(5) Hybrid solid modelers are capable of supporting several coexisting solid representations. Consistency enforcement is materialized in conversion algorithms that can go from one representation to another (Mantyla 1988). Most modern CAD systems include a primary CSG representation plus a B-rep. CSG has the virtue that it can be converted to any other representation (Mantyla 1988).

Topologically, solid models must be bounded, closed, regular (i.e. with no dangling lower dimensional edges or faces) subsets of E^3 (Euclidean 3-D space). They are formally called manifold solids (3-manifolds). A 3-manifold is a volume that is completely enclosed by a collection of 2D faces (2-manifolds) such that each edge is shared only by two faces and each vertex is the apex of only one cone of faces. This mathematically sound representation permits CAD systems to distinguish the interior, the boundary, and the complement of a solid.

B.5 Advanced Solid models

However, the basic assumption behind solid modeling is that all physical objects are solids (i.e. every real object has a thickness and a volume). Therefore, all entities that can be manipulated within solid modelers have a volume (i.e. they are three-dimensional entities). However, for CAD representations this is a very restrictive fact. For structural analysis purposes, wire-frame representation is the most suitable spatial representation. Therefore, during conceptual design the ultimate goal is to obtain a complete wire-frame representation of the structural system consisting of: zero-dimensional entities (connections), one-dimensional entities (columns and beams), and two-dimensional entities (slabs, shear walls, and retaining walls). In addition, as design evolves, the dimensionality of structural components will change, and most structural elements will

become three-dimensional. Furthermore, if the structure, at any stage of evolution, is to be integrated with the three-dimensional architectural entities, a mixed dimensional model is required. Traditional solid modelers enforce dimensional homogeneity. Therefore, more advanced geometric representations need to be developed that allow lower dimensional entities to be combined with 3D solid entities in a unique building model. Since the early nineties researchers are actively studying advanced spatial representations that support mixed-dimensional geometry and design evolution. Since the intention is still to represent valid "real world" solid objects, these representations are classified within the field of solid modeling.

Informally, non-regular and dimensionally heterogeneous solids are called non-manifold solids. They provide a unified representation of wire-frame, surface and solid materials, which may include irregularities, such as dangling faces and edges, as well as interior boundaries. The ability of a spatial representation to model non-manifold solids brings several additional advantages, it enables CAD applications to: capture internal structures such as cracks and separations, model non-homogeneous solids that are internally composed of various materials, model complex assemblies, etc (Kumar et al. 1999). Examples of non-manifold advanced representations are: the selective geometry complex (SGC) by Rossignac and O'Connor (1990), the constructive non-regularized geometry (CNRG) by Rossignac and Requicha (1991), and the non-manifold solids by Kumar et al. (1999). Informally, SGC is a non-regularized B-rep, enhanced with cellular topology (for modeling objects composed of many adjacent, and possibly overlapping, internal cells, such as the spaces that make up a building), and CNRG is a non-regularized CSG. They extend the theory of B-rep and CSG to model non regularized objects and operations.

However, traditional solid modeling is robust because it relies heavily on the concepts of point-set topology and algebraic topology to ensure validity, consistency, and completeness of the model. For modeling non-regular objects the concepts of point-set topology are still valid but algebraic topology is not strictly applicable anymore. It has to be adapted for the new types of objects and a new algebra is required. Therefore, performing geometric operations may be expensive for large-scale systems.

Traditional CAD systems now incorporate mixed-dimensional representations. However, they do not coexist in the same model. Instead, they are treated as independent non-compatible representations that cannot be mapped to one another (e.g. one cannot perform a direct mapping from a wire frame representation of the structural system to its evolved solid representation). This is because different data structures are needed for solid and wire-frame models, and operators for manipulating and interrogating models with different dimensionalities are specific to each data structure (Sriram et al. 1995).

B.6 Geometric modeling kernels

Every commercially available CAD system has an underlying geometric/solid modeling kernel (GMK), which is the core of the application and includes all the mathematics for supporting spatial design. The power of a GMK comes from the complexity of the objects that it can model. The most widely used GMKs are ACIS (Spatial Corp. 2004), and Parasolid (Unigraphics 2004). While ACIS powers applications like AutoCAD, Form-Z and keyCreator among others, Parasolid powers applications such as Microstation, SolidWorks and SolidEdge. However, other important CAD applications, such as CATIA and Pro/Engineer incorporate their own GMK. Most of these applications are widely used by the mechanical and aerospace industry (see Appendix A).

In terms of modeling power, all commercially available kernels claim that they provide the best modeling features, such as the capability to model all kinds of mathematical curves and surfaces. They also include different spatial representations for modeling solids, such as CSG and B-rep. for visualization purposes wire-frame is also used. However, even though they may visually seem to be working together in a unique model, these representations do not coexist in the same model. One of the latest advances in such kernels (e.g. ACIS and Parasolid) is the inclusion of non-manifold topology. Therefore, theoretically, they can model mixed-dimensional geometry, as well as internal partitions and cracks (i.e. using SGC with cellular topology). However, non-manifold topology is still limited in power and reliability. Other characteristics of most commercial GMKs are the following: they are object-oriented and have been developed in C++, therefore, extensibility is promoted; they provide the possibility for inquiries in geometry and topology, as well as clash detection and minimum distance, mass, volume, and inertia calculations.

In academic research circles several GMKs were proposed over the nineties. Examples include Noodles non-manifold GMK (Gursoz et al. 1991) developed at Carnegie Mellon University (CMU), Genesis boundary modeler (Heisserman 1994) also developed at CMU, GNOMES object-oriented non-manifold modeler (Sriram et al. 1995), which is based on the SGC. All these kernels have been tested on research projects in building applications; Noodles was used by Zamanian et al. (1992) to propose a model for managing spatial and functional building information; Genesis was used by Meyer (1995) to support conceptual structural design; GNOMES was used by Sriram et al. (1995) to model structural elements such as beams and slabs. Furthermore, later versions of

Genesis have been implemented with KBES, and tested by Boeing for supporting generative engineering design (Heisserman et al. 2000).

In addition to modeling power and robustness, interoperability is another aspect of great concern to developers, clients and end-users of GMKs (i.e. how to communicate 3D models between applications). When different applications are powered by the same GMK interoperability is guaranteed without any sort of translation. For example, ACIS-enabled applications use the ACIS Geometry Bus to communicate design, and Parasolid-enabled applications use the Parasolid Data Pipeline. However, when CAD applications do not share the same GMK a bi-directional translation program is required. Fortunately, international standardization efforts have been undergoing over the last fifteen years for the creation of protocols to communicate information between 3D models (ISO-STEP 2004).

B.6.1 Comparative of geometric modeling kernels

Some aspects of geometric modeling and GMKs need to be studied in more detail under the context of the specific needs of conceptual design of building structures. The most important requirement from a GMK for conceptual design is to have the capabilities for supporting design refinement and abstraction by combining multiple geometric representations. This is achieved through a non-manifold GMK that supports handling and combining mixed-dimensional entities. Another concern is related to the support of GMKs for the multidisciplinary aspects of design. Could we combine the structural system and the architecture into a single spatial, non-manifold, representation? Is it practical or convenient to do so? Or is it better to have two separate but properly related representations?

One advantage of having a unique non-manifold representation is shown in Figure B.1. The figure shows the Intersection operation between a 3-D space "A" and a 2-D wall "B"; the result is a 1-D column "C". As explained earlier, this type of operations between mixed-dimensional entities is only possible through a unique non-manifold representation that models spaces, walls, and columns, among others, in a single representation.

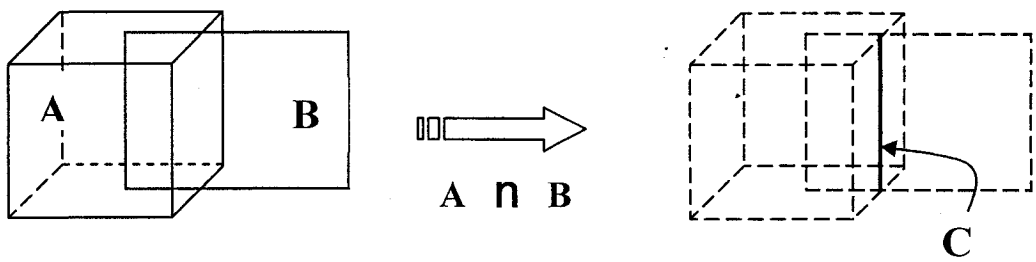


Figure B.1 Example of a non-manifold operation.

Table B.1 presents a comparison of six geometric modeling kernels that were investigated as part of this research project. Reliability and robustness were judged based on the CAD applications supported by those kernels. It is important to note that Gnomes was discarded from the start based on personal communications with its developers. Gnomes' developers argued that the kernel is only apt for helping in the exploration of advanced kernel issues rather than being used as the underlying kernel for a CAD application or prototype.

In table B.1 it is shown that the investigated kernels offer similar advanced solid modeling capabilities. For example, all of them represent mixed-dimensional geometry and provide non-regularized Boolean operations. The history manager keeps track of changes to the model and allows undo and redo options on the model.

Description \ Name and developer	- 1 - ACIS	- 2 - Parasolid	- 3 - Open-Cascade	- 4 - Shapes	- 5 - SMLib	- 6 - Genomes
	Spatial Corp	Unigraphics	Open Cascade S.A.	GeoSmith Consulting	SMS	NIST
General						
Type (licensed, open, proprietary, research)	Licensed	Licensed	Open source	Licensed	Licenced	Research
Development environment	C++ / OO API	C++ / OO API	C++ source code / OO	C++ / OO API	C++ source code / OO	C++ source code / OO
Main areas of support	Mechanical AEC	Mechanical AEC	Mechanical various	Geosciences	Medicine Geosciences various	Mechanical AEC
Reliability and robustness (1: best..... 5)	1	1	2	NA	2	5
Documentation and support (1:best..... 5)	1	2	NA	NA	NA	5
Affordability (1: best.... 5)	2	3	1	4	5	1
Advanced solid modeling						
Mixed-dimensional geometry	yes	yes	yes	yes	yes	yes
Non-regularized Booleans	yes	yes	yes	yes	yes	yes
Section cutting	yes	yes	NA	yes	yes	NA
Tesselation, meshing, triangulation	yes	yes	yes	yes	yes	yes
Pattern manipulation	yes	yes	NA	NA	NA	no
History manager at kernel level	yes	yes	NA	yes	yes	no
Parametric at kernel level	no	no	no	no	no	no

Abbreviations:
OO: object-oriented
API: application programming interface
AEC: architecture, engineering, construction

Table B.1 Comparative of geometric modeling kernels

From table B.1, the main drawback from all kernels investigated is their inability to support parametric design at the kernel level. For obvious reasons, in this research project only geometric modellers offering open-architecture for third-party add-ins were

investigated. At the time of the investigation no geometric modeller was available offering both, open-architecture and parametric capabilities. By contrast, several mechanical and architectural CAD applications were found that incorporated parametric capabilities (For example Pro/ENGINEER® by PTC for mechanical engineering and Revit by Autodesk for architectural design). Those applications rely on proprietary geometric modellers that are conceived exclusively to support specific a commercial application. Therefore, given that no parametric capabilities are provided at kernel level, they have to be provided at a higher level.

Considering all the aspects from Table B.1, ACIS and Parasolid seem to be the best candidates for becoming the underlying geometric modellers for CAD applications. In this research project, ACIS was chosen over Parasolid mainly due to its slightly more affordable price, comprehensive documentation and more effective support.

B.7 Geometrical reasoning

Geometrical reasoning capabilities are supported by implicit and explicit relationships among entities in the design model. Implicit relationships are not "physically" defined using "links" in the design representation but are computed on demand via spatial operations. By contrast, explicit relationships are always specified with "links" in the design representation. As described by Rivard et. al (2000C) the relationships between entities can be categorized in three groups: (1) spatial relationships (e.g. adjacent, overlap and contained), (2) class specific relationships (e.g. inheritance, aggregation and association) and (3) domain-specific relationships (e.g. supports, attached-to and connects). Another group is required for improved geometrical reasoning, namely (4)

parametric relationships, which allow entities to be related mostly through dimensional and geometric constraints. Even though domain-specific relationships are implemented as object-oriented class associations, they are treated separately in order to emphasize their importance geometrical reasoning. The following paragraphs elaborate on the four types of relationships that support geometrical reasoning.

B.7.1 Spatial Relationships

Spatial relationships are defined mathematically through point-set topology. Figure B.2 illustrates the most common types of spatial relationships which are described in the paragraphs that follow.

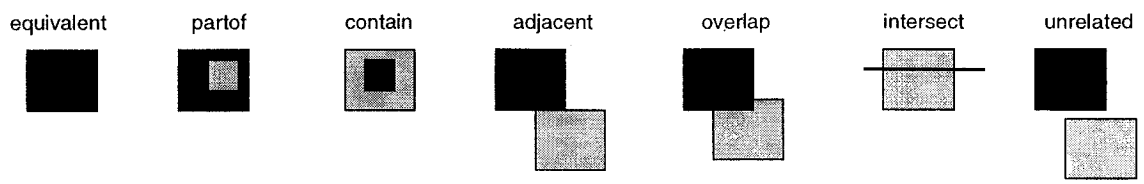


Figure B.2 Spatial relationships between two entities (Zamanian 1992)

- *Equivalent:* the two point-set representations are equivalent and occupy the same space in E^3 (i.e. the euclidean three-dimensional space).
- *Part of:* the first point-set representation is a proper subset of the second subset representation.
- *Contain:* the second point-set representation is a subset of the first point-set representation.
- *Adjacent:* two point-set representations intersect only at their boundaries and not their interiors.

- *Overlap:* the interiors of two point-set representations have the same dimensionality and intersect.
- *Intersect:* the interiors of two point-set representations have different dimensionalities and intersect.
- *Unrelated:* the point-set representations including their boundaries do not intersect.

As stated by Zamanian, the spatial relationships between entities are not explicitly specified in the design representation. Instead, these relationships are computed by an underlying geometric modeler. Some examples of the geometrical reasoning capabilities provided by the above relationships are presented as follows:

- Adjacent: verify a gravity load path to the ground.
- Overlap: verify if beam "B1" and beam "B2" overlap.
- Intersect: find a Structural Element that could support beam "B1"
- Contain: Walls "W1" and "W2" must be coplanar to the plane of the frame where they belong.
- Contain: All beams in a frame assembly must be contained in the plane of the frame.
- Adjacent plus geometric linearity: all columns in a column stack assembly must be collinear.
- Adjacent plus geometric coplanarity: Verify vertical continuity of a group of walls.

B.7.2 Class-specific relationships

While the first group of relationships is derived directly from a geometric modeler, relationships of the second group are explicitly defined in the design representation, but

validated through geometric modeling operations. Examples of the use of these relationships are the following:

- Inheritance: a frame assembly is a special kind of structural assembly.
- Inheritance: a simple-gravity frame is a special kind of frame assembly.
- Aggregation: beam "B1" belongs to the structural assemblies "frame1" and "floor1".
- Aggregation: a frame has beams and columns.

B.7.3 Domain-specific relationships

Domain-specific relationships are also explicitly defined in the designed representation and validated through geometric modeling operations. Examples of domain-specific relationships are the following:

- Beam “B1” is supported by column “C1”.
- Joist "J1" is supported by "B1" and "B2".
- Space “A” is connected to space “B” through a door.

B.7.4 Parametric relationships

Parametric relationships are defined through geometric operations such as: parallelism, coplanarity and linearity, plus dimensional constraints. Examples of parametric relationships are the following:

- A column line must be parallel to wall "W1" and maintain a given distance from it.
- Place wall "W1" symmetrically with respect to wall "W2" about the axis "Y-Y".
- All frames in the Y direction should run parallel and maintain the given structural bay dimensions.

Parametric relationships are usually explicitly defined in the representation; however they may also be used only as reference aids for model construction and therefore become implicit, in which case whenever changes take place in the model these relationships are not preserved. When parametric relationships are explicitly established they are called parametric associations among entities. Thus, when entities are parametrically associated changes can be propagated among them as follows:

- Column line "CL-1" is specified to be parallel to wall "W1" and to preserve the specified distance to it. Then, if the position of the wall is adjusted, the position of the column line is adjusted accordingly.
- Column "C1" is parametrically associated with the cantilever roof that it supports so that if the cantilever size is reduced, the column position adjusts accordingly.

Parametric associations have already been successfully implemented in some advanced CAD packages. They enhance those relationships and introduce a level of sophistication to them; for example, equations can be introduced defining complex relationships among entities in order to preserve some specified geometry or topology. However, since these relationships are explicitly defined within the model, if they are not implemented properly they may overcrowd the model and become a source of pitfalls that will considerable deter the efficiency and reliability of the application. In addition, they interfere with the designer's workflow because they usually require the designer to stop momentarily the design process to explicitly define associations among entities.

APPENDIX C

SYNTHESIS ALGORITHMS

In sections C.1 and C.2 ***Synthesis Algorithms*** are listed. For each algorithm, the action that it performs is explained, as well as the list of arguments it receives and the ones that it returns. The algorithms from the lists whose name is followed by a "*" symbol, are explained in detail using a flowchart or pseudo-code in section C.3.

C.1 Core *Synthesis Algorithms*

	Algorithm	Action/Arguments/Return
1	*verifyWallContinuity**	
	Action	Verifies if a given **`Wall`** or a **`Wall`** segment is continuous down to the ground
	Arguments	A **`Wall`** or a **`Wall`** segment from a given **`Storey`**
	Return	A list of continuous **`Walls`**, a gravity load path (i.e. a geometric planar strip) and a list of **`Stories`**
2	*verifyColumnContinuity*	
	Action	Verifies if a given **`Column`** is continuous down to the ground
	Arguments	A **`Column`** from a given **`Storey`**
	Return	A list of continuous **`Columns`**, a gravity load path (i.e. a geometric linear strip) and a list of **`Stories`**
3	*findSupportsFromArchitecture**	
	Action	Searches within the ground **`Storey`** for **`Walls`** and **`Columns`** that are continuous up to the **`Roof`**
	Arguments	None
	Return	**`Column`** and **`Wall`** strips indicating continuous gravity load paths down to the ground

4	*verifyStructuralLayoutConstraints*	
	Action	Verifies within each **Space** to find the ones that incorporate **Structural Layout Constraints**
	Arguments	None
	Return	List of **Spaces** with associated **Structural Layout Constraints**

Table C.1 *Inspection Algorithms*

	Algorithm	Action/Arguments/Return
1	*wsIntFrame*	
	Action	Integrates (i.e. adds and connects properly) a **Wall Stack** to a coplanar **Frame Assembly**
	Arguments	**Wall Stack** and **Frame Assembly**
	Return	None
2	*csIntFrame*	
	Action	Integrates (i.e. adds and connects properly) a **Column Stack** to a **Frame Assembly**
	Arguments	**Column Stack** and **Frame Assembly**
	Return	None
3	*frameXframe*	
	Action	Intersect two **Frame Assemblies** and produces a **Column Stack**
	Arguments	Two **Frame Assemblies**
	Return	**Column Stack**
4	*frameXfloor**	
	Action	Intersects a **Frame Assembly** and a **Floor Assembly** and generates **Beams**
	Arguments	**Frame Assembly** and **Floor Assembly**
	Return	A list of **Beams**
5	*generateSlabElements* *	
	Action	Generates all the **Slab Elements** for a given **Floor Assembly**
	Arguments	**Floor Assembly**
	Return	List of **Slab Elements**

6	*generateBeamGraph*	
	Action	Generates an adjacency graph connecting all the **Beams** for a given **Floor Assembly**
	Arguments	A "seed" **Beam** from a **Floor Assembly**
	Return	Adjancency graph with connected **Beams**
7	*uniformDepth* *	
	Action	Generates a floor framing layout for a 3-level **Floor Assembly** (i.e. **Slab Element** on top of Secondary **Beam** on top of Primary **Beam**)
	Arguments	**Floor Assembly**
	Return	None

Table C.2 *Configuration Algorithms*

	Algorithm	Action
1	*verifyGravityLoadPaths**	
	Action	Given any **Building Element**, the algorithm verifies if it is supported down to the ground.
	Arguments	**Building Element**
	Return	List of connected **Structural Elements** that provide support
2	*verifyLateralLoadPaths**	
	Action	Checks that each **Floor Assembly** (i.e. diaphragm) is properly supported by at least three lateral resisting **Frame Assemblies** and/or **Wall Stacks**, that are not parallel or intersecting at a single point
	Arguments	None
	Return	True: the structural system is laterally supported False: lateral supports are missing
3	*verifyLengthOfCantilevers*	
	Action	Verifies that the cantilevered length of a given **Beam** should not exceed 1/5 (for concrete) or 1/3 (for steel) of the length of its continuous span for concrete and steel (Eng-tips 2005)
	Arguments	Cantilevered Beam
	Return	True: does not exceed limit False: exceeds limit

Table C.3 *Verification Algorithms*

C.2 Ancillary *Synthesis Algorithms*

	Algorithm	Action
1	*divideArchitecturalWall*	
	Action	Divides an **AWall** to get the geometry of a **SWall**
	Arguments	**AWall** and a list of dividing planes (that could be intersecting **Frame Assemblies**)
	Return	Geometry of **SWall**
2	*findBeamSupports*	
	Action	Searches for supporting primary **Beams** within the **Floor Assembly** or **SWalls** and/or **SColumns** from the **Storey** below
	Arguments	**Beam**
	Return	List of supporting **Structural Elements**
3	*findCollinearNodeConnections*	
	Action	Looks for all the **Node Connections** within certain given alignment
	Arguments	Alignment given by a point and a direction
	Return	List of **Node Connections**
4	*findSlabElementSupports*	
	Action	Searches for supporting **Beams** within the **Floor Assembly** or **SWalls** and/or **SColumns** from the **Storey** below
	Arguments	**Slab Element**
	Return	List of supporting **Structural Elements**
5	*generateAdjacencyGraphPS*	
	Action	Generates an adjacency graph of all the **Primary Spaces** in a **Storey**
	Arguments	**Storey**
	Return	Adjacency graph
6	*getSpaceCentroid*	
	Action	Gets the centroid of a given **Space**
	Arguments	**Space**
	Return	Centroid point

7	*frameContributeVL*	
	Action	Verifies if a **Frame Assembly** contributes to the lateral resistance of the **Vertical Lateral Subsystem** in a given direction. This is done by projecting the geometry of the **Frame Assembly** in the given direction, which should be more than 50% of its length.
	Arguments	Geometry of **Frame Assembly**, **Vertical Lateral Subsystem** direction
	Return	True: contributes False: does not contribute
8	*wsContributeVL*	
	Action	Verifies if a **Wall Stack** contributes to the lateral resistance of the **Vertical Lateral Subsystem** in a given direction. This is done by projecting the geometry of the **Wall Stack** in the given direction, which should be more than 50% of its length.
	Arguments	Geometry of **Wall Stack**, **Vertical Lateral Subsystem** direction
	Return	True: contributes False: does not contribute
9	*sortStories*	
	Action	Sorts a list of **Stories** from the ground to the roof
	Arguments	List of **Stories**
	Return	Sorted list of **Stories**
10	*sortCollinearNCs*	
	Action	Sorts a list of collinear **Node Connections** in one direction
	Arguments	List of **Node Connections**
	Return	Sorted list of **Node Connections**
11	*testInterfere*	
	Action	Test if any two **Representation Entities** interfere
	Arguments	Two **Representation Entities**
	Return	Geometric entity where interference takes place
12	*testCoplanarity*	
	Action	Test if any two planar **Representation Entities** are coplanar
	Arguments	Two **Representation Entities**
	Return	True: complanar False: not complanar

13	*testParallellism*	
	Action	Test if any two planar or linear **Representation Entities** are parallel
	Arguments	Two **Representation Entities**
	Return	True: parallel False: not parallel
14	*verifyBeamSupports*	
	Action	Verifies that a **Beam** has at least two supports or one fixed support.
	Arguments	**Beam**
	Return	True: supported False: unsupported

Table C.4 List of ancillary *Synthesis Algorithms*

C.3 Detailed description of the main synthesis algorithms

C.3.1 Algorithm *verifyWallContinuity*

Figure C.1 presents the algorithm using a flowchart. The algorithm takes as inputs a wall-segment which is a portion of **Wall** "w" that is selected by the engineer for testing vertical continuity, and a list of **Stories** from the **Architectural Model** sorted in ascending order. A wall-segment is a planar face that is a subset of the geometry of a **Wall** which is required because the engineer may want to select only a portion of a **Wall** for structural purposes. However, if the engineer decides to select an entire **Wall** for continuity testing, then the wall-segment is simply the geometry of the selected **Wall**.

The algorithm returns a list of continuous **Walls** and a continuous gravity load-path, which is represented as a wall-strip. If the load paths are narrowed in certain **Stories**, the algorithm also returns a list with the **Stories** where load path narrowing takes place.

As indicated in Figure C.1, the algorithm makes three copies (go, gs and gm) of the selected wall-segment for comparison purposes. Then it translates vertically the copy (gs)

to each of the **Stories** to be investigated for continuity. The algorithm begins the search for continuous **Walls** in the **Stories** below (d = -1) the given **Storey** by descending one **Storey** at a time. It then searches the **Stories** above (d = +1) by ascending one **Storey** at a time. It searches for **Walls** that are coplanar and intersecting the translated copy (gs) of **Wall** "w". If no intersection (gi) is found, then continuity is broken and the algorithm verifies if this took places before reaching the ground. If this is the case, then the returned list of continuous **Walls** is empty and no load-path is returned. If intersecting **Walls** are found, the algorithm compares the intersection (gi) with the original geometry of the selected wall-segment (go). If the intersection is smaller than the geometry of "w" then the Storey under study is stored in a list. If the intersection just found is smaller than the minimum intersection found so far (gm), then the new intersection becomes the smallest one so far and this new intersection (gi) is used for testing (gs) the Stories immediately above or below, depending on the direction of the search. At the end of the search, if the smallest intersection (gm) is smaller than the geometry (go) of Wall "w" then "gm" is copied to all the continuous stories, and through Boolean union a wall-strip is obtained representing a narrowest continuous gravity load-path. Otherwise, if Wall "w" has suffered no size reduction, the load-path is created by copying the geometry of all continuous Walls. The list of Stories where the load-path is narrowed is used by the engineer for checking with the architect if the Wall can be widened in those Stories. If the engineer accepts a load-path, then the computer executes the post-processing algorithm *divideArchitecturalWalls* to get the geometry of **SWalls** from the wall-strip and generate a **Wall Stack**.

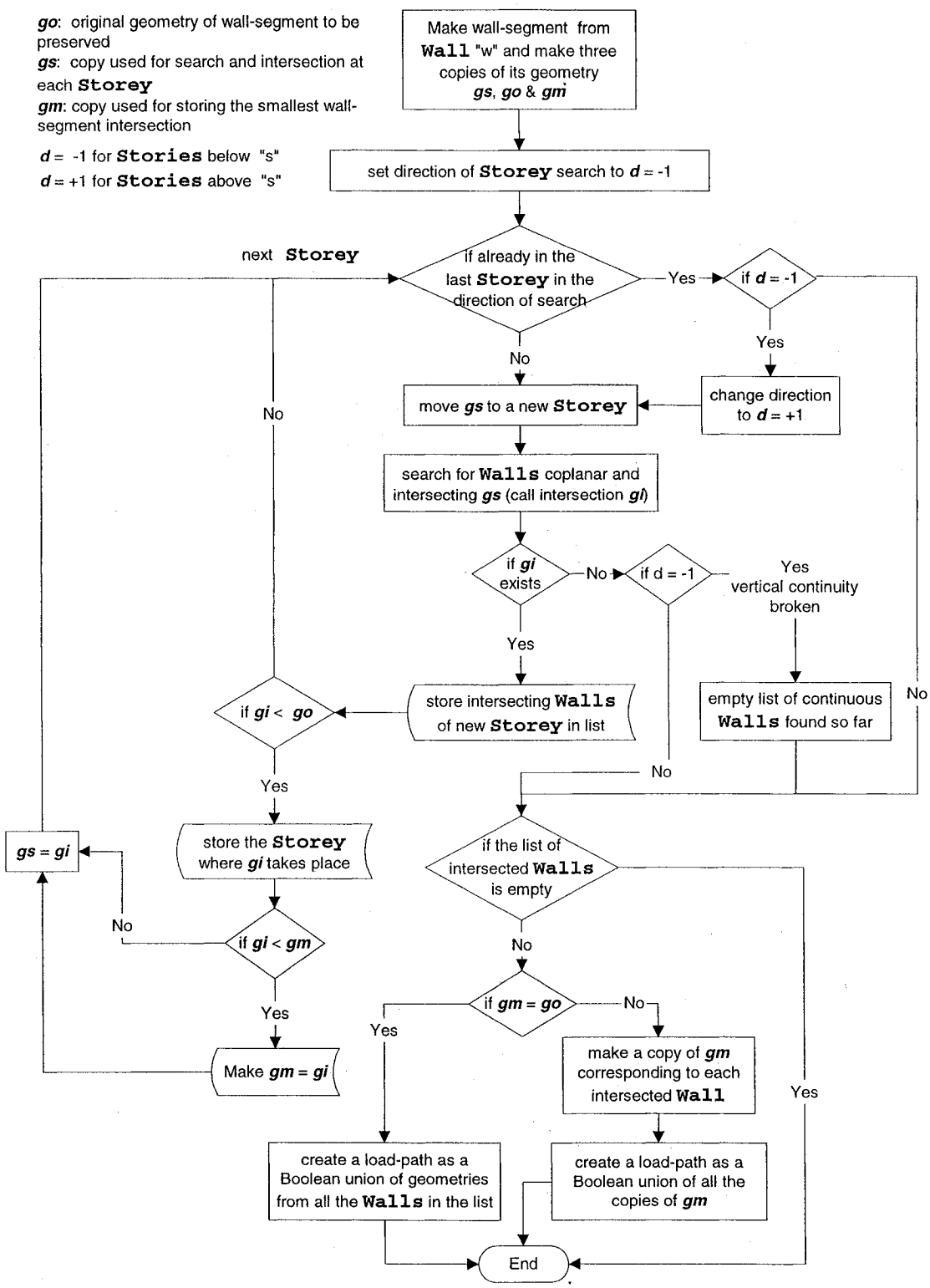


Figure C.1 Flowchart of the algorithm *verifyWallContinuity*

C.3.2 Algorithm *findSupportsFromArchitecture*

Algorithm *findSupportsFromArchitecture* makes use of the algorithm *verifyWallContinuity* (as described in section C.3.1) as well as the algorithm *verifyColumnContinuity* for searching for vertical supports from the **`Architectural Model`**. The algorithm is illustrated using a flowchart in Figure C.2. The search for vertical supports takes place by inspecting each **`Wall`** and each **`Column`** (if placed by the architect) at the lowest **`Storey`**. Considering **`Walls`**, the algorithm checks first if a **`Wall`** being tested for continuity is permanent and opaque, if this is the case, then the algorithm tests its continuity by searching for matches at each consecutive **`Storey`** until the top of the building is reached or the **`Wall`** is interrupted at an intermediate **`Storey`**. A **`Wall`** that is interrupted at an intermediate **`Storey`** is considered continuous up to that upper **`Storey`**. Once the search for **`Walls`** is finished, **`Columns`** placed by the architect are tested for continuity. While inspecting **`Columns`**, the algorithm also asks each **`Column`** whether it is intended to be structural or simply decorative. All the **`Columns`** that are intended (i.e. by the architect) to be structural are tested for continuity by the algorithm. At the end, for each continuous **`Wall`** the algorithm returns a Wall gravity load-path represented by a planar wall-strip, and for each continuous **`Column`** it returns a Column gravity load-path linear column-strip. Then the engineer can select any load-path at will so that **`SWalls`** and **`SColumns`** are generated with their corresponding **`Wall Stacks`** and **`Column Stacks`**. In this case, post-processing algorithm *divideArchitecturalWall* is also executed when applicable for obtaining the geometry of each **`SWall`** when it is a subset of the geometry of a corresponding **`AWall`**.

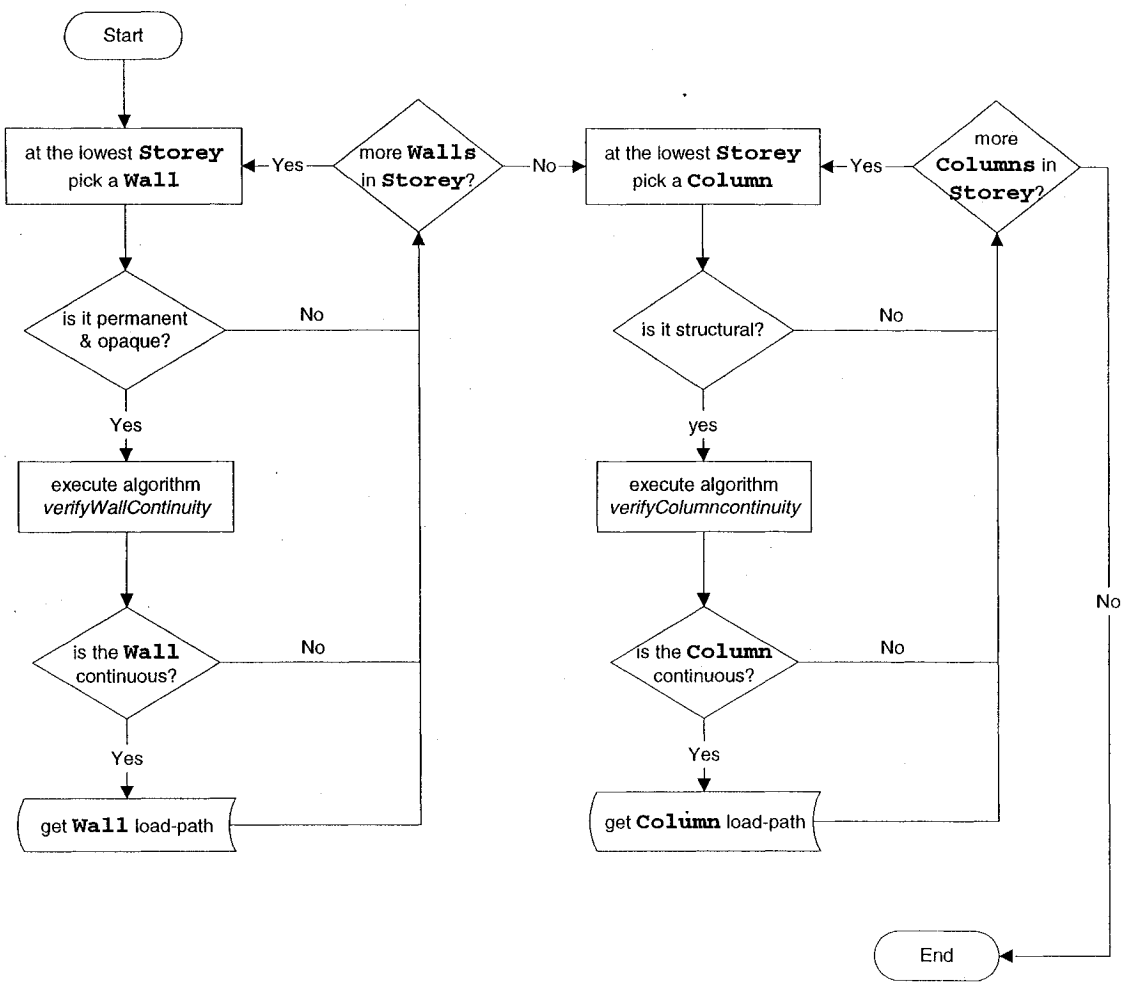


Figure C.2 Flowchart of the algorithm *findSupportsFromArchitecture*

The above inspection algorithms illustrate how the reasoning by the engineer using an **`Architectural Model`** can be supported while looking for gravity load paths to the ground. More powerful algorithms can be devised for assisting the search not only for gravity but also for lateral load paths to the ground in the building architecture. It is envisioned that these algorithms could check the amount and configuration of potential structural **`Walls`** in the floor plan for proper lateral resistance. These algorithms are left for future research.

C.3.3 Algorithm *frameXFloor*

The algorithm *frameXfloor* receives as inputs a **Frame Assembly** "fr" and a **Floor Assembly** "fl", intersects their abstract geometries (i.e. a vertical plane for **Frame Assembly** "fr" and a horizontal plane for **Floor Assembly** "fl") and creates primary **Beams** "b". This algorithm is executed after ***Configuration Algorithms*** *wsIntFrame* (adds **Wall Stacks** to **Frame Assemblies**), *csIntFrame* (adds **Column Stacks** to a **Frame Assemblies**), and *frameXframe* (intersects any two **Frame Assemblies** to get **Columns**). Therefore, before execution, in addition to the abstract geometry the **Frame Assembly** "fr" includes **Column Stacks** and possibly **Wall Stacks** with **Columns** and **Walls** connected through **Node Connections** and **Line Connections**. By contrast, the **Floor Assembly** "fl" incorporates only an abstract geometry. At the end of this algorithm, newly created primary **Beams** are added to both the **Frame Assembly** "fr" and the **Floor Assembly** "fl".

The intersection between the **Frame Assembly** "fr" and the **Floor Assembly** "fl" is a geometric entity, called alignment "a" in the algorithm, which is a collection of collinear intersection lines, having at least one. As indicated in Figure C.3, the algorithm first sorts all Node Connections from Frame Assembly "fr" that lie within alignment "a" and creates a primary Beam, when applicable, between two consecutive Node Connections ("nci" and "ncj"). Once all Beams have been created they are integrated to the intersecting Frame Assembly "f" and Floor Assembly "a". However, before actually creating Beams several conditions need to be verified (lines 5 through 9 in Figure C.3).

```
Given: Frame Assembly "fr" and Floor Assembly "fl"

1:  If Frame Assembly "fr" and Floor Assembly "fl" intersect at alignment "a"

2:  find the Node Connections in Frame Assembly "fr" that lie within alignment "a"
3:  sort the list of Node Connections in the direction of alignment "a"

4:  for each pair of consecutive node connections: "nci" and "ncj"
5:      if there is no Beam already supported at "nci" and "ncj"
6:          if there is no SWall connected to "nci" and "ncj" from below
7:              if there is no opening in Floor Assembly "fl" between "nci" and "ncj"
8:                  if Floor Assembly "fl" is related to a given Structural Zone "sz"
9:                      if Structural Zone "sz" accepts the Beam layout
10:                         create a Beam "b" between "nci" and "ncj"
11:                     else
12:                         if no Construction Line exists aligned with "a"
13:                         create Construction Line "cl" from alignment "a"
14:                         add Construction Line "cl" to list based on direction
15:                 else
16:                     create a Beam "b" between "nci" and "ncj"
17:             else
18:                 if the conflicting MSPS that makes the opening accepts a Beam
19:                     create a Beam "b" between "nci" and "ncj"

20: for each bounding vertex of each intersecting line "l" from alignment "a"
    whose coordinates do not coincide with those of any Node Connection
21:     if no Beam aligned with "a" passes already through such vertex
22:         create a cantilever Beam "b" between the vertex of line "l" and the closest
            node connection in the direction of line "l"

23: for each Beam "b" created
24:     Frame assembly "fr": addStructuralElement( Beam "b" )
25:     Floor assembly "fl": addStructuralElement( Beam "b" )
```

Figure C.3 Pseudo-code of the algorithm *frameXfloor*

The algorithm first checks that there is no Beam in Frame Assembly "fr" or Floor Assembly "fl" that spans between "nci" and "ncj" (line 5). Then it checks that there is no SWall between those Node Connections (line 6) and that there is no opening interfering in the layout of the Beam (line 7). In lines 8 and 9, the algorithm verifies that the Floor Assembly "fl" is constrained by a Structural Zone "sz", in which case it verifies that the

Structural Zone "sz" is not constraining the layout of the intended Beam. If the Beam layout is constrained by "sz", then the algorithm creates a Construction Line (cf. 5.2.2.1 and 5.2.2.5) with alignment "a" and length given by the intersection between "a" and "sz". The algorithm first checks that such Construction Line has not been created already. Construction Lines are used later by ***Configuration Algorithm*** uniform Depth (cf. 5.3.2.2) for generating the layout of the structurally constrained Floor Assembly "fl". If the type of **Structural Layout Constraint** associated with the Structural Zone "sz" is **Column-Free Constraint**, then the algorithm does not create any **Beam** as a result of this process. If the type of constraint is **Alignment Constraint**, then the algorithm creates **Beam**s only following the given alignment. For **Local-Grid Constraint**s all **Beam**s are created. For example, in Figure 5.23 (b) no **Beam**s are created by the algorithm, it creates instead **Construction Line**s in the principal directions of the length of the full **Space**. In Figure 5.23 (c) the only **Beam**s that are created follow the alignment given by the **Line Constraint**. The rest of the area is filled with **Construction Line**s. Therefore, when a **Floor Assembly** is inside or at the top of a constrained **Structural Zone**, algorithm *frameXfloor* may result in: (1) **Beam**s that are incorporated to the corresponding **Floor Assembly**, (2) **Beam**s and **Construction Line**s both passed to **Floor Assembly**, (3) a list of **Construction Line**s which is passed to **Floor Assembly**. **Construction Line**s provide a pattern with tentative locations for primary and secondary Beams.

Since openings in Floor Assemblies are made by Multi-storey Primary Spaces (cf. 5.2.1.1), in Lines 18 and 19 the algorithm asks the conflicting MSPS whether the conflicting Beam is accepted or not. In Lines 20 through 22 the algorithm verifies the

existence of cantilever Beams. The algorithm checks the bounding vertices of each intersection line in order to detect if they do not coincide with any Node Connection (i.e. from Columns or Walls). In such a case, a cantilever Beams is required. However, before creating a cantilever Beam the algorithm checks if there is already an existing (not cantilevered) Beam in such position. Finally, in lines 22, 23 and 24 each primary Beam that has been created is incorporated into the Frame Assembly "fr" and the Floor Assembly "fl".

Typically, when a **Beam** is added to a **Frame Assembly** (line 24), the **Frame Assembly** takes care of connecting it properly to its supports, which may be **Columns** and/or **Walls**. However, in some situations, for example in large open spaces, **Floor Assemblies** are the ones that take care of generating, configuring and supporting **Beams**. In any case, after a **Beam** is added to a **Floor Assembly** (line 25), the **Floor Assembly** verifies if it is properly supported and its length is within the user-specified thresholds. The algorithm *verifyBeamSupports* verifies that a **Beam** has at least two supports or one fixed support (i.e. that restricts rotation about "z" axis) as described in section 5.2.2.2.1, and that its span length is within the thresholds specified by **Floor Assembly** "fl". If a newly incorporated **Beam** is not properly supported, algorithm *findBeamSupports* searches for supporting primary **Beams** within the **Floor Assembly** "fl" or **Columns** and **Walls** within the **Storey** below.

C.3.4 Algorithm *uniformDepth*

This algorithm generates floor framing layouts for **Floor Assemblies** located within or above structurally constrained **Structural Zones**. This is a post-processing algorithm

executed after the *frameXfloor* algorithm generates **Beams** and/or **Construction Lines** depending on the available supports. As indicated by its name, the algorithm is based on the structural principle that long, lightly loaded secondary **Beams** delivering their loads to shorter primary **Beams**, or **Girders**, leads to uniformity of structural depth (Shaeffer 1998).

The algorithm *uniformDepth* is described in Figure C.4 using pseudo-code. **Construction Lines** provide a pattern used by the algorithm for generating primary and secondary **Beams**. **Construction Lines** are given in two lists, one for each orthogonal direction. The algorithm first groups the smallest **Construction Lines** and makes primary **Beams** (lines 3 to 6 in Figure C.4), then it partitions each **Construction Line** intersecting the smallest lines and for each partition it creates secondary **Beams** (lines 7 to 13 in Figure C.4). The newly created **Beams** are incorporated into the **Floor Assembly** "fl" and the corresponding **Frame Assembly** that take care of properly supporting them. Once a **Beam** is created its corresponding **Construction Line** is deleted. Finally, the algorithm recursively calls itself (line 19 in Figure C.4). The recursive call is necessary for the cases when **Alignment Constraints** produce an implicit division of **Spaces** into "sub-spaces" with a resulting division of the **Floor Assembly** at the top. In lines 14 and 15 from Figure C.4, for each **Beam** that is created the algorithm finds a **Frame Assembly** where it belongs. This is done through a containment operation between the geometry of the given **Beam** (which is a line) and the abstract geometry of each **Frame Assembly** (which is a vertical plane). If the geometry of a **Beam** is contained within the abstract geometry of a **Frame Assembly** then the **Beam** belongs to that **Frame Assembly**.

```
    Given: Floor Assembly "fl"

    Note. The Floor assembly "fl" to be framed has two list of Construction Lines:
1:   construction_list1 and construction_list2, one for each orthogonal direction

2:  sort construction_list1 and construction_list2 in ascending order of the construction line length

    begin with the list having the minimum-length Construction Line: e.g. construction_list1
3:  while the length of each construction line in the list is the same as minimum length
4:      create a primary Beam with the geometry of the Construction Line
5:      delete the Construction Line
6:      check next Construction Line in the list

7:  for each Construction Line in construction_list2
8:      using the primary Beams created with construction_list1 partition each Construction Line
        in construction_list2 into smaller Construction Lines
9:      add partitions to construction_list2

10: for each Construction Line in construction_list2
11:     if it does not intersect any Construction Line in construction_list1
12:         use this Construction Line to create secondary Beams
13:         delete its corresponding Construction Line

14: for each Beam that has been created
15:     find its owning Frame Assembly
16:     add Beam to corresponding Frame Assembly
17:     add Beam to Floor assembly "fl"

18: if both lists of Construction Lines are not empty
19:     execute algorithm uniformDepth
```

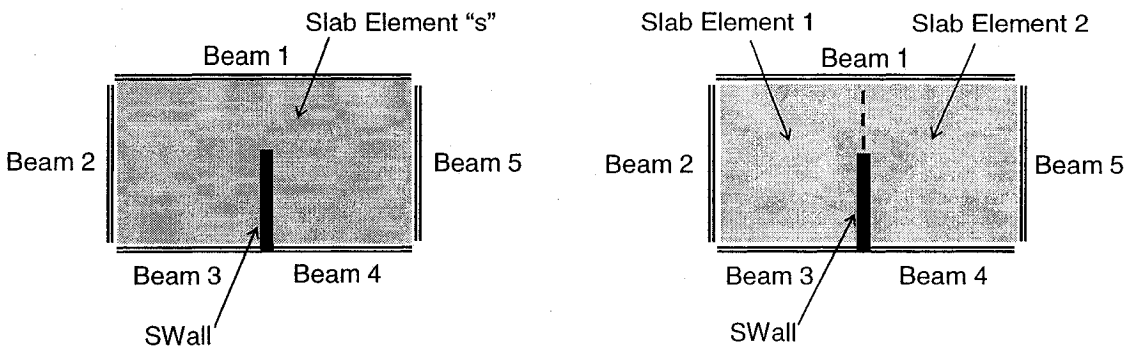
Figure C.4 Pseudo-code of the algorithm *uniformDepth*

C.3.5 Algorithm *generateSlabElements*

The algorithm *generateSlabElements* receives a **Floor Assembly** "fl" as input and creates its **Slab Elements** by partitioning the abstract geometry of the **Floor Assembly** "fl" using the geometry of its primary **Beams** and **SWalls** from the **Storey** below. It uses any initial primary **Beam** "b" from Floor Assembly "fl" as a "seed" to create a wire-frame graph connecting all the primary **Beams** belonging to "fl". The wire-frame graph is then used to slice the abstract geometry (a horizontal plane) of the **Floor Assembly** "fl" and produce **Slab Elements**. Once **Slab Elements** are created from primary **Beams**, the algorithm searches for **SWalls** in the **Storey** below that may intersect some of the newly

created **Slab Elements**. It then uses those **SWalls** to divide the **Slab Elements** that they intersect and create new **Slab Elements**.

The step of slicing the abstract geometry of the **Floor Assembly** "fl" using the wire-frame graph is performed by a function of the underlying geometric modeller ACIS (Spatial Corp. 2004). The step of dividing a **Slab Element** "s" to generate new **Slab Elements** may result in adding new **Beams** to the **Floor Assembly** "fl" as illustrated in Figure C.5. In this case the discontinuous line indicates that a new **Beam** is required for generating the two **Slab Elements**.



(a) **Slab Element** generated by **Beams** (b) Sub-divided by **SWall** underneath

Figure C.5 Generating two **Slab Elements** from one by using a **SWall** below

In Line 3 the recursive algorithm *generateBeamGraph(* **Beam** *"b", "wireFrame")* is called by the algorithm *generateSlabElements.* This function generates the complete wire-frame graph from the geometry of all **Beams** in the **Floor Assembly** "fl" as described in Figure C.6.

```
Given: Beam "b" and "wireFrame"

1:  mark Beam "b" as selected
2:  get the Node Connections of Beam "b"
3:  for each Node Connection "nc" of Beam "b"
4:      for each Beam "c" connected to Beam "b" through each Node Connection "nc"
5:          if Beam "c" has not been selected yet
6:              copy the geometry of Beam "c" and name it "beamGeometry"
7:              perform boolean_union("beamGeometry", "wireFrame")
                Note: the result is stored in wireFrame
8:              execute generateBeamGraph( Beam "c", "wireFrame")
```

Figure C.6 Pseudo-code of recursive algorithm for generating a wire-frame graph from **Beams**

The recursive algorithm *generateBeamGraph* traverses the topological structure of the **Floor Assembly** "fl" and adds, through Boolean union, the geometry of each new **Beam** found to the geometry of the wire-frame graph. Thus, the wire-frame grows as the geometry of each new **Beam** is added.

Once **Slab Elements** are created, the algorithm *verifySlabElementSupports* verifies that each **Slab Element** is properly supported by two **Beams** (that have been verified in advance to be properly supported) or a **Beam** and a **Wall**, or two **Columns**. It a **Slab Element** is not properly supported the algorithm *findSlabElementSupports* searches for its supports. It looks for supports at the perimeter of the **Slab Element**, such as **Beams**, as well as **Walls** and **Columns** from the **Storey** below. It then creates **Line Connections** and/or **Node Connections** joining each **Slab Element** to its supports.

C.3.6 Algorithm *verifyGravityLoadPaths*

The algorithm *verifyGravityLoadPaths* is described using pseudo-code in Figure C.7. It receives as input a **Building Element** "be" (cf. 2.1.1.1.2) such as a **Wall**, a **Column**, a

piece of furniture, or an appliance, and performs a search for supporting **Structural Elements** starting at the **Slab Element(s)** that support (i.e. are in contact with) the **Building Element** directly, and ending at the **Ground**.

```
Given: a Building Element "be"

1:  find Floor Assembly "fl" that supports Building Element "be"
2:  find each Slab Element "sl" in direct contact with the Building Element "be"
3:  for each Slab Element "sl" in contact with the Building Element "be"
4:      verify that "sl" is properly supported using algorithm verifySlabElementSupports
5:      if Slab Element "sl" is properly supported
6::         get its supporting Structural Elements "se"
7:          for each supporting secondary Beam "bs"
8:              verify that "bs" is properly supported using algorithm verifyBeamSupports
9:              if "bs" is properly supported
10:                 continue testing its supporting primary Beams, Columns and/or Walls
11:             else
12:                 return false: load path interrupted at secondary Beam "bs"
13:         for each supporting primary Beam "bp"
14:             verify that "bp" is properly supported using algorithm verifyBeamSupports
15:             if Beam "bp" is properly supported by Walls and/or Columns
16:                 for each supporting SColumn "c"
17:                     verify that "c" is properly supported at the Ground
18:                     if "c" is properly supported
19:                         true: load path found
20:                     else
21:                         false: load path interrupted at Column "c"
22:                 for each supporting SWall "w"
23:                     verify that "w" is properly supported at the Ground
24:                     if "w" is properly supported
25:                         true: load path found
26:                     else
27:                         false: load path interrupted at Wall "w"
28:             else
29:                 false: load path interrupted at primary Beam "bs"
30:     else
31:         false: load path interrupted at Slab Element "sl"
```

Figure C.7 Pseudo-code of the algorithm *verifyGravityLoadPaths*

Each **Structural Element** that is found in the search path is verified for supports. **Slab Element** supports are verified (line 4 in Figure C.7) using algorithm *verifySlabElementSupports* as introduced in section 5.3.2.3, **Beam** supports are verified (line 8 and 14 in Figure C.7) using algorithm *verifyBeamSupports* as described

introduced in section 5.3.2.1. For support verification for **Columns** and **Walls** (lines 11 and 17 in Figure C.7) the algorithm simply traverses their topology down to the **Ground**. If the **Ground** is found, then the **Building Element** "be" is supported, otherwise it is not.

Note that the algorithm simply verifies load paths through low level geometric operations. Tributary areas are not considered and the actual load distribution in the supports is not considered either because it requires the use of engineering first principles governing redundant systems (unless the system is statically determinate). Nevertheless, existing computer applications for structural engineering already deal with gravity load distribution.

C.3.7 Algorithm *verifyLateralLoadPaths*

For lateral load path verification, **Floor Assemblies** are treated as rigid "diaphragms" (i.e. having negligible deformation) that transmit horizontal loads to vertical **Structural Assemblies** that can be **Moment-Resistant Frames**, **Braced Frames** or **Wall Stacks** or a combination of them. Complete lateral load paths to the ground are provided if all building "diaphragms" are supported against movements in any two orthogonal directions and rotations about a vertical axis. These conditions are satisfied if there are at least three vertical **Structural Assemblies** that are not all parallel and do not meet at a single point (Schodek 2004). In agreement with the ***Integrated Representation***, the algorithm assumes all vertical **Structural Assemblies** are always supported at the **Ground**.

The algorithm is described using pseudo-code in Figure C.8. It finds first all vertical **Structural Assemblies** that can resist lateral load and lie or project within any given orthogonal directions (line 1 in Figure C.8). Then it verifies that the number of vertical

Structural Assemblies is greater than three and that they are not parallel and do not meet at a single point (lines 2, 3 and 4 in Figure C.8). If this is the case, then the algorithm verifies that each **Floor assembly** "fl" is connected to the vertical **Structural Assemblies** just found (line 5). If a **Floor Assembly** "fl" is not connected to all the vertical **Structural Assemblies** (line 6), then the algorithm checks that it is connected to at least three not-parallel and not-intersecting vertical **Structural Assemblies** (lines 7, 8 and 9). If this is the case, then the **Floor Assembly** "fl" is supported against lateral loads. Otherwise, the **Floor Assembly** "fl" is not supported for lateral loads.

```
1:   find the vertical lateral-load resisting Structural Assemblies that lie or project in some given
     "x" and "y" directions
2:   if the number of vertical Structural Assemblies is greater than or equal to three
3:       if they are not parallel
4:           if they do not intersect at a single point
5:               for each Floor Assembly in the Building
6:                   if a Floor Assembly "fl" is not connected to all
                     vertical Structural Assemblies through Line Connections
                     (for Wall Stacks) or primary Beams (for Frame Assemblies)
7:                       If the number of vertical Structural Assemblies that support
                         Floor Assembly "fl" is greater than or equal to three
8:                           If they are not parallel
9:                               If they do not intersect at a single point
10:                                  true: the Floor Assembly "fl" is supported
11:                              else
12:                                  false: the Floor Assembly "fl" is not supported
13:                          else
14:                              false: the Floor Assembly "fl" is not supported
15:                      else
16:                          false: the Floor Assembly "fl" is not supported
17:                  else
18:                      true: the Floor Assembly "fl" is supported
19:          else
20:              false: vertical Structural Assemblies meet at a single point
21:      else
22:          false: vertical Structural Assemblies are parallel
23:  else
24:      false: vertical Structural Assemblies are not enough
```

Figure C.8 Pseudo-code of the algorithm *verifyLateralLoadPaths*

The algorithm *verifyLateralLoadPaths* relies exclusively on geometry and topology for verifying lateral load paths to the **Ground**. However, the algorithm does not consider the actual stiffness of vertical **Structural Assemblies** which is a function of their projected length in the direction of the lateral load, as well as the material and cross-sectional dimensions of their constituent **Structural Elements**. In addition, more advanced lateral-load resisting systems demand more complex lateral-load verifications. For example, a "tube" system (or a building core) acts three-dimensionally by resisting lateral loads in one direction through the action of vertical **Structural Assemblies** in both orthogonal directions. Nevertheless, this algorithm performs an initial lateral load path verification that gives a first warning to the engineer and the architect indicating a possible lack of lateral support in any given direction.

APPENDIX D

EXAMPLE OF DRIVER FUNCTION

Notes:

a) The "//" symbol indicate that a comment follows, which is ignored by C++.
b) A line with three dots ". . ." is not C++ code but indicates that many statements similar to the ones above the dotted line follow but were removed only to avoid creating an excessively large appendix.

```
      // ********************************************************************
      // Partial initialization of rectangular site
      // Arguments:
      // name of the Site and its opposite corners (low left and up right)
      // The Site is at z = 0

1.    Site *mySite = new Site("Downtown Montreal", position(0, 0, 0),
      position(200000,100000,0));

      // Arguments: name, site and number of stories.
2.    Building *etsBldg = new Building("ETS", mySite, 7);

      // Argument: Building
3.    ArchitecturalModel *archTest = new ArchitecturalModel(etsBldg);

      // ********************************************************************
      // Architect defines Global Grid for the project at ground level:

4.    position gridOrigin(0,0,0);
5.    unit_vector dir1(1,0,0);
6.    unit_vector dir2(0,1,0);

7.    int module1 = 8000;
8.    int module2 = 8000;

9.    archTest->setGrid(gridOrigin, dir1, dir2, module1, module2);

      // ********************************************************************
      // Architect defines ASlabs (i.e. architectural view of slabs)
      // Arguments: level above ground & thickness both in mm

10.   ASlab *as1 = new ASlab(0,0);
11.   ASlab *as2 = new ASlab(3000,0);
12.   ASlab *as3 = new ASlab(6000,0);
13.   ASlab *as4 = new ASlab(9000,0);
14.   ASlab *pl5 = new ASlab(12000,0);
15.   ASlab *pl6 = new ASlab(15000,0);
```

```
16.   ASlab *pl7 = new ASlab(18000,0);
17.   ASlab *pl8 = new ASlab(21000,0);

      // ************************************************************************
      // Architect creates Storeys
      // Arguments: Storey name,
      //            ASlab acting as the Floor for all Storey Spaces,
      //            ASlab acting as the Ceiling for all Storey Spaces,
      //            Owning Architectural Model

18.   Storey *storey1 = new Storey("SS2", as1, as2, archTest); // 2nd basement
19.   Storey *storey2 = new Storey("SS1", as2, as3, archTest); // 1st basement
20.   Storey *storey3 = new Storey("RC",  as3, as4, archTest); // ground floor
21.   Storey *storey4 = new Storey("P",   as4, as5, archTest); // 1st floor
22.   Storey *storey5 = new Storey("D",   as5, as6, archTest); // 2nd floor
23.   Storey *storey6 = new Storey("T",   as6, as7, archTest); // 3rd floor
24.   Storey *storey7 = new Storey("Q",   as7, as8, archTest); // 4th floor

      // ************************************************************************
      // Architect creates Primary Spaces
      // Arguments: the name of the Primary Space and its owning Storey

      // PS: Sous-sol 2
25.   SingleStoreyPS *ps1 = new SingleStoreyPS("ss2-1", storey1);
26.   SingleStoreyPS *ps2 = new SingleStoreyPS("ss2-2", storey1);
27.   SingleStoreyPS *ps3 = new SingleStoreyPS("ss2-3", storey1);
28.   SingleStoreyPS *ps4 = new SingleStoreyPS("ss2-4", storey1);

      // PS: Sous-sol 1
29.   SingleStoreyPS *ps5 = new SingleStoreyPS("ss1-1", storey2);
30.   SingleStoreyPS *ps6 = new SingleStoreyPS("ss1-2", storey2);
31.   SingleStoreyPS *ps7 = new SingleStoreyPS("ss1-3", storey2);
32.   SingleStoreyPS *ps8 = new SingleStoreyPS("ss1-4", storey2);

      // PS: RC
33.   SingleStoreyPS *ps9 = new SingleStoreyPS("rc-1", storey3);
34.   SingleStoreyPS *ps10 = new SingleStoreyPS("rc-2", storey3);
35.   SingleStoreyPS *ps11 = new SingleStoreyPS("rc-3", storey3);

      . . .

      // PS: Permier
36.   SingleStoreyPS *ps34 = new SingleStoreyPS("p-1", storey4);
37.   SingleStoreyPS *ps35 = new SingleStoreyPS("p-2", storey4);
38.   SingleStoreyPS *ps36 = new SingleStoreyPS("p-3", storey4);

      . . .

      // PS: Deuxieme
39.   SingleStoreyPS *ps55 = new SingleStoreyPS("d-1", storey5);
40.   SingleStoreyPS *ps56 = new SingleStoreyPS("d-2", storey5);
41.   SingleStoreyPS *ps57 = new SingleStoreyPS("d-3", storey5);

      . . .

      // PS: Troisieme
42.   SingleStoreyPS *ps87 = new SingleStoreyPS("t-1", storey6);
```

```
43.   SingleStoreyPS *ps88 = new SingleStoreyPS("t-2", storey6);
44.   SingleStoreyPS *ps89 = new SingleStoreyPS("t-3", storey6);

      . . .

      // PS: Quatrieme
45.   SingleStoreyPS *ps104 = new SingleStoreyPS("q-1", storey7);
46.   SingleStoreyPS *ps106 = new SingleStoreyPS("q-3", storey7);
47.   SingleStoreyPS *ps107 = new SingleStoreyPS("q-4", storey7);

      . . .

      // ********************************************************************
      // Architect creates the geometry of PS for each Storey
      // given the coordinates of each vertex in their perimeter
      // Note that all Spaces are created at Z = 0, however, the program
      // changes the elevation to that of the owning Storey

      // Declare a list of positions (i.e. vertex coordinates)
48.   std::deque<position> poslst;

      // Sous-sol 2:

49.   poslst.push_back(position(89000,42000,0));      // Units in mm
50.   poslst.push_back(position(57000,58000,0));
51.   poslst.push_back(position(59000,62000,0));
52.   poslst.push_back(position(51000,66000,0));
53.   poslst.push_back(position(63000,90000,0));
54.   poslst.push_back(position(99000,72000,0));
55.   poslst.push_back(position(96000,66000,0));
56.   poslst.push_back(position(100000,64000,0));
57.   poslst.push_back(position(94000,52000,0));
58.   poslst.push_back(position(90000,54000,0));
59.   poslst.push_back(position(87000,48000,0));
60.   poslst.push_back(position(91000,46000,0));

61.   ps1->setGeometry(poslst);
62.   poslst.clear();

63.   poslst.push_back(position(129000,22000,0));
64.   poslst.push_back(position(89000,42000,0));
65.   poslst.push_back(position(91000,46000,0));
66.   poslst.push_back(position(99000,42000,0));
67.   poslst.push_back(position(102000,48000,0));
68.   poslst.push_back(position(94000,52000,0));
69.   poslst.push_back(position(103000,70000,0));
70.   poslst.push_back(position(119000,62000,0));
71.   poslst.push_back(position(117000,58000,0));
72.   poslst.push_back(position(125000,54000,0));
73.   poslst.push_back(position(127000,58000,0));
74.   poslst.push_back(position(143000,50000,0));

75.   ps2->setGeometry(poslst);
76.   poslst.clear();

77.   poslst.push_back(position(128000,20000,0));
78.   poslst.push_back(position(73000,20000,0));
```

```
79.   poslst.push_back(position(73000,50000,0));
80.   poslst.push_back(position(129000,22000,0));

81.   ps3->setGeometry(poslst);
82.   poslst.clear();

83.   poslst.push_back(position(73000,26000,0));
84.   poslst.push_back(position(69000,26000,0));
85.   poslst.push_back(position(69000,20000,0));
86.   poslst.push_back(position(35250,20000,0));
87.   poslst.push_back(position(37000,34000,0));
88.   poslst.push_back(position(41000,34000,0));
89.   poslst.push_back(position(41000,42000,0));
90.   poslst.push_back(position(38250,42000,0));
91.   poslst.push_back(position(41000,66000,0));
92.   poslst.push_back(position(73000,50000,0));

93.   ps4->setGeometry(poslst);
94.   poslst.clear();

      . . .

      // Sous-sol 1:

95.   poslst.push_back(position(89000,42000,0));
96.   poslst.push_back(position(57000,58000,0));
97.   poslst.push_back(position(59000,62000,0));
98.   poslst.push_back(position(51000,66000,0));
99.   poslst.push_back(position(63000,90000,0));
100.  poslst.push_back(position(99000,72000,0));
101.  poslst.push_back(position(96000,66000,0));
102.  poslst.push_back(position(100000,64000,0));
103.  poslst.push_back(position(94000,52000,0));
104.  poslst.push_back(position(90000,54000,0));
105.  poslst.push_back(position(87000,48000,0));
106.  poslst.push_back(position(91000,46000,0));

107.  ps5->setGeometry(poslst);
108.  poslst.clear();

      . . .

      // Quatrieme:

      . . .

109.  poslst.push_back(position(40000,58000,0));
110.  poslst.push_back(position(41000,66000,0));
111.  poslst.push_back(position(47000,63000,0));
112.  poslst.push_back(position(44000,58000,0));

113.  ps120->setGeometry(poslst);
114.  poslst.clear();

      // ********************************************************************
      // Architect defines and draws multi-storey Primary Spaces (msps):
      // which aggregate Single Storey PSs (ssps) in a stack
```

```
      // Declare a list of Storeys that the msps spans
115.  std::deque<Storey *> stList;

      // C-1 (Core 1):
      // list of Storeys that the MSPS spans
116.  stList.push_back(storey1);
117.  stList.push_back(storey2);
118.  stList.push_back(storey3);
119.  stList.push_back(storey4);
120.  stList.push_back(storey5);
121.  stList.push_back(storey6);
122.  stList.push_back(storey7);

123.  MultiStoreyPS *msps1 = new MultiStoreyPS("C-1", stList);

124.  poslst.push_back(position(99000,42000,0));
125.  poslst.push_back(position(87000,48000,0));
126.  poslst.push_back(position(90000,54000,0));
127.  poslst.push_back(position(102000,48000,0));

128.  msps1->setGeometry(poslst);

129.  stList.clear();
130.  poslst.clear();

      // C-2 (Core 2):
      // list of Storeys that the MSPS spans
131.  stList.push_back(storey1);
132.  stList.push_back(storey2);
133.  stList.push_back(storey3);
134.  stList.push_back(storey4);
135.  stList.push_back(storey5);
136.  stList.push_back(storey6);
137.  stList.push_back(storey7);

138.  MultiStoreyPS *msps2 = new MultiStoreyPS("C-2", stList);

139.  poslst.push_back(position(49000,62000,0));
140.  poslst.push_back(position(57000,58000,0));
141.  poslst.push_back(position(59000,62000,0));
142.  poslst.push_back(position(51000,66000,0));

143.  msps2->setGeometry(poslst);

144.  stList.clear();

145.  poslst.clear();

      . . .

      // Omni sports:
      // list of Storeys that the MSPS spans
146.  stList.push_back(storey6);
147.  stList.push_back(storey7);

148.  MultiStoreyPS *msps7 = new MultiStoreyPS("Omni", stList);
```

```
149.  poslst.push_back(position(83000,50000,0));
150.  poslst.push_back(position(51000,66000,0));
151.  poslst.push_back(position(63000,90000,0));
152.  poslst.push_back(position(95000,74000,0));

153.  msps7->setGeometry(poslst);

154.  stList.clear();
155.  poslst.clear();

      // Gym:
      // list of Storeys that the MSPS spans
156.  stList.push_back(storey6);
157.  stList.push_back(storey7);

158.  MultiStoreyPS *msps8 = new MultiStoreyPS("Gym", stList);

159.  poslst.push_back(position(95000,74000,0));
160.  poslst.push_back(position(89000,62000,0));
161.  poslst.push_back(position(113000,50000,0));
162.  poslst.push_back(position(119000,62000,0));

163.  msps8->setGeometry(poslst);

164.  stList.clear();
165.  poslst.clear();

      // ************************************************************************
      // The architect creates a Structural Layout Constraint (SLC) for MSPS:

      // Column-Free Layout Constraint (CF_SLC):
166.  CF_slc *cfc1 = new CF_slc(msps7);
167.  CF_slc *cfc2 = new CF_slc(msps8);

      // ************************************************************************
      // The Architect initializes and draws walls in each Storey:

168.  const int lowCorners = 2;
169.  position ptsw[lowCorners];
170.  int st2Lev = storey2->getLevel();
171.  int st3Lev = storey3->getLevel();
172.  int st4Lev = storey4->getLevel();
173.  int st5Lev = storey5->getLevel();
174.  int st7Lev = storey7->getLevel();

      // Draw Walls in Rez-de-chaussee (storey3):
175.  int wallType = 1;        // Curtain wall
176.  ptsw[0] = position(126000,36000,st3Lev);
177.  ptsw[1] = position(132000,48000,st3Lev);
178.  AWall *w1 = new AWall(ptsw, wallType, storey3);

179.  ptsw[0] = position(132000,48000,st3Lev);
180.  ptsw[1] = position(127000,58000,st3Lev);
181.  AWall *w2 = new AWall(ptsw, wallType, storey3);

      . . .
```

```
182.  // Draw Walls in Quatrieme etage (storey 7):
183.  wallType = 1;            // Curtain wall
184.  ptsw[0] = position(126000,36000,st7Lev);
185.  ptsw[1] = position(132000,48000,st7Lev);
186.  AWall *w45 = new AWall(ptsw, wallType, storey7);

      . . .

187.  ptsw[0] = position(115000,34000,st7Lev);
188.  ptsw[1] = position(126000,36000,st7Lev);
189.  AWall *w55 = new AWall(ptsw, wallType, storey7);

      // Draw (retaining) Walls in Sous-sol 2 (storey 1):
190.  wallType = 0;            // Interior wall
191.  ptsw[0] = position(129000,22000,st1Lev);
192.  ptsw[1] = position(143000,50000,st1Lev);
193.  AWall *w56 = new AWall(ptsw, wallType, storey1);
194.  w56->setName("rw1");

195.  ptsw[0] = position(143000,50000,st1Lev);
196.  ptsw[1] = position(63000,90000,st1Lev);
197.  AWall *w57 = new AWall(ptsw, wallType, storey1);
198.  w57->setName("rw2");

199.  ptsw[0] = position(63000,90000,st1Lev);
200.  ptsw[1] = position(49000,62000,st1Lev);
201.  AWall *w58 = new AWall(ptsw, wallType, storey1);
202.  w58->setName("rw3");

      . . .

      // Core Walls Sous-sol 2:
203.  wallType = 0;            // Interior wall
204.  ptsw[0] = position(125000,54000,st1Lev);
205.  ptsw[1] = position(127000,58000,st1Lev);
206.  AWall *cw1 = new AWall(ptsw, wallType, storey1);
207.  cw1->setName("cw1");

208.  ptsw[0] = position(127000,58000,st1Lev);
209.  ptsw[1] = position(119000,62000,st1Lev);
210.  AWall *cw2 = new AWall(ptsw, wallType, storey1);
211.  cw2->setName("cw2");

      . . .

      // ********************************************************************
      // The Architect creates Secondary Spaces by grouping Primary Spaces
      // A selection trick is used for selecting Spaces since the Building
      // is skewed. The centroid of each Space is compared with an
      // imaginary line that divides spaces to be selected.
      // In the future a GUI will make this selection easier.

212.  std::deque<PrimarySpace *> psList;

      // Number of PSs per Storey:
213.  int noPsStorey1 = storey1->getNoPSs();
```

```
214.  int noPsStorey2 = storey2->getNoPSs();
215.  int noPsStorey3 = storey3->getNoPSs();
216.  int noPsStorey4 = storey4->getNoPSs();
217.  int noPsStorey5 = storey5->getNoPSs();
218.  int noPsStorey6 = storey6->getNoPSs();
219.  int noPsStorey7 = storey7->getNoPSs();

220.  SingleStoreyPS *psi;
221.  BODY *psGeom, *abGeom, *cdGeom;
222.  position ap(13000,80000,0);
223.  position bp(151000,11000,0);
224.  position centroid, cp, dp, ip;

      // Building North:
      // Select Spaces from the first Storey:
225.  for (int ss = 1; ss <= noPsStorey1; ss++)
226.  {
227.  psi = storey1->getPs(ss);
228.  psGeom = psi->getGeometry();
229.  centroid = archTest->getGre()->getCentroid(psGeom);

230.  cp.x() = centroid.x();
231.  cp.y() = centroid.y() + 100000; cp.z() = centroid.z();
232.  dp.x() = centroid.x();
233.  dp.y() = centroid.y() - 100000; dp.z() = centroid.z();

234.  ap.z() = centroid.z();
235.  bp.z() = centroid.z();

236.  abGeom = archTest->getGre()->setGbGeometry(ap, bp);
237.  cdGeom = archTest->getGre()->setGbGeometry(cp, dp);

238.  ip = archTest->getGre()->intersectWires(abGeom, cdGeom);

239.  if(centroid.y() > ip.y())
240.  psList.push_back(psi);
241.  }

      // Select Spaces from the second Storey:
242.  for (ss = 1; ss <= noPsStorey2; ss++)
243.  {
244.  psi = storey2->getPs(ss);
245.  psGeom = psi->getGeometry();
246.  centroid = archTest->getGre()->getCentroid(psGeom);

247.  cp.x() = centroid.x();
248.  cp.y() = centroid.y() + 100000; cp.z() = centroid.z();
249.  dp.x() = centroid.x();
250.  dp.y() = centroid.y() - 100000; dp.z() = centroid.z();

251.  ap.z() = centroid.z();
252.  bp.z() = centroid.z();

253.  abGeom = archTest->getGre()->setGbGeometry(ap, bp);
254.  cdGeom = archTest->getGre()->setGbGeometry(cp, dp);

255.  ip = archTest->getGre()->intersectWires(abGeom, cdGeom);
```

```
256.  if(centroid.y() > ip.y())
257.  psList.push_back(psi);
258.  }

      . . .

259.  SecondarySpace *BldgNorth = new SecondarySpace(psList);

260.  psList.clear();

      // Building South:
      // Select Spaces from the first Storey:
261.  for (ss = 1; ss <= noPsStorey1; ss++)
262.  {
263.  psi = storey1->getPs(ss);
264.  psGeom = psi->getGeometry();
265.  centroid = archTest->getGre()->getCentroid(psGeom);

266.  cp.x() = centroid.x();
267.  cp.y() = centroid.y() + 100000; cp.z() = centroid.z();
268.  dp.x() = centroid.x();
269.  dp.y() = centroid.y() - 100000; dp.z() = centroid.z();

270.  ap.z() = centroid.z();
271.  bp.z() = centroid.z();

272.  abGeom = archTest->getGre()->setGbGeometry(ap, bp);
273.  cdGeom = archTest->getGre()->setGbGeometry(cp, dp);

274.  ip = archTest->getGre()->intersectWires(abGeom, cdGeom);

275.  if(centroid.y() < ip.y())
276.  psList.push_back(psi);
277.  }

      // Select Spaces from the second Storey
278.  for (ss = 1; ss <= noPsStorey2; ss++)
279.  {
280.  psi = storey2->getPs(ss);
281.  psGeom = psi->getGeometry();
282.  centroid = archTest->getGre()->getCentroid(psGeom);

283.  cp.x() = centroid.x();
284.  cp.y() = centroid.y() + 100000; cp.z() = centroid.z();
285.  dp.x() = centroid.x();
286.  dp.y() = centroid.y() - 100000; dp.z() = centroid.z();

287.  ap.z() = centroid.z();
288.  bp.z() = centroid.z();

289.  abGeom = archTest->getGre()->setGbGeometry(ap, bp);
290.  cdGeom = archTest->getGre()->setGbGeometry(cp, dp);

291.  ip = archTest->getGre()->intersectWires(abGeom, cdGeom);

292.  if(centroid.y() < ip.y())
```

```
293.  psList.push_back(psi);
294.  }

      . . .

295.  SecondarySpace *BldgSouth = new SecondarySpace(psList);

296.  psList.clear();

      // ************************************************************************
      // Structural engineer initializes Structural System associated with the
      // Building

297.  StructuralSystem *struTest = new StructuralSystem(etsBldg);


      // ************************************************************************
      // Engineer defines ISVs from Secondary Spaces

      // Arguments: structural system and associated SS
298.  Isv *isv1 = new Isv(struTest, BldgNorth);
299.  Isv *isv2 = new Isv(struTest, BldgSouth);

      // ************************************************************************
      // Engineer defines four structural subsystems: HG, VG, VLX, VLY
      // Note. CSA: Column Stack, WSA: Wall Stack,
      //       B1FA: Slab on Beam Floor Assembly, MRF: Moment Resistant Frame

      // ************ ISV1 (Building North):

      // Horizontal gravity subsystem:
300.  std::deque<char *> hg1List;
301.  char *myHg = "B1FA";           // Type of Floor Assembly
302.  hg1List.push_back(myHg);
303.  isv1->setSubSysHG(hg1List);

      // Vertical gravity subsystem:
304.  std::deque<char *> vg1List;
305.  char *myVg = "CSA";            // Type of vertical support
306.  vg1List.push_back(myVg);
307.  isv1->setSubSysVG(vg1List);

      // Vertical lateral subsystem in 1st typically "x" direction:
308.  int xPos = 2;
309.  int yPos = -1;

310.  std::deque<char *> vl11List;
311.  char *vl1a = "MRF";            // One type of lateral support
312.  vl11List.push_back(vl1a);
313.  char *vl1b = "WSA";            // Another type of lateral support
314.  vl11List.push_back(vl1b);
315.  isv1->setSubSysVL1(xPos, yPos, vl11List);

      // Vertical lateral subsystem in 2nd typically "y" direction:
316.  xPos = 1;
317.  yPos = 2;
```

```
318.  std::deque<char *> vl21List;
319.  char *vl2a = "MRF";           // One type of lateral support
320.  vl21List.push_back(vl2a);
321.  char *vl2b = "WSA";           // Another type of lateral support
322.  vl21List.push_back(vl2b);
323.  isv1->setSubSysVL2(xPos, yPos, vl21List);

      // ************* ISV2 (Building South):

      // Horizontal gravity subsystem:
324.  std::deque<char *> hg2List;
325.  myHg = "B1FA";
326.  hg2List.push_back(myHg);
327.  isv2->setSubSysHG(hg2List);

      // Vertical gravity subsystem:
328.  std::deque<char *> vg2List;
329.  myVg = "CSA";
330.  vg2List.push_back(myVg);
331.  isv2->setSubSysVG(vg2List);

      // Vertical lateral subsystem in 1st direction (typically "x" direction):
332.  xPos = 1;
333.  yPos = 0;

334.  std::deque<char *> vl12List;
335.  vl1a = "MRF";
336.  vl12List.push_back(vl1a);
337.  vl1b = "WSA";
338.  vl12List.push_back(vl1b);
339.  isv2->setSubSysVL1(xPos, yPos, vl12List);

      // Vertical lateral subsystem in 2nd direction (typically "y" direction):

340.  xPos = 0;
341.  yPos = 1;

342.  std::deque<char *> vl22List;
343.  vl2a = "MRF";
344.  VL2 subsystem
345.  vl22List.push_back(vl2a);
346.  vl2b = "WSA";
347.  VL2 subsystem
348.  vl22List.push_back(vl2b);
349.  isv2->setSubSysVL2(xPos, yPos, vl22List);

      // ********************************************************************
      // Define SZs by grouping PSs and SSs.

      // Define First Structural Zone (SZ1):
350.  std::deque<Space *> spaceList1;
351.  spaceList1.push_back(msps7);        // Groups only one Space
352.  char *myname = "Omni Sports";
353.  StructuralZone *sz1 = new StructuralZone(spaceList1, isv1, myname);

      // Define Second Structural Zone (SZ2):
354.  std::deque<Space *> spaceList2;
```

```
355.  spaceList2.push_back(msps8);
356.  myname = "Gym";                          // Groups only one Space
357.  StructuralZone *sz2 = new StructuralZone(spaceList2, isv1, myname);

      // Need to define a 3rd SZ (SZ3).
      // This is necessary because the last slab is divided
      // into three slabs as required by the SZs (Gym and Omni-Sports)
      // This last SZ groups all remaining PSs in the last Storey:

358.  std::deque<Space *> spaceList3;
359.  for (ss = 1; ss <= noPsStorey7; ss++)
360.  {
361.  psi = storey7->getPs(ss);
362.  psGeom = psi->getGeometry();
363.  centroid = archTest->getGre()->getCentroid(psGeom);
364.  cp.x() = centroid.x();
365.  cp.y() = centroid.y() + 100000; cp.z() = centroid.z();
366.  dp.x() = centroid.x();
367.  dp.y() = centroid.y() - 100000; dp.z() = centroid.z();

368.  ap.z() = centroid.z();
369.  bp.z() = centroid.z();

370.  abGeom = archTest->getGre()->setGbGeometry(ap, bp);

371.  cdGeom = archTest->getGre()->setGbGeometry(cp, dp);

372.  ip = archTest->getGre()->intersectWires(abGeom, cdGeom);

373.  if(centroid.y() > ip.y())
374.  {
375.  if(psi->getPartOfMsPs())
376.  {
377.  if(psi->getMsps()->getName() != "Omni" &&
378.  psi->getMsps()->getName() != "Gym")
379.  spaceList3.push_back(psi);
380.  }
381.  else
382.  spaceList3.push_back(psi);
383.  }
384.  }

385.  myname = "Quatrieme";
386.  StructuralZone *sz3 = new StructuralZone(spaceList3, isv1, myname);

      // ************************************************************************
      // Engineer creates Moment Resistant Frames (MRF)

387.  char *frameType;
388.  char *frameName;

      // ISV1 (North Building):
389.  frameType = "MRF";
390.  frameName = "frx11";

391.  int xPos1, yPos1, xPos2, yPos2;
```

```
      // frx11:
392.  xPos1 = 49000;
393.  yPos1 = 62000;
394.  xPos2 = 129000;
395.  yPos2 = 22000;

396.  isv1->setFrame2D(frameType, frameName, xPos1, yPos1, xPos2, yPos2);

      // frx12:
397.  frameType = "MRF";
398.  frameName = "frx12";

399.  xPos1 = 51000;
400.  yPos1 = 66000;
401.  xPos2 = 131000;
402.  yPos2 = 26000;

403.  isv1->setFrame2D(frameType, frameName, xPos1, yPos1, xPos2, yPos2);

      . . .

      // fry11:
404.  frameType = "MRF";
405.  frameName = "fry11";

406.  xPos1 = 49000;
407.  yPos1 = 62000;
408.  xPos2 = 63000;
409.  yPos2 = 90000;

410.  isv1->setFrame2D(frameType, frameName, xPos1, yPos1, xPos2, yPos2);

      . . .


      // ISV2 (South Building):
411.  frameType = "MRF";
412.  frameName = "frx21";

      // frx21:
413.  xPos1 = 35250;
414.  yPos1 = 20000;
415.  xPos2 = 128000;
416.  yPos2 = 20000;

417.  isv2->setFrame2D(frameType, frameName, xPos1, yPos1, xPos2, yPos2);

      . . .

      // ***********************************************************************
      // Engineer creates a Floor SA (B1FA)

      // For FloorSAs that are not constrained by any SZ:
418.  StructuralZone *szPtr = 0;

419.  char *flType;
char *flName;
```

```
      // ************ ISV1 (Building North):

      // First Floor Slab
420.  flType = "B1FA";
421.  flName = "fl11";

422.  isv1->setFloorSA(flType, flName, pl2, szPtr);

      // Second Floor Slab
423.  flType = "B1FA";
424.  flName = "fl12";

425.  isv1->setFloorSA(flType, flName, pl3, szPtr);

      . . .

      // Seventh Floor Slab
426.  flType = "B1FA";
427.  flName = "fl17a";

428.  isv1->setFloorSA(flType, flName, pl8, sz3);

      // Seventh Floor Slab
429.  flType = "BGFA";
430.  flName = "fl17b";

431.  isv1->setFloorSA(flType, flName, pl8, sz1);

      // Seventh Floor Slab
432.  flType = "BGFA";
433.  flName = "fl17c";

434.  isv1->setFloorSA(flType, flName, pl8, sz2);


      // ************ ISV2 (Building South):

      // First Floor Slab
435.  flType = "B1FA";
436.  flName = "fl21";

437.  isv2->setFloorSA(flType, flName, pl2, szPtr);

      . . .

      // ***************************************************************************
      // Assign specific floor type to each Floor Assembly:

438.  StructuralAssembly *saPtr;
439.  FloorSA *fsaPtr;

      // ISV1:
440.  saPtr = isv1->getHGS()->getSAName("fl11");
441.  fsaPtr = static_cast<FloorSA *>(saPtr);
442.  SteelDeck *sd1Ptr = new SteelDeck(fsaPtr);
```

```
    . . .
443.  saPtr = isv1->getHGS()->getSAName("fl17b");
444.  fsaPtr = static_cast<FloorSA *>(saPtr);
445.  OpenWeb *ow1Ptr = new OpenWeb(fsaPtr);

446.  saPtr = isv1->getHGS()->getSAName("fl17c");
447.  fsaPtr = static_cast<FloorSA *>(saPtr);
448.  OpenWeb *ow2Ptr = new OpenWeb(fsaPtr);

    . . .

449.  saPtr = isv2->getHGS()->getSAName("fl26");
450.  fsaPtr = static_cast<FloorSA *>(saPtr);
451.  SteelDeck *sd13Ptr = new SteelDeck(fsaPtr);

452.  saPtr = isv2->getHGS()->getSAName("fl27");
453.  fsaPtr = static_cast<FloorSA *>(saPtr);
454.  SteelDeck *sd14Ptr = new SteelDeck(fsaPtr);

    // ****************************************************************
    // Engineer verifies AWall continuity within the architectural model
    // Arguments: the name of the AWall,
    // and the Storey where the AWall belongs.

455.  char *lowStorey;
456.  char *highStorey;

457.  archTest->verifyWallContinuity("cw1", "SS2", lowStorey, highStorey);

    . . .

    // ****************************************************************************
    // Make Wall Stacks (WSA):
    // First a list of continuous Walls is created
    // Next, the list is passed to the new WSA along with the WSA name

458.  std::list<AWall *> awList;

    // BUILDING NORTH:

459.  archTest->getContinuousWalls("cw1", "SS2", awList);
460.  char *wsaName = "wsa1";
461.  isv1->setWSA(awList, wsaName);
462.  awList.clear();

463.  archTest->getContinuousWalls("cw2", "SS2", awList);
464.  wsaName = "wsa2";
465.  isv1->setWSA(awList, wsaName);
466.  awList.clear();

    . . .

    // BUILDING SOUTH:

467.  archTest->getContinuousWalls("cw19", "SS2", awList);
468.  wsaName = "wsa19";
```

```
469.  isv2->setWSA(awList, wsaName);
470.  awList.clear();

      . . .

471.  archTest->getContinuousWalls("cw26", "SS2", awList);
472.  wsaName = "wsa26";
473.  isv2->setWSA(awList, wsaName);
474.  awList.clear();
```