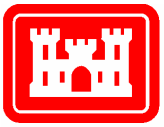
ERDC/CERL TR-05



US Army Corps
of Engineers®
Engineer Research and
Development Center

n-Dimensional Abstraction Patterns in NP-Hard Design Space

A Parametric Methodology for Spatial (Re)Configuration

Van J. Woods

Sept 2005

Construction Engineering
Research Laboratory

ERDC/CERL TR-05
September 2005

n-Dimensional Abstraction Patterns in NP-Hard Design Space

A Parametric Methodology for Spatial (Re)Configuration

Van J. Woods

Construction Engineering Research Laboratory
PO Box 9005
Champaign, IL 61826-9005

Final Report

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

ABSTRACT: Current state of the art research has demonstrated the applicability of parametric shape grammar, optimization, and constraint-based interactive techniques for the space layout problem. However, they fall short in the following primary areas: performance, flexibility, and control. The basic premise of this work is based on the principle that a hierarchical pattern-based design method will provide a more robust approach.

Contents

List of Figures and Tables ..... **v**

Conversion Factors ..... **vi**

Preface ..... **vii**

1 Introduction **1**

Introduction 1

Army Relevance 2

2 Background **3**

Problem Definition 3

Space Layout Examples 4

Research Approaches 6

Optimization *6*

Shape Grammars *9*

Case Based Reasoning *11*

Parametric Modeling *11*

Current Approaches in Practice 12

Traditional CAD *12*

BIM *13*

Characteristics of Design 14

Summary 16

3 Pattern Based Approach **17**

Definition of n-Dimensional Abstraction Patterns 17

Pattern language 17

Primary Syntax *18*

Interactive Configuration Exploration (ICE) *20*

Interaction Object Graphs (IOG) *23*

Process Architecture 25

Detailed Example 27

RectanglePattern 27

Explanation of Detailed Example 29

Definition *29*

Interaction *29*

Summary 31

4 Pattern Catalog 33

Explanation 33
Container 34
LinearSequence 38
DoubleLoadedCorridor 41
TConfig 45
LConfig.DLC2SLC 49

5 Pattern Analysis 52

Introduction 52
Patterns Discussed 52
BooleanContainer 52
LConfig.DLC2DLC 55
UConfig 56
RacetrackCorridor 57
Circulation Patterns 58
Pattern Interaction 59

6 Conclusion 61

Summary 61
Future Research 62

7 APPENDIX: Building Catalog 64

8 APPENDIX: QFB 91

QFB Encoder-Analyzer 91
Source Code for QFB Output File Generator 91

9 APPENDIX: Rectangle Code 94

Description 94

10 Bibliography 111

REPORT DOCUMENTATION PAGE 114

List of Figures and Tables

Figures

Figure 1: Example of space boundaries and their segmentation. 3

Figure 2: A sketch from a sustainability design charrette. The left image is the original plan, and the right image is the desired plan. 4

Figure 3: Three different layouts that are topologically identical and geometrically unique. 8

Figure 4: Syntax and semantics of the Interactive Configuration Exploration (ICE) notation (Moustapha, 2004). 23

Figure 5: Process architecture. 25

Figure 6: User dialog box automatically generated from Relationships variables. 30

Figure 7: BooleanContainer examples. 53

Figure 8: LConfig.DLLC2DLC examples. 55

Figure 9: UConfig pattern examples. 56

Figure 10: Ft. Lewis FY06 Barracks module configuration as instantiation of UConfig. 56

Figure 11: Racetrack corridor pattern examples 57

Figure 12: Strategies for circular dependencies. 57

Figure 13: Circulation pattern topologies by number of segments 58

Figure 14: System architecture diagram of Cube virtual reality environment and CAD Integration 60

Tables

Table 1: Conversion factors. vi

Table 2: Syntax and semantics of the Interaction Object Graph (IOG) notation (Carr, 1995). 24

Conversion Factors

Non-SI* units of measurement used in this report can be converted to SI units as follows:

Multiply	By	To Obtain
acres	4,046.873	square meters
cubic feet	0.02831685	cubic meters
cubic inches	0.00001638706	cubic meters
degrees (angle)	0.01745329	radians
degrees Fahrenheit	(5/9) x (°F – 32)	degrees Celsius
degrees Fahrenheit	(5/9) x (°F – 32) + 273.15.	kelvins
feet	0.3048	meters
gallons (U.S. liquid)	0.003785412	cubic meters
horsepower (550 ft-lb force per second)	745.6999	watts
inches	0.0254	meters
kips per square foot	47.88026	kilopascals
kips per square inch	6.894757	megapascals
miles (U.S. statute)	1.609347	kilometers
pounds (force)	4.448222	newtons
pounds (force) per square inch	0.006894757	megapascals
pounds (mass)	0.4535924	kilograms
square feet	0.09290304	square meters
square miles	2,589,998	square meters
tons (force)	8,896.443	newtons
tons (2,000 pounds, mass)	907.1847	kilograms
yards	0.9144	meters

Table 1: Conversion factors.

* *Système International d'Unités* ("International System of Measurement"), commonly known as the "metric system."

Preface

This study was conducted for Headquarters, U.S. Army Corps of Engineers (HQUSACE) under Project 4A162784AT41, "Military Facilities Engineering Technology"; Work Unit AB0, "Generation of Requirements Driven Preliminary Design Models." The technical monitor was Paul Howdyshell.

The work was performed by the Engineering Processes Branch (CF-N), of the Facilities Division (CF), Construction Engineering Research Laboratory (CERL). The CERL Principal Investigators involved were Van Woods and Michael Case. The technical editor was William J. Wolfe, Information Technology Laboratory. Don Hicks is Chief, CECER-CF-N, and L. Michael Golish is Operations Chief, CECER-CF. The associated Technical Director was Paul A. Howdyshell, CEERD-CV-ZT. The Director of CERL is Alan W. Moore.

CERL is an element of the U.S. Army Engineer Research and Development Center (ERDC), U.S. Army Corps of Engineers. The Commander and Executive Director of ERDC is COL James R. Rowan, and the Director of ERDC is Dr. James R. Houston.

1 Introduction

Introduction

Geometric reasoning is an essential aspect of practically all military planning and design problems. However, existing design methods are too slow, expensive, unpredictable, and inflexible to meet the needs of the Army Transformation. In terms of complexity theory, automated layout and configuration of geometry for design has been demonstrated to be a Nondeterministic Polynomial Hard (or NP-Hard) problem. This means it is at least NP-Complete, which leads to the worst-case solution generation by exhaustive search. Search time rises to a power of the number of independent variables. Since possibilities are potentially infinite, combinatorial explosion makes the problem computationally intractable. Optimization techniques also have difficulties due to the nonlinear functions which are intrinsic to the problem description, and the incompleteness and emergence of constraints defining the problem. The only possible ways to overcome these problems are to reduce the vastness of the search space or to re-characterize the problem to allow it to be solved by more efficient techniques.

Current state of the art research has demonstrated the applicability of parametric shape grammar, optimization, and constraint-based interactive techniques for the layout problem. However, they fall short in the following primary areas: performance, flexibility, and control. The basic premise of this work is based on the principle that a hierarchical pattern-based design method will provide a more robust approach. More specifically, this work will extract a combination of discrete and continuous topological and geometric patterns which demonstrate common applicability in military design problems.

We hypothesize that if common topological and geometric patterns can be abstracted from typical military geometric reasoning problems, that large portions of the search space can be transformed from an exhaustive search problem to a parametric and deterministic problem. Although the worst case global optimization of the problem remains NP-Complete, portions of the problem become P (solvable in polynomial time) and thus significantly less complex. Additionally, constraints can be expressed which relate to context specific issues rather than general, globally applicable constraints, allowing for more suitability, flexibility, and control towards a desired outcome.

Army Relevance

The problem of producing quality designs in a short period of time is ubiquitous throughout the Army and the Department of Defense. The rapid implementation cycles imposed by Army Transformation exacerbates the problem. In an analysis conducted by HQUSACE and reported in a briefing before the USACE Transformation Task Force, it was reported that the traditional Military Construction (MCA) process would not be capable of providing facilities for the number of Brigade Combat Teams proposed under Army transformation unless the MCA process could be significantly shortened. There are other examples that impose a requirement for rapid design.

In the increased OPTEMPO following the terrorist attacks of September 11th, 2001, it is becoming increasingly apparent that U.S. forces will be required to deploy to various locations around the world, establish a temporary base camp presence, and then move on. Establishment of each base camp represents a significant commitment of funds and materiel under conditions in which time is at a premium. In the training domain, design and modeling of Military Operations in Urban Terrain (MOUT) facilities represents a significant investment of resources that must satisfy rigorous constraints to provide realistic training. Finally, the design and siting of range facilities involves placement of standardized templates on variable terrain, subject to constraints imposed by noise, threatened and endangered species, cultural resources, dust, frequency encroachment, and other factors. In short, sustainable development of the objective force requires an approach to design that can produce rapid results consistent with a wide range of requirements and constraints.

All of these design problems share particular characteristics that are addressed by the proposed research. First, they involve reasoning about geometry. Second, they are subject to multiple and potentially conflicting requirements and constraints. Finally, they are all important to achieving the primary goals of Army transformation: force projection, ready equipment, trained and ready forces, and well-being. Army Transformation is an ambitious goal, requiring the full engagement of the entire Army team. Building the infrastructure required by the Objective Force will require at least an order of magnitude improvement in achieving the speed, quality, and flexibility of design products in the domains described above. Efforts to achieve these results through business process improvements alone have not proven sufficient. If successful, the theoretical advances enabled by this research will enable a significant degree of design pattern use and reuse that, combined with improved practices, will result in robust and sustainable mission attainment.

2 Background

Problem Definition

Many approaches have been suggested towards the solution of solving the space layout problem in architectural design. More specifically, the space layout problem is defined here as:

> *"For a given set of spaces, find possible arrangements of the spaces such that no two spaces overlap and satisfy given constraints."*

This definition is adapted from Flemming's extensive body of work which utilized a more constrained definition dealing solely with orthogonal spaces (Flemming, 1978, 1986, 1987). Spaces are considered in this work as finite basic elements with boundaries defined as *simple polygons* within a *plane*. This is most closely formally comparable to Knight's description of a medial algebra of shape with boundaries defined in U_{12}, and for diagonal shapes defined in U_{22}, using the common U_{ij} indication to represent elements of dimension i in a space of dimension j (Knight, 2003). Boundaries represent spatial centerlines, and are segmented such that each segment is adjacent to, at most, two spaces (illustrated as dashed lines in Figure 1).

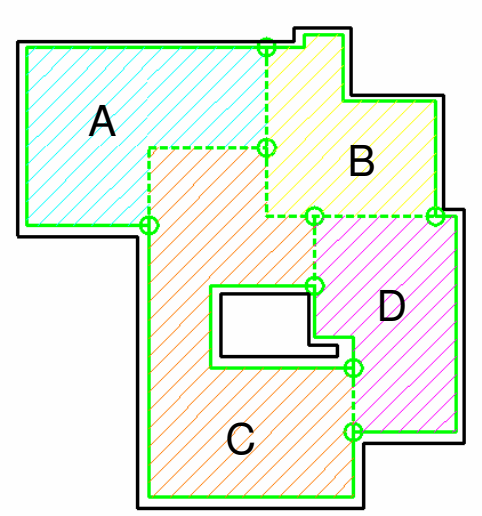


Figure 1: Example of space boundaries and their segmentation.

The specification of *simple polygon* is meant as is commonly defined in computational geometry where segments of a polygon are allowed to wrap around and touch but are not permitted to overlap, nor is the polygon allowed to have holes. This does not prevent the building from having courtyards or open spaces as is demonstrated in Figure 1. Later in the work these definitions will be expanded upon to allow intermediate states where overlapping is permitted, however they must always resolve to an end state where no overlapping occurs. Additionally, containment will be introduced which is not an equivalent condition, as the definition of overlap is confined to elements within each level of the hierarchy. A *layout* is considered a collection of spaces in a particular configuration, where no two spaces overlap, and discontinuities within the layout are permitted. The same overlap rules apply to layouts as do spaces.

Space Layout Examples

To provide a concrete example, Figure 2 illustrates an actual floorplan sketch from a design charrette for a barracks. The purpose of the charrette was to bring the user and designer together to identify and incorporate sustainability issues as early as possible into the design process. The scenario in the charrette was that the designer wanted to propose an alternative configuration for the layout that would obtain better energy performance, would take up less area on the site, and would cost less, all characteristics which would assist greatly in increasing the sustainability performance rating.

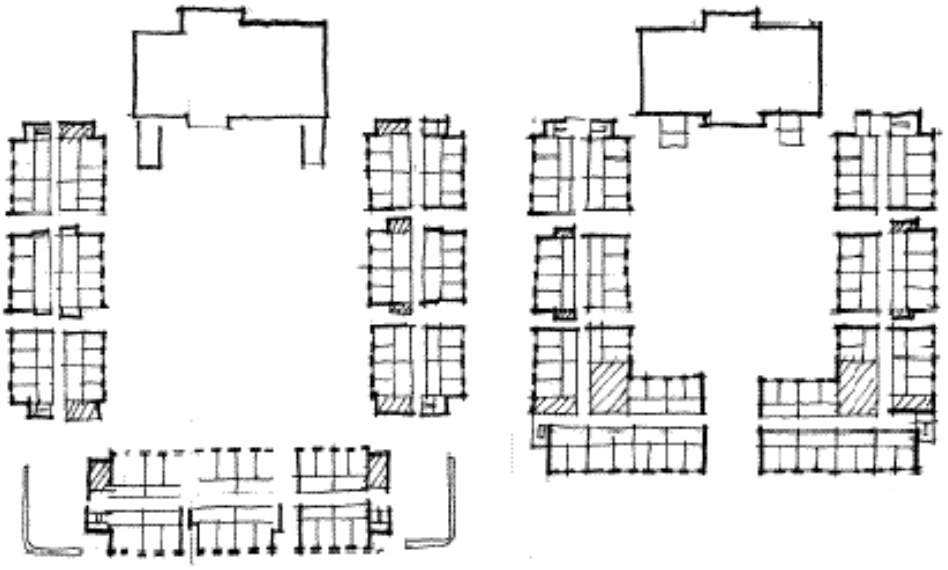


Figure 2: A sketch from a sustainability design charrette. The left image is the original plan, and the right image is the desired plan.

In the context of this research, this scenario presents a classic space layout problem. Given an architectural program with a desired number of occupants, a budget, a schedule, some performance criteria, and a customer, the designer envisioned a potential layout modification which would achieve a more desirable outcome. The concept was to change the spatial configuration of three separate linear buildings and one shared common support building to two L-shaped buildings and a common support building.

While at first glance the sketch in Figure 2 seems easy enough to produce, a designer today is currently faced with a very manual process requiring changes to complex criteria with many design parameter interdependencies in order to implement such a change. As a result the desired change was never implemented as the modifications to the design were too time consuming and complex to implement. In addition to there not being an efficient method for a user to generate such a layout modification, there is not a suitable automated computational method as well. In other words, there is no computable description or algorithm which would have produced the desired layout based upon a specified set of requirements and criteria.

Another example of space layout can be seen in Base Camp design. Base camps are commonly temporary, quickly built, military camps which support the military operations of a deployed unit. One of the goals of Army Transformation is to move a brigade anywhere in the world in 96 hours, and one division in 120 hours. Therefore, a significant controlling factor in the base camp design problem is time. Additionally, it is unacceptable to sacrifice other design criteria in return for the speedup of design. For example, a rapidly produced layout that only houses 90% of the required troops is unacceptable.

An additional controlling factor in base camp design is flexibility. As is commonly the case, base camps are commonly designed within rapidly changing military and political situations. A pre-generated solution that is reused (as is the current practice) is clearly a timesaver, however may not be flexible enough to accommodate the available sites or changing conditions. Ideally, computational assistance in the form of unassisted and/or user-mediated space layout generation would contribute significantly to the effectiveness of the development and modification of layouts. Many research approaches have been proposed that address this general issue and will be described in the next sections.

Research Approaches

Optimization

Optimization techniques applied to the space layout problem provide a robust automated method for seeking an optimal configuration according to a set of specified constraints.

The design optimization problem is formally defined as:

$$\begin{aligned} &\text{minimize} && \boldsymbol{f}(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{h}(\boldsymbol{x}) = 0 \\ & && \boldsymbol{g}(\boldsymbol{x}) \leq 0 \\ & && \boldsymbol{x} \in \aleph \subseteq \Re^{n} \end{aligned}$$

where $\boldsymbol{x}$ is the vector of design variables, $\boldsymbol{f}(\boldsymbol{x})$ is the vector of objective functions which describe the goals to be attained in the design, $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ are sets of equality and inequality constraints, and $\aleph$ is the set constraint of the n-dimensional real space $\Re^{n}$ (Choudhary and Michalek, 2005). Choudhary and Michalek continue to describe the two primary solution generation techniques: gradient-based and derivative-free. Gradient based methods are applicable to continuous variables and utilize the gradient of the objective function and constraints to guide the direction of search. With smooth, continuous functions gradient based methods are fast, can handle a large number of variables and constraints, and can guarantee an optimum with one criterion.

Derivative free techniques work through repeated function evaluations, while utilizing intelligent or random search techniques. They have the advantages of not requiring specific mathematical properties for the model functions, and can incorporate local and global search. They can be classified as either deterministic or stochastic (such as simulated annealing and genetic algorithms). The drawbacks with this approach is the low number of variables which can be typically handled, time to convergence, and lack of convergence problems.

In the research tool Interactive Layout Design Optimizer, Michalek defines the following Geometric constraints: Bound Size (min/max area, min/max length and width), Force Inside, Prohibit Intersection, Force Intersection (by overlap distance by minimum door size d), Force to Edge, Minimum Aspect Ratio, Window Size, Cost Constraint (calculated by material costs of walls and windows), Minimum Natural Lighting, Minimize Heating/Cooling Cost, Minimize Lighting Cost, Minimize Wasted Space, Minimize Accessways, Minimize Hallway. The following were identified as being disjunctive and non-

linear, and hence computationally expensive: Prohibit Intersection, Force Intersection, and Force to Edge. He also defines the following Topological constraints: Overlap Constraint, Path Constraint, Connectivity Constraint, Envelope Constraint, Planarity Constraint. To date this appears to be the most comprehensive set of constraints implemented in a space layout prototype tool.

Michalek utilizes a hybrid approach which breaks the problem down into two stages. The first stage deals with solving the topology which is combinatorial and computationally expensive, and the second with geometric optimization. Michalek utilizes a genetic algorithm (GA) and sequential quadratic programming (SQP) algorithms for searching the topological solution space, and GA's, Simulated Annealing (SA), and SQP for the geometric optimization. The tool can be run as an automated design tool or an interactive tool (Michalek et. al., 2002; Michalek and Papalambros, 2002).

The approach of separating the topological from geometric solution generation has also been explored in other work as well and has had positive results (Medjdoub and Yannou, 2000). Medgjdoub and Yannou approach the problem as a Constraint Satisfaction Problem (CSP) and utilize a constraint programming technique with a dynamic space ordering (DSO) heuristic. Their work confirmed applicability to medium sized problems by exhaustive enumeration (20 spaces, 2 stories), with optimal solutions with one criterion.

Another approach worth mentioning are the evolutionary approaches described by Gero et. al. (Damski and Gero, 1997; Gero and Kazakov, 1997; Jagielski and Gero, 1997). The primary benefit of the evolutionary GA approach is that it is capable of scaling to large problems. In one approach, it was interesting to note the choice of utilizing a halfplane representation which enabled non-straight segments (Damski and Gero, 1997). What is unclear from the work is how non-straight arrangements would be generated or transformed through the iterative search. It is assumed a random generation approach would be utilized. What was also unclear was the actual overall desirability of the solutions found, as they appeared very random and fragmented. Additionally, with the GA approach there is no guarantee that the algorithm will converge to a feasible solution, and no sort of optimality can be assured.

Michalek's and the other approaches utilizing optimization has demonstrated promise. However, the primary problems are the inability to scale to large scale problems, and secondly is the limited amount of user control. Frequently, solutions are generated which fully satisfy the constraints and optimization functions, however are not flexible enough to incorporate subjective or implicit objectives and techniques. This frequently results in the Human-Computer

Interaction (HCI) concept of the "violation of the principle of least astonishment" wherein the user receives an unexpected result and user's control is subjugated to the algorithm's process.

The implementation presented by Michalek does incorporate the option for user-mediation in the topological stage. Other approaches have also attempted to incorporate optimization and constraint solving with user interaction (Harada, 1997). However, the mapping from topology to geometry is not surjective, that is to say that for every topological solution there exists more than one potential geometric solution. If a specific geometric configuration is desirable from a particular topological configuration, there is no effective way to specify this within an optimization approach (see example in Figure 3) unless there is a specific constraint that expresses that particular geometric condition. The tradeoff is that an optimization routine can search a much larger portion of a solution space than a user can manually, however it frequently cannot ensure that the characterization of solution space accurately reflects implicit and tacit requirements which are only available utilizing human evaluation.

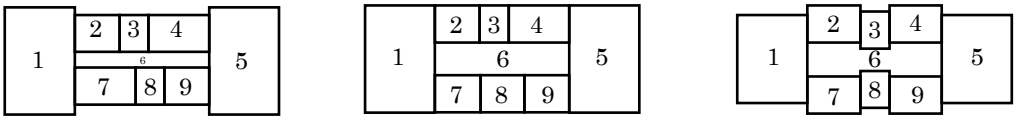


Figure 3: Three different layouts that are topologically identical and geometrically unique.

Another problem is that in automated methods, constraints are commonly defined so as to be low-level enough to apply globally, rather than incorporating unique constraints which are context specific. This is not in reference to instance specific values, but rather context specific constraint types. Also, if a new global constraint is desired, the cost and skill set required to develop additional constraint definitions makes it prohibitive to designers. And finally, the inability for automated approaches to handle non-orthogonal and non-rectangular spaces in a meaningful, non-random fashion is a limitation in a real world context. For example, an auditorium is commonly non-rectangular, and site conditions commonly require non-orthogonal site layouts.

Approaches which are non-deterministic also seem to cause problems within the context of design. By deterministic, this implies that a solution procedure generates the same solution each time if given the same starting point and constraints. To many designers this implies transparency of method, that the procedure could demonstrate to the designer how a particular result was

achieved. Non-deterministic methods are sometimes distrusted by designers because they very rarely produce the same result and the rationale for how a particular solution was generated is difficult to demonstrate.

Within space layout specifically, many optimization approaches make assumptions which are actually design variables which carry many interdependencies with other design variables. For example, many times an optimization routine transforms a design to generate the next layout which will be evaluated in such a manner that takes the take quantity of spaces as given. Instead what is observed empirically is that a designer might want to perform a tradeoff analysis of size of room vs number of rooms or any number of other options based on what issues are of relevance to the particular design context.

The approach taken here and described in more detail later on is to treat the computational mechanism more as a descriptive language rather than an optimization-driven approach. This trades the minimization of an optimization function approach for more of a systems of systems or series of equations type approach for which traditional optimization routines can play a part within if desired.

Shape Grammars

Shape grammars have been shown over the last two decades to provide a powerful method for describing, generating, and analyzing designs. Particularly interesting is the observation that rules which describe a particular style tend to produce variants in the corpus. Such was demonstrated with the Queen Anne grammar and others (Flemming 1987). Shape grammars are a generative mechanism invented by Stiny which takes the basic form:

A five-tuple (***S***, ***L***, ***T***, ***G***, ***I***), where

- ***S*** is a set of shape rules in the form of A → B. This indicates that when a shape A is found, it is substituted by a shape B.
- ***L*** is a set of labels that are used to control computations.
- ***T*** is a set of transformations (rotation, translation, scaling, reflection or any combination thereof).
- ***G*** is a set of functions that assigns values to parameters in rules
- ***I*** is the initial shape to which the first rule applies.

Another way this can be described is:

$$t(g(A)) \leq C \quad [1]$$

$$C' = C - t(g(A)) + t(g(B)) \quad [2]$$

This indicates that [1] is the match step where if A is in C subject to g functions and t transformations, that [2] A is subtracted from C and B is applied resulting in the new state C'. The shape grammar formalism defines some very interesting aspects not found in other computational methods which make it particularly amenable to design. The three main unique characteristics include maximal shapes, non-atomicity, and emergence. For a more complete explanation see (Stiny, 1980; Stiny, 1994).

Recent developments have added significant capabilities to the formalism. Two notable extensions include the addition of constructs for formal logic and shape algebras (Chase, 1996; Stouffs and Krishnamurti, 1994), and secondly on the extension to include descriptors which provides a control structure or meta-language for shape grammar execution. This introduces the means to "1) explicitly determine the sequence in which a set of rules is applied, 2) restrict rule application with a filtering process, and 3) use context to guide the rule matching process" (Liew, 2004). The evaluation here of Liew's work is that it adds to the foundational computational concept of the rewrite rule the notion of a control structure, two elements required of any complex computational language. Prior to Liew's work, the application of shape grammars tended to be more informally defined.

An additional avenue which has been demonstrated with shape grammars is their integration into an optimization algorithm where the grammar rules are applied to evolve the design during the iteration phase. The appeal here is that the transformations adhere to the designer's goals rather than subjecting the geometry to (seemingly) random perturbations. Shea's eifForm application (Shea and Cagan, 1999) for generating structurally sound freeform designs is one example. Another is in the layout module of the Software Environment to Support the Early Phases in Building Design (SEED-Layout) where a constraint propagation methodology with grammar influences was developed (Flemming and Woodbury, 1995).

One of the limitations with shape grammars is the lack of robust and generally applicable implementations from which rules and transformations can be tested. It is currently a computational theory without a complete and general computational implementation. Prior issues with ambiguity of sequencing, rule

application, and rule matching have been recently addressed by Liew's control structure, which was seen as a major advancement towards the eventual utility and usefulness.

Case Based Reasoning

Case based reasoning (CBR) is a theory of problem solving that is fundamentally different from other major artificial intelligence (AI) approaches. Instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between elements, CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations which are termed "cases". A new problem is solved by finding a similar past case and reusing it within the context of a new problem. A second important difference is that CBR incorporates incremental, sustained learning, since a new experience is retained each time a problem has been solved making it available for future problem solving. The approach in this work utilizes the concepts of design patterns which is a form of CBR, but not formally so. The definition of patterns and their utilization will be described in the next chapter.

CBR in the geometric context has had some integration with constraint techniques which have resulted in positive performance results. One notable instance is by Faltings in the two prototype systems FAMING and CADRE (Faltings, 1997). In Faltings work he correctly identifies that in certain design contexts the space of acceptable designs is very sparse, and that the utilization of cases avoids the problem of determining how to generate feasible designs from the infinite set of possible geometric combinations.

Parametric Modeling

There has been little work done involving the application of parametric modeling to spatial configurations. Prior research at CERL with Facility Composer and the Modular Design System (MDS) has demonstrated that for many facilities there is a direct relationship between the configuration of a space with the function or functions it supports (Facility Composer, 2005; Modular Design System, 2005). In those systems, design knowledge was encapsulated in parametric "modules" or "preconfigured solutions". These were then reusable in other design problems where spaces of similar functions were desired.

The parametric capabilities were simplistic allowing, for example, the perimeter of a space to be dynamic, while the contents would only update if there was an associated discrete matching layout of the same size. For example, a 10x10 office could be scaled to a 12x12 and if a 12x12 interior layout for furniture, electrical,

mechanical and structural existed it would populate the space with that information, otherwise it would place the next smallest size centered within the bounds of the spatial centerline for a designer to manually arrange at a later time.

Part of that work also demonstrated that a large degree of building elements (interior/exterior walls, floors, ceilings, columns, roof), could be derived from a spatial centerline definition of the spaces in a building with associated design criteria. This illustrated that downstream modeling could be generated, and the primary objective of the designer then became the process of manipulating the spatial configuration from which the building elements would automatically be generated. There wasn't any automation, however, that explicitly assisted with the automation of the spatial configuration process.

Current Approaches in Practice

In contrast to research prototypes and theories described previously, this section will describe current capabilities and limitations.

Traditional CAD

Traditional computer-aided design (CAD) systems provide three primary strategies for supporting computational assistance for geometric design: non-persistent geometric transformations, utilization of reusable groups of geometry, and tools.

The primary and most basic operation of CAD systems can be described as providing an interface for users to apply geometric and Boolean transformations to persistent geometric primitives. The basic process involves selection of a particular command which is applied to a selected set of geometry from which transformations (like mirroring) are applied and a result is generated. The result does not store the method that created it, and hence is not aware of the history of how it was generated. This results in geometric configurations which undergo a linear progression of successive transformations guided by user direction. If a user changes their mind about a previously applied transformation or transformation value, their only choice is to revert back to the next prior state and to reapply the newly desired transformations or values rather than being able to change a prior value to match a new design rationale and recalculating the design.

The second basic strategy involves providing reusable groups of geometry in different design contexts. These are commonly referred to as *blocks* or *cells*, or *referencing*. These blocks are simply subsets of designs which can be reused by reintroducing them into new designs. As simple as it seems, this is one of the greater efficiency multipliers for designers using CAD systems today. Unfortunately the elements are exact copies which are no easier to modify to achieve new results. This commonly results in mass replication not because that is what would be advantageous for a design but rather because of the simplicity of creating a lot of content without a lot of effort.

The third approach involves providing problem specific decision support in the form of tools. This enables computational support for higher-level design semantics which utilize constructs such as looping and conditional logic along with user input to define a series of steps which can produce and refine meaningful results. A simple example would be in a dimensioning tool. Tools can also come from custom software development such as with the Corps of Engineers developed Theater Construction Management System (TCMS 2005). TCMS assists a user by providing a case base of reusable blocks in an easy to use catalog as well as to tie in associated data such as a bill of materials or construction sequence data. These approaches are notoriously difficult to implement and maintain, however result in greater associated efficiencies.

BIM

More recently the CAD industry has begun to provide tools specific to particular design domains. For example, instead of a general geometry processing engine that could be utilized in any number of contexts, CAD vendors are now producing tools specific to architectural, structural, mechanical and electrical design within the building domain. This movement has come to adopt the term "Building Information Modeling" or BIM to reflect the higher-level semantic data associated with the graphical elements. The graphics primitives, and the tools which support their creation, are now dealing with objects such as walls, floors, doors and windows, rather than simple geometry.

Unfortunately, with traditional CAD and now with BIM, the problem is still one of a process of manual management of disparate pieces of geometry. Even with parametric modeling tools, it is still a very cumbersome and error prone process to make changes which would propagate to many areas of the design. The cost of making even small changes is still either too time consuming or unpredictable. Shifting the process from a drafting or documentation centric process to a modeling approach is clearly an improvement. However, real advancement will come when we move from modeling individual pieces to the act of designing

wherein multiple pieces can be modified, even dramatically, in a predictable manner.

Characteristics of Design

Kim provides an excellent empirical description of designers and the design process: "What happens in the mind of the designer is a series of incessant hypotheses of the potentiality of an idea, synthesis of a scheme, and judgment of the plausibility of the hypothesis, guided by a holistic understanding of the design situation" (Kim 1980). Space layout is a process that occurs throughout design. However, it is a predominant activity during the early phases of design where significant decisions which impact the facility's lifecycle performance are established. Facility design differs from other domains that deal with space layout, such as VLSI (very large scale integration) circuit design, in the nature of the problem characteristics.

VLSI and other certain domains which utilize layout routines commonly have well-defined optimality criteria. For example, with VLSI, design criteria consists of minimizing, maximizing, or balancing the criteria of area, aspect ratio, wire length, wire delay, wire congestion, and power consumption. This does not simplify the computational problem by any means, as VLSI floorplanning has been demonstrated to be NP-Hard. However, in this context, when an optimization algorithm is developed which can demonstrate improved asymptotic performance or better coverage of the problem search space, this demonstrates a clear advancement in the capability of chip design.

With building design and other "open" domains the situation is not as straightforward. The complicating factors are that 1) choices and constraints are not all guaranteed to be known beforehand, 2) if known, they may not be formally expressed, 3) there can be conflicting criteria, and 4) the relevance, meaning, or rationale for those constraints can change in light of new information. Additionally, the result cannot be formally evaluated as a design's value is subjected to complex cultural conditions. Optimal in this situation is contextual and the context readily evolves throughout the design process. In the science of design theory, this is commonly referred to as the emergent nature of design (Chase, 1997), or more colorfully as the nature of "wicked" problem domains (Rittel and Webber, 1973).

In closed domains, Artificial Intelligence theories treat the notion of ***search*** within the solution space for the optimal result as the predominant problem solving methodology. The traditional search paradigm operates on the

assumption that a change in the goal state results in a different problem space. In open domains it is more appropriate to describe the approach as the process as ***exploration***. With the exploration model of problem solving, the interdependence of the problem specification and the solution are explicitly addressed. By far the most rigorous argument for the notion of design exploration comes from Woodbury, a long standing researcher in the design theory community (Woodbury, 2006). Readers are encouraged to review the article for a complete appreciation of the case for an exploration based approach to design within the context of design automation and design theory.

Smithers and Troxell provide a formal description of design processes based on the model of exploration (Smithers and Troxell, 1990) as:

A six-tuple ($\boldsymbol{P_s}$, $\boldsymbol{T}$, $\boldsymbol{R_{i\text{-}f}}$, $\boldsymbol{H}$, $\boldsymbol{D_s}$, $\boldsymbol{E_d}$), where

- $\boldsymbol{P_s}$ is the space of possible solutions defined in terms of precedents that are represented by completely defined states. The structure of the space is described by sets of properties, their associated value ranges, and relations between properties.

- $\boldsymbol{T}$ is a set of operations that compute new properties from existing ones. State descriptions are refined or inconsistencies are detected.

- $\boldsymbol{R_{i\text{-}f}}$ is an ordered set of non-empty sets of properties. Attributes within these sets represent the required design solution. $\boldsymbol{R_i}$ is the initial set, $\boldsymbol{R_f}$ the final set defining the found solution. In between are sets of intermediate requirements with the current one being referred to as $\boldsymbol{R_c}$.

- $\boldsymbol{H}$ is the design history; it is a sequence of property sets and applied transformations.

- $\boldsymbol{D_s}$ is the set of solution states that possess properties specified by a $\boldsymbol{R_c}$.

- $\boldsymbol{E_d}$ is the process that extends $\boldsymbol{D_s}$ by taking $\boldsymbol{P_s}$, $\boldsymbol{T}$, $\boldsymbol{R_c}$, $\boldsymbol{H}$, as input and outputting a series of transformation on $\boldsymbol{R_c}$.

Herzog provides a good description of the process when he states:

"It is part of the design process to explore the space of possible designs to discover the structure of the solution space. While the design process is going on, different sets of requirements emerge. At the same time design specifications are acquired that partially meet the current set of requirement descriptions. The

result of the design process is a consistent requirement description and an associated design specification. Design requirement and specification are two tightly coupled aspects of the same process. It is thus often impossible to decompose design problems and find independent solutions to these subproblems (Herzog, 1994)."

In "The Computability of Architectural Knowledge," Oxman and Oxman describe design as a simultaneous top-down and bottom-up process where the solution at any given level in the decision making hierarchy forms the context for the next (Oxman and Oxman, 1990). The position taken in this work is as design being similar to solving a system of equations for which the parameters and relationships of the parameters are not fully known beforehand. After a few iterations, and evaluation of the results, many or most of the parameters and their relationships are established. Once a solution has been specified which demonstrates utility in certain contexts, it can be used like a formula and solved for different parameter values depending on the context and desired results, and adapted to new contexts. This approach allows for solving of expressions based on fixed and variable, or known and unknown conditions. This also reveals when a pattern cannot be solved due to too many unknowns which can then guide the design session towards attempting to make a determination for resolution or finding a different approach.

Summary

The original characterization of the space layout problem by Flemming has some definite computational advantages in that it limits the complexity of the layout problem yet is limited in real world application. Optimization approaches face challenges in the primary areas of 1) performance scalability for realistically sized problems, 2) lack of flexibility to describe new desirable conditions in the form of constraints or optimization functions, and 3) lack of control towards achieving solutions which incorporate tacit and emergent requirements and support user control. What is needed instead is a methodology which can incorporate the strengths of the optimization approaches within a framework that enables the user to maintain finer control over the exploration of the design solution space.

3 Pattern Based Approach

Definition of n-Dimensional Abstraction Patterns

A hierarchical, pattern-based approach is defined here that addresses the limitations described in the prior section. The notion of a Pattern as intended in this work is influenced by Christopher Alexander's definition:

> "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. […] As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves. […] As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant." (Alexander, 1977)

The term "n-dimensional abstraction patterns" is used in this paper to describe a pattern definition that is multidimensional (patterns can contain other patterns) and hierarchical (patterns can extend from more fundamental patterns). Given that a pattern describes a certain design practice, its parameters and transformations carry significance towards generating a desirable result. This is in contrast to optimization methods which attempt to find generally applicable low-level constraints (such as an adjacency constraint across all spaces in a layout). Instead, a pattern defines a unique (and possibly highly complex) set of parameters and algorithms which generate allowable variations which enable it to produce meaningful solutions. The pattern effectively encapsulates the implicit and tacit requirements frequently found in design, and produces desirable results as long as it is utilized in the appropriate context.

Pattern language

The pattern based approach used in this work utilizes a number of existing constructs to define a generative computable geometric descriptive pattern

language. It would be within the scope of additional applied research to create a computational implementation of such a language, whereas here the objective is to demonstrate the concept, its utility, and feasibility. The language is primarily based on a notation called Interactive Configuration Exploration (ICE) as its primary geometric and generative description, with extensions to provide additional control structure constructs, an equation-like approach to parameter definitions, an overall pattern-like syntax, and Interaction Object Graphs (IOG) for extended interaction definition, all of which are described in more detail below.

Primary Syntax

Each pattern definition has the following sections: **Signature**, **Also Known As**, **Description**, **Required Inputs**, **Optional Inputs**, **Definition**, **Relationships**, **Interaction**, and **Examples**. The **Signature** describes the name of the pattern and the order of the parameters, much like a typical function definition in any of today's modern procedural programming languages. Multiple signatures can exist per pattern. The signature and assignment syntax is:

$$ResultantShape = \text{PatternName}\ [\ p_1,\ p_2,[\ p_3\ ]\ ]$$

Inner brackets indicate an optional parameter. ResultantShape represents the output of a Pattern, and PatternName represents the call to instantiate a pattern. **Also Known As** provides substitute pattern names which can be used in exchange for the primary pattern name. **Description** is utilized to provide an explanation of the pattern, and ideally will describe the context of utility of the pattern as originally intended by Alexander, and an indication of the behavior its user should expect. **Required Inputs** and **Optional Inputs** provide explanations of the required and optional input parameters. Optional parameters indicate default values in bold used if no values are provided. **Definition** is the bulk of the pattern logic which algorithmically defines the behavior that the pattern exhibits. The ICE generative formalism is utilized, which is described in more detail in the next section.

Relationships are an extension to the ICE formalism (described in more detail later in this section) which describe the relationship that a set of parameters should maintain. In this way they act more like formulas which can be utilized to manipulate a design rather than as an assignment of a value to a variable. For example, if a rectangle pattern describes parameters Length and Width, and the following additional relationships were defined, one would be able to manipulate Length and Width in relation to Area and/or AspectRatio. Depending on the

known values, you could solve for the unknown. Also, a known value can be manipulated independently while keeping the others constant.

*Area == Length * Width*
AspectRatio == Length / Width

The mechanism for implementation of this concept (as well as documentation of associated complexities) have been described in numerous other places based on the notion of solving for degrees of freedom. The convention of using two sequential equals signs is used to remind the user that this is not a simple assignment of values to a variable, but rather an equation which binds the variables in a particular set of relationships.

Where a direction vector is required, it is commonplace to indicate this with the shorthand $p_1 \rightarrow p_2$ which indicates the vector generated from the point p_1 to p_2. Additionally, the shorthand convention (x,y) is used to indicate a point at coordinate x, y; the minus symbol to indicate the shortest distance between two elements such as the distance between two points, ie. $p_2 - p_1$, or the shortest distance between a point and a line, ie. $p_1 - l_1$; | | to indicate a line from origin to endpoint, ie. $|p_1, p_2|$; and the tilde to indicate the angle between two valid elements (or more accurately in ICE, two transformations) such as two lines, ie. $|p_1,p_2| \sim |p_1,p_3|$.

Points in this approach are treated as coordinate systems, meaning that using a point as an origin for a transformation will force the transformation to occur within that coordinate system. As an example, the move transformation of a point p_2 relative to another point p_1 moves p_2 x and y units in the coordinate system of p_1 (which could very well be rotated to align to another element).

The **Interaction** section utilizes the IOG formalism (described in more detail later in this section) within a statechart to describe the user interaction definition. IOG's are necessary in order to provide a complete language specification, however was not a primary focus in the pattern definitions defined in this research. It became clear early on that additional examination would be required from end users in order to determine useful interaction sequences, which was concluded to be a non-essential task for the purposes of this basic research. The primary objective identified here was to specify and demonstrate the key elements of the pattern approach. The last section, **Examples**, are simply instantiations of the patterns which illustrate its behavior.

Interactive Configuration Exploration (ICE)

The formal generative notation called "Interactive Configuration Exploration", or ICE, was developed at Carnegie Mellon University. Its primary purpose was for describing geometric configurations by means of formally defined generative and relational structures (Moustapha, 2004). It has been utilized for describing user transformations during protocol analyses of designers (Akin and Moustapha, 2003). It was not created solely for that specific purpose however, but rather for more general purposes as a formal, generative, descriptive language for the creation and manipulation of geometric configurations by designers and researchers.

Two other languages were evaluated, one defined by Leyton (Leyton, 2001) and another by Cha and Gero (Cha and Gero, 2001). Leyton's notation was interesting, yet was determined to be too verbose and more focused on providing a means of describing cognitive processes. Cha and Gero's approach had many similarities to the ICE notation in its intent, however was not as feature rich or rigorous as the ICE implementation.

The pattern **Definition** section utilizes ICE extensively to define the pattern behavior. The serious reader needs to review Moustapha's paper in complete detail before proceeding, as a significant portion that follows in this paper will rely on its syntax and semantics. What follows here is a synopsis of the language syntax and constructs to give the more casual reader a brief orientation.

The ICE notation is based on a notion of a regulating element. The regulator is the primary means by which geometry is created, modified, and defined. In all there are five types of regulators with their associated Greek representation in parentheses: transformations (Δ), constraints (ϕ), hierarchies (Ψ), variations (Ξ), and operations (Ω). The language is very flexible as it allows transformations of geometry as well as transformations of regulators to be defined.

An extension of the ICE notation is defined in this work that allows for more complex configurations to be developed through the application of conditional logic and iteration. Conditional expressions and iteration are utilized to build complex chunks of reusable design logic that can respond to different contexts rather than following simple sequential progression.

The syntax employed for the conditional evaluation is:

if *LHS* = *RHS* then ExpressionToEvaluate

where LHS indicates the left hand side, and RHS indicates the right hand side of any operation (not just equality as shown in the example) which results in a boolean condition. The commonly understood *else* and *elseif* syntax is also available but not shown.

The syntax employed for iteration is:

$$ResultantAggregate = \sum_{i=0}^{n} \begin{bmatrix} ResultantShapes_1 = \text{Expression}_1[\, i \,] \\ ResultantShapes_2 = \text{Expression}_2[\, i \,] \\ ResultantShapes_x = \text{Expression}_x[\, i \,] \end{bmatrix}$$

where *ResultantShapes* is a vector of shapes which can be indexed into within the number of iterations defined by the series, and *ResultantAggregate* is the conjunction ($\wedge$) of $ResultantShapes_{1..x}$.

If more flexibility and greater control is desired, the *for* loop more commonly utilized in procedural programming is also available:

for (VariableInitialization, BooleanExpression, VariableIncrement)
$ResultantShapes_1$ = Expression_1[*variables*]
$ResultantShapes_2$ = Expression_2[*variables*]
$ResultantShapes_x$ = Expression_x[*variables*]

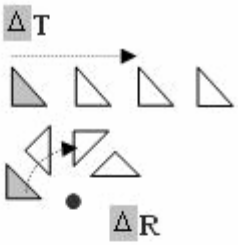
where VariableInitialization indicates the setting of starting values of one or more variables, BooleanExpression indicates the stopping condition for the iteration with any expression that results in a Boolean condition, and VariableIncrement indicating the change in value to any variables to apply at each successive iteration.

The final extension utilized in this work is that all variables are actually arrays of variables meaning that a user can pass a series of shapes as easily as one could pass a single shape, where for the single shape case the variable is simply an array with one slot filled. The pattern implementer must consider this when defining the pattern to the greatest extent possible. In order to be fully implemented, the extensions described would require formalization utilizing Backus Normal Form (BNF) therefore allowing a complete non-ambiguous definition as well as an automated method for generating an associated parser.

Following are a few of the tables reprinted from Moustapha's manuscript which demonstrate the ICE syntax.

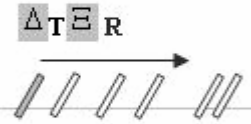
Regulators based on geometric transformations Δ

Translation	$\Delta\mathbf{T}(\text{shape})$	$\Delta\mathbf{T}^1[\ \{\bar{p},\bar{t},d,n\}\ (\text{shape})\ ]$
Rotation	$\Delta\mathbf{R}(\text{shape})$	$\Delta\mathbf{R}^0[\ \{\bar{p},\alpha,n\}\ (\text{shape})\ ]\ )\ ]$
Mirror	$\Delta\mathbf{M}(\text{shape})$	$\Delta\mathbf{M}^2[\ \{\bar{p},\bar{t},\bar{v},n\}\ (\text{shape})\ ]$



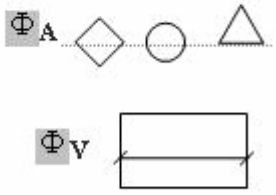
Regulators based on variations Ξ

Rhythm	$\Xi\mathbf{R}(s_0 - s_n)$	$\Xi\mathbf{R}\ [\ \{a,\boldsymbol{f},c\}\ (s_0 - s_n)\ ]$



Regulators based on constraints Φ

Alignment	$\Phi\mathbf{A}(s_0 - s_1)$	$\Phi\mathbf{A}^2[\ \{\bar{p},\bar{t},\bar{v}\}\ (s_0 - s_1)\ ]$
Dimension	$\Phi\mathbf{V}(s)$	$\Phi\mathbf{V}[\{min,max,mod\}(s)]$

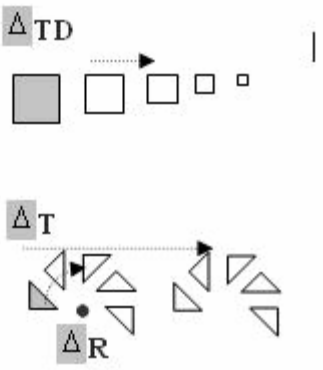


Regulators based on operations Ω

Difference	$\Omega\mathbf{D}(s_0 - s_1)$	$\Omega\mathbf{D}\ [\ \{\}\ (s_0 - s_k)]$

Composition strategies

Simultaneous composition	$\Delta\mathbf{TD}(\text{shape})]$	$\Delta\mathbf{T}^1\Delta\mathbf{D}^0[(\bar{p}_T,\bar{p}_D\bar{t},\bar{k},d,n)(\text{shape})]$
Successive composition	$\Delta\mathbf{T}(\Delta\mathbf{R}(\text{shape}))$	$\Delta\mathbf{T}^1[\ \{\bar{p},\bar{t},d,n\}\ ($ $\Delta\mathbf{R}^1[\ \{\bar{p},\bar{t},\alpha,n\}\ (\text{shape})]\ )]$



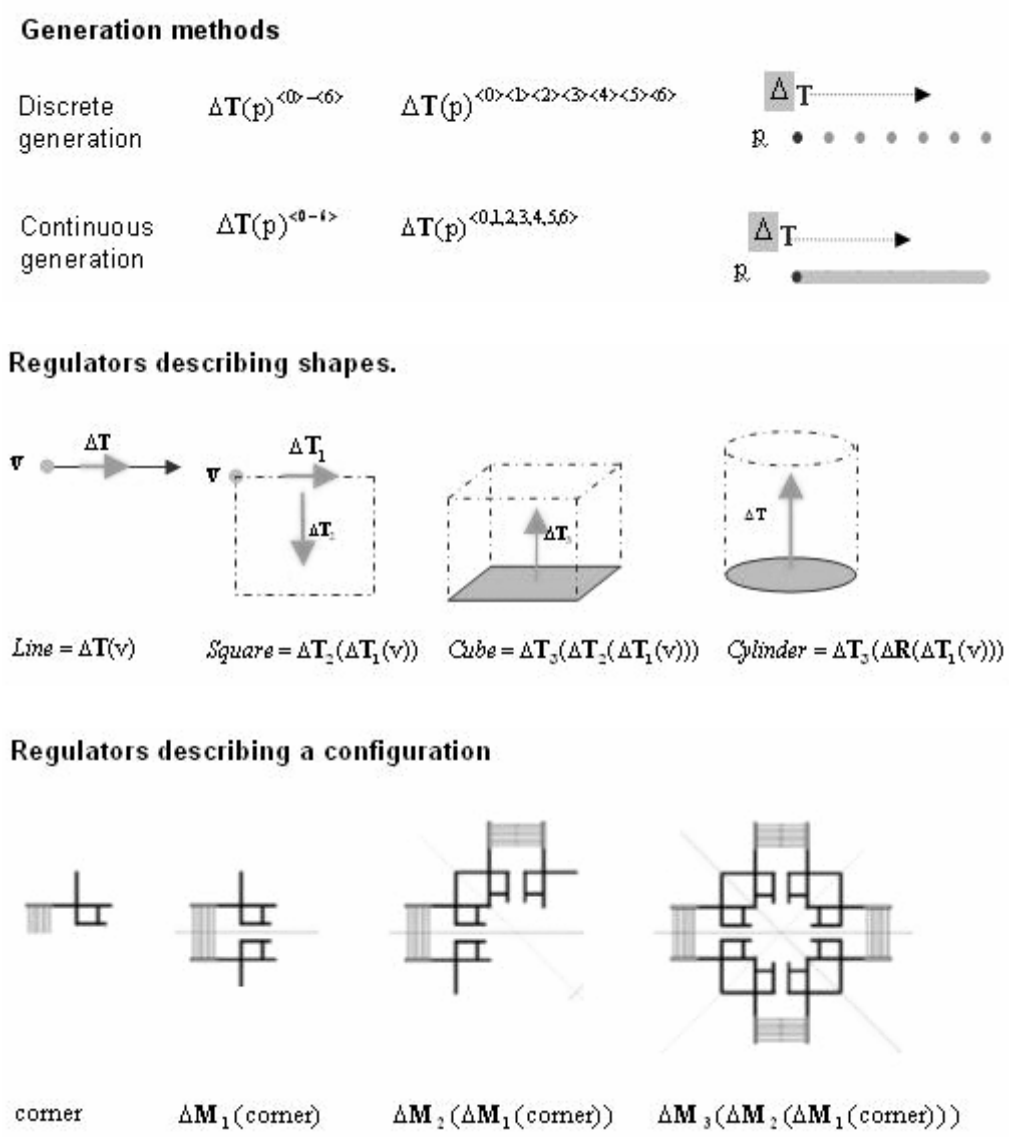


Figure 4: Syntax and semantics of the Interactive Configuration Exploration (ICE) notation (Moustapha, 2004).

Interaction Object Graphs (IOG)

Interaction Object Graphs, or IOG's is a notation developed at the University of Maryland (Carr, 1995). While ICE already provides a regulator mechanism for controlling user interaction, the notation utilized within IOG's was found to be a desirable way to control interaction sequences. First, it enables a meta-control structure, much like the conditional and iteration constructs defined in the previous section. This means that IOG's can operate on regulators as well as across groups of regulators, allowing finer grained control and more complex

interaction sequences to be defined. Additionally, it is a compact and easy to follow notation.

Defining user interactions was not a significant focus of this work, however a complete implementation of a pattern language would require a construct such as this as the intent is for a pattern description to be able to be controlled algorithmically as well as through user interaction sequences. Combined within a statechart, IOG's can express complex, non-linear and conditional user interactions with a minimal overhead in complexity. Carr's work demonstrated that IOG's are Turing-complete, therefore demonstrating that it is a robust and expressive computational specification methodology.

System Attribute	Description
M@	A point containing the current mouse location.
MΔ	A point containing the difference between the current mouse location and the previous mouse location.
<key>Dn	The <key> is currently held down. This is limited to modifier keys such as Shift, Ctrl, or Alt.
<key>Up	The <key> is not currently held down. This is limited to modifier keys such as Shift, Ctrl, or Alt.
KbdBuffer	A holding string which accumulates keyboard input translated into characters.

Event Notation	Event Description
M↓ or Mv	Primary mouse button pressed. (Usually the left.)
M↑ or M^	Primary mouse button released.
M↓↑ or Mv^	Primary mouse button clicked.
M↓↑↓↑ or Mv^v^	Primary mouse button double clicked.
M2↓ or M2v	Second mouse button pressed. (Usually the right.)
M2↑ or M2^	Second mouse button released.
M2↓↑ or M2v^	Second mouse button clicked.
M2↓↑↓↑ or M2v^v^	Second mouse button double clicked.
M3↓ or M3v	Third mouse button pressed. (Usually the middle.)
M3↑ or M3^	Third mouse button released.
M3↓↑ or M3v^	Third mouse button clicked.
M3↓↑↓↑ or M3v^v^	Third mouse button double clicked.
ΔM	The mouse has changed position.
<key>↓ or <key>v	The key represented by <key> has been pressed. For example Shift↓ is the shift key has been pressed.
<key>↑ or <key>^	The key represented by <key> has been released.
ΔKbdBuffer	The KbdBuffer has been changed.
time>nn	At least nn units of time have elapsed. This may be a number or a system constant such as CLICKTIME.
AppFeedback	The application has changed some data value in the interface.

Table 2: Syntax and semantics of the Interaction Object Graph (IOG) notation (Carr, 1995).

Process Architecture

Patterns represent procedural chunks of design logic chained together to generate a description of a particular design solution. Optimization algorithms can be utilized to order a configuration within the context of itself (denoted as "optimize internally" in Figure 5) or across a dependency chain of patterns (denoted as "optimize externally with stopping rule" in Figure 5). Optimize in this context is intended differently than is thought of traditionally, where here it can represent any algorithm which affects a configuration, which may or may not lead to cycles. If cycles are introduced, then an expression is required which defines a stopping condition, which is re-evaluated at each cycle.

This procedural-like approach is specifically chosen as it encourages a level of user traceability of the sequences taken to get from an initial state to another state. The input parameters are preprocessed according to the defined Relationships to ensure the problem is never under or over constrained. Additionally, an update cycle is triggered when any pattern changes therefore effectively enabling child patterns to trigger updates in parent patterns, which could be described otherwise as supporting top-down or bottom-up processes.

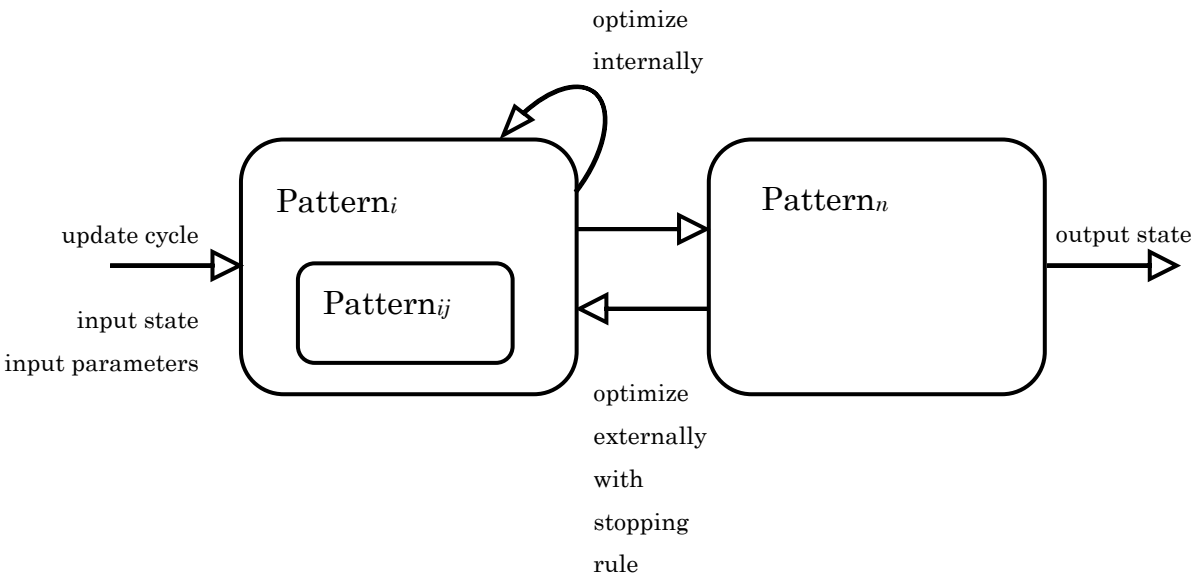


Figure 5: Process architecture.

Another general principle of the language is that it is never in an over or under constrained state, or more precisely that there are no inferences made or heuristics applied as to the method a computation should assume. All behaviors are required to be delegated to the designer's definition and decision, as a basic

principle of this approach is that all nuances are under direct designer control. Inferences and use of complex optimization techniques become a choice, not a foundational aspect built in as a requirement of the approach as in other optimization-based approaches.

While not explored in any detail in this work, patterns could also utilize a matching scheme similar to shape grammars. The idea would be that for any collection of geometry there could be multiple applicable patterns, rather than there being a one to one association between a configuration and a pattern. One additional requirement would be that a pattern would then have to additionally define its matching condition.

Detailed Example

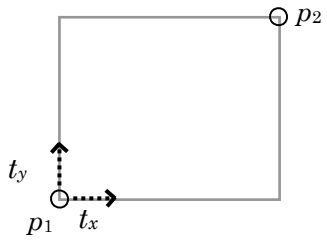
The following is a detailed annotated example of a simple pattern.

RectanglePattern

Signature:

RectanglePattern [p_1, p_2 [α]]

RectanglePattern [p_1, l, w , [α]]



rectangle = R [(0,0), 6, 5]

Also known as:

R, Rect, Box

Description:

This is a very simple example pattern that is intended to demonstrate how pattern definitions work. The rectangle is a basic primitive in any design system, yet its complexity is revealed when one studies any implementation that supports user interaction. There are numerous ways that rectangles can be interactively placed depending on the desired behavior and context. This pattern demonstrates but one, a rectangle by two points. For the sake of comparison an identical feature coded in MicroStation/J which illustrates relative compactness of the pattern approach is available in Appendix 9.

Required Inputs:

p_1 = origin, p_2 = extents

l = length, w = width

Optional Inputs:

α = rotation [angle = **0**]

Definition:

if $\alpha \neq 0$ $p_1, p_2 = \Delta R^0 [\{ p_1, \alpha, 1\} (p_1, p_2)^{<1>}]$

$rect = \Delta T_2 [\{ p_1, p_1 \rightarrow (p_1.x, p_2.y), 0, 1\} (\Delta T_1 [\{ p_1, p_1 \rightarrow (p_2.x, p_1.y), 0, 1\} (p_1)])]$

Relationships:

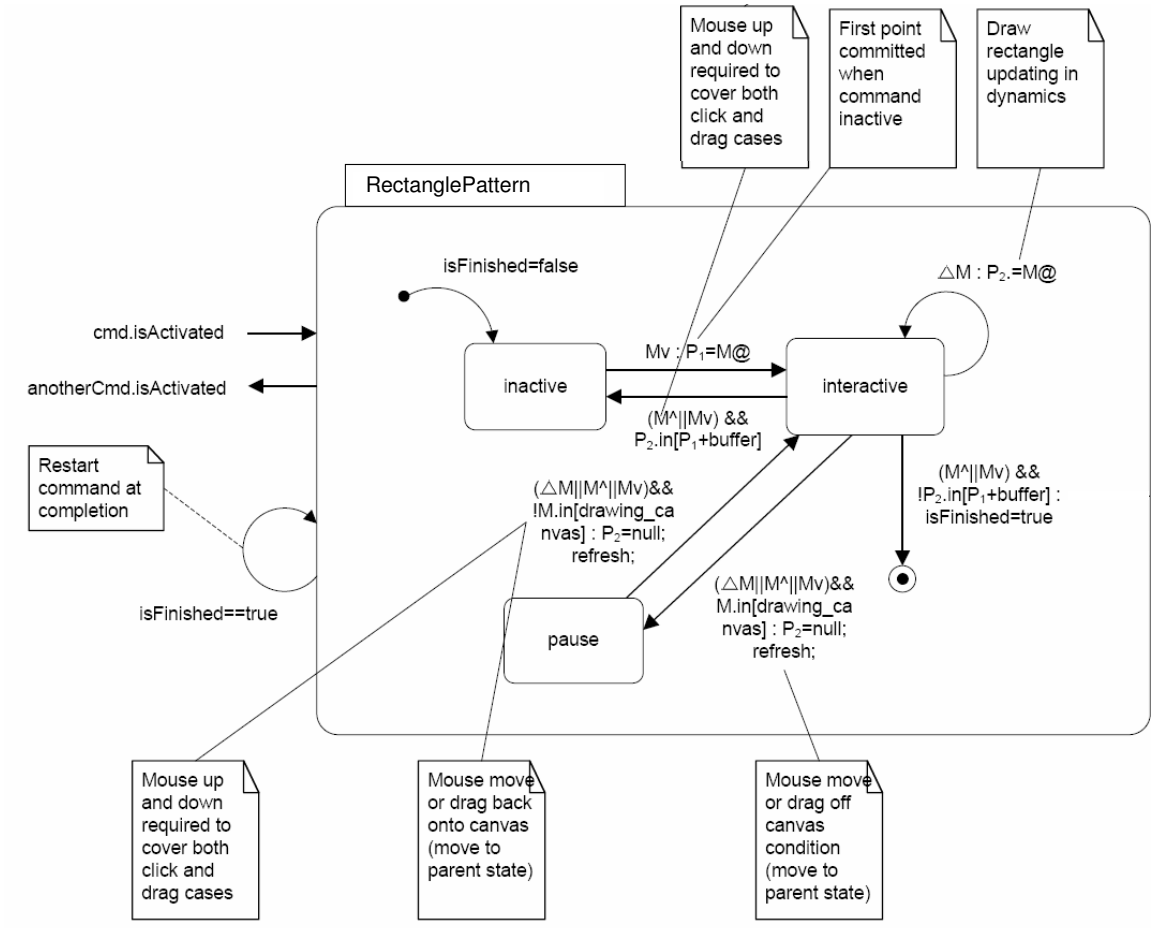
$Area == l * w$

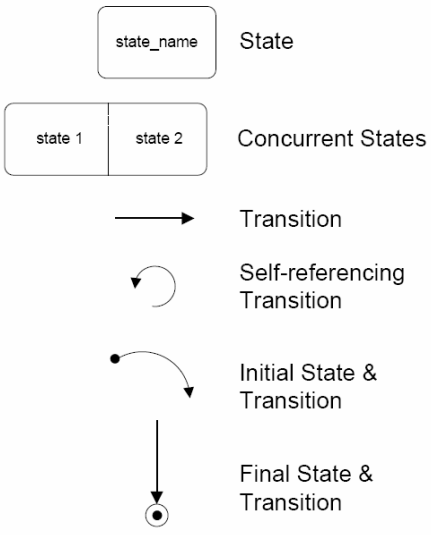
$AspectRatio == l / w$

$l == p_2.x - p_1.x$

$w == p_2.y - p_1.y$

Interaction:





Operations and Operators

IF : THEN
IF Condition True (Event/State) : THEN perform one or more procedures (value assignment)
Example: Mv : P_1=M@
Translates to: IF mouse down click event is triggered
THEN set P_1 to be the new mouse location

== test for equality
Syntax: valueA == valueB
example - height==6 tests whether height variable is equalTo 6, if so then expression returns true

= value assignment operator (example isFinished=true)
example - isFinished=true assigns value true to isFinished

System Events

△M = mouse moved
M^ = mouse up
Mv = mouse down

System Objects

P_n = Point, (P.x, P.y, P.z), where n=ID
M@ = mouse location, returns Point

Explanation of Detailed Example

Definition

The first step checks for the existence of a counter clockwise rotation value, which if exists then rotates p_1 and p_2 around p_1 by angle α once. The bracketed <1> indicates "non-generative", a form of subset generation which omits the original shape effectively moving the transformed shape. The definition for the geometric generation of the rectangle is as follows. ΔT_1 performs the first step where p_1 is transformed in the direction vector from p_1 to ($p_2.x$, $p_1.y$), 0 distance apart and of quantity 1. This effectively sweeps a point to generate a line. The output is a segment which is then transformed by T_2 in a similar fashion in the y direction to create the rectangle. Of note, if *length* and p_1 but not *width*, *area*, nor *aspect ratio* were entered, the preprocessing of the Relationships would indicate that the user would need to either enter a *width* value or a $p_2.y$ value to complete the required inputs. The applies to any other combinations which satisfy the complete solvability for all variables.

Interaction

When the RectanglePattern state is entered into, the interaction sequence begins by setting isFinished to false and then enters into an inactive state. This changes once a mouse down occurs at which point p_1 variable is set to the current mouse location. At this point the system is in an interactive state where if a mouse move occurs p_2 is updated with the current mouse location. If a mouse up or mouse down occurs (mouse up because the mouse button may have never been released after the mouse down for p_1) and p_2 is within a buffer tolerance to avoid too small of a rectangle, then the command becomes inactive again. If still in an interactive state, and the mouse changes in any way and falls outside the drawing canvas, then the pause state is entered into until the mouse re-enters the canvas area. If in the interactive state, and a mouse up or mouse down occurs and is not within the buffer, then isFinished is set to true and the command is completed. Not shown here but common for this type of interaction is an abort feature wherein a right click or escape key would cause an inactive state to become enabled. If isFinished is true then the command is restarted (but not unloaded). If another command is started this command is unloaded.

The dialog box displayed in Figure 6 is automatically generated from the variables in the Relationships section. The checkmark would indicate a fixed value whereas an unchecked value would indicate a free variable which could either be manipulated interactively within the drawing canvas or within the dialog box. Degrees of freedom would have to be calculated to ensure that an

under or over constrained condition does not occur by checking or unchecking too many variables. Points are excluded in this example as it is customary to input them interactively, however it would be possible for them to be included if desired. Such capability could potentially be implemented utilizing a technology like the Extensible Application Markup Language (XAML), a declarative language for describing complex visual user interfaces at runtime. As mentioned previously, the Interaction definitions will not be explored in any more depth than illustrated in the example here, as the primary focus is more on the specification of the Definition and Relationships sections, leaving user interaction as an implementation detail.

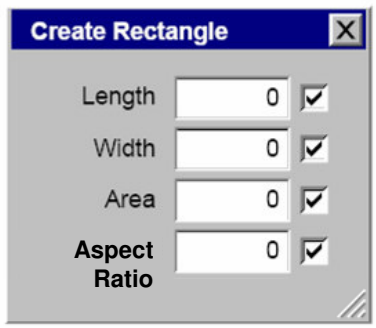


Figure 6: User dialog box automatically generated from Relationships variables.

While the interaction specification may seem unnecessarily complex to create a simple rectangle, the key observation to be made is that graphical user interaction is much more complicated than appears on the surface, and equally complicated to code by hand. Simple geometry creation tools in typical applications hide all of this complexity from users to the point that simple geometry creation appears trivial.

One of the key benefits to this approach is that this specification approach carries a much higher Degree of Structural Information Encoded (DOSIE), a term utilized in database development which gives credit to approaches which maximize expressibility in relation to compactness of specification. In other words, this approach is significantly more compact than the only other available technique of manually encoding such behavior which is an extremely verbose and complicated process. Readers are encouraged to refer to Appendix 9 which provides an example implementation created in raw code, in order to assess the specification required to implement the behavior described here compared to manual coding techniques.

Summary

In summary, the following characteristics are introduced via the n-dimensional abstraction pattern based approach:

1. Shift in problem approach from search to exploration. Rather than relying on a small set of globally or locally optimized parameters, the pattern based approach utilizes contextually relevant parameters and behavioral definitions. The pattern approach is specifically geared towards defining contextually compatible solutions in a flexible manner. It overcomes the common problems associated with implicit, tacit, and emergent constraints and requirements, conditions common and significant to design problems.

2. Strategic progression with localized optimization. The primary end-user utilization of the pattern based approach is envisioned to be as a user-directed process moving strategically through a series of steps to produce a desirable result. Progression leads to predictable changes that the designer expects and has full control over. This subjugates the final evaluation of value of solution quality in design to human evaluation while the computer is responsible for producing the expected result as quickly as possible.

3. Reduction of search space size. This approach trades exhaustive search for localized search, and trades constraint solving for parameter reduction and degree of freedom analysis. This results in more deterministic behavior and the likelihood of polynomial performance.

4. Parameterized solutions can be used to improve existing pure optimization methods. Many optimization routines are very sensitive to their start point. Utilizing a pattern with its default values as a starting point enables the algorithm to narrow the breadth of alternatives to avoid meaningless possibilities, therefore guiding the solver to a more likely solution space. Additionally, design state transformations within optimization routines could be controlled by pattern definitions therefore encouraging more realistic progression through the search space. Transformation of the actual pattern definition could also be employed to mutate the generation mechanism if novel solutions were desired.

5. While not rigorously analyzed, a design solution expressed in this notation (as with all generative and parametric languages) is much more compact than lower-level traditional computational geometry techniques

of persistence, such as storing coordinates of all entities. Instead what is described and persisted are the relationships (geometric and logical) and the rules for maintaining those relationships in a desirable (from a design perspective) manner.

4 Pattern Catalog

Explanation

This section demonstrates patterns which are utilized in numerous military designs. Some of them are foundational enough that they are applicable to most designs, military or otherwise.

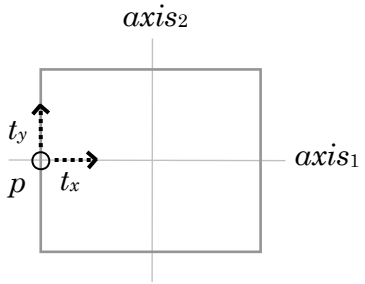
Container

Signature:

Container [p_1, *t, s,* [*z, a, l, d, o*]]

Also known as:

C, shape, s



C [p_1, $p_1 \rightarrow (5,0)$, *rect*]

Description:

The Container is the basis for many subsequent patterns defined here. Containers can contain containers as its contents. The advantage of the Container abstraction is that it simplifies the concept of hierarchy with the intuitive notion of items being contained within other items. The container can have items dropped into it and it will update the arrangement of its contents. In many ways the Container is analogous to text formatting familiar to most modern word processors wherein shapes could be specified to be left justified, full justified, or right justified, and can be extended to include concepts such as wrapping and others. The Containment approach encourages top down and bottom up modification wherein if the user modifies the container geometry, its contents are updated, whereas the opposite is also true where if the contents are modified the parent containers and its parent containers on up are updated. For clarification, vertical justification refers to shape's location in relation to axes, not where the origin is on the shape; so, vertical align top means shape is above axes and its origin is on the bottom of the shape if container is sized to fit by contents or to the top of the container if sized by container.

Required Inputs:

p_1 = origin

t = direction

s = shape/s contained

Optional Inputs:

z = container size/shape options

[size: z_f = **fit container to contents** | z_{sh} = container shape as override to dimensions derived from contents + (z_s = scale contents to fit container | z_{co} = clip contents to container | z_{cc} = clip container to contents)],

a = axis options

[rotation: ar_o = **at origin** | ar_p = angle around point | ar_a = angle around origin],

l = contained shape/s alignment on axis options

[rotation: lr_i = **axis** | lr_a = angle],

[horizontal justification: lj_l = **left** | lj_r = right | lj_c = center | lj_f = full],
[vertical justification:
// align top/center/bottom of shape to axis, or if full to fill the container along
// secondary axis
lj_t = top | lj_c = **center** | lj_b = bottom | lj_f = full,
// align all shapes to the top/bottom of shape; options include max for topmost or
// bottom most point on shape, left/right/top/bottom for smallest/largest y
// obtained from sweepline swept axially from the left/right/top/bottom, or center
// for projection from center of shape axially along shape to top or bottom
lj_{tops} = (shape, options) , $lj_{bottoms}$ = (shape, options)],
d = display options
[display boundary with offset: d_o = **0**],
[display container: d_{cr} = y | ***n***],
[display contained: d_{cd} = ***y*** | n | *displayFaintly*],
o = start/end/top/bottom content offset a distance from the container boundary out
[offset x start: ox_s = **0**, offset x end: ox_e = **0**],
[offset y top: oy_t = **0**, offset y bottom: oy_b = **0**]

Definition:

// 1. rotate origin; recall that points are actually coordinate sytems and anything placed
// using this point as a reference will be placed within that coordinate system
$p_1 = \Delta R^0 [\{ (p_1, p_1 \rightarrow t, 1 \} (p_1)^{<1>}]$

// 2. create axis
// offset axis start pt
if $ox_s \neq \varnothing$ $axisStPt = \Delta T_1 [\{ p_1, p_1 \rightarrow (p_1.x – ox_s, p_1.y), 0, 1 \} (p_1)^{<1>}]$
// offset axis end pt
if $ox_e \neq \varnothing$ $axisEndPt = \Delta T_1 [\{ p_1, p_1 \rightarrow (t.x + ox_e, p_1.y), 0, 1 \} (p_1)^{<1>}]$
// create primary axis from axis st to axis end pts
$axisPrimary = \Delta T_1 [\{ axisStPt, axisStPt \rightarrow axisEndPt, 0, 1 \} (p_1)]$
// create secondary axis by rotating primary axis about axis center
$axisSecondary = \Delta R^0 [\{ (axisEndPt – axisStPt)/2 , -90, 1 \} (axisPrimary)]$
$axes = axisPrimary \wedge axisSecondary$
// rotate primary and secondary axis additionally as necessary
if $ar_p \neq \varnothing$
// first move axis center to rotation point
$axes = \Delta T_1 [\{axisPrimary.midPt, axisStPt \rightarrow ar_p.point, 0, 1\}(axes)^{<1>}]$
// then rotate about point
$axes = \Delta R^0 [\{ ar_p.point, ar_p.angle, 1 \} (axes)^{<1>}]$
else if $ar_a \neq \varnothing$
// rotate about origin
$axes = \Delta R^0 [\{ p_1, ar_a, 1 \} (axes)^{<1>}]$

// 3. establish container/contained relationship between a boundary and its
// contained shapes; constraint initializes variables and tracks and triggers changes
containmentRelationship = ΨH [{ } (*boundary*, *cs*)]

// 4. handle container or constituent changes on change or on first time through
if Δ *cs* or *containmentRelationship* = ∅
    // constrain shapes along axis, using justification and rotation options
    cs = ΦA² [{ *primaryAxis.axisStartPt,*
        primaryAxis.startPt → *primaryAxis.axisEndPt, lj* + *boundary* } (*s*)]

    // set rotation of each shape on axis
    $cs = \sum_{i=1}^{cs.n} \Delta R^0 [\{ cs[i].\text{origin}, lr_a, 1 \} (cs[i])^{<1>}]$
    // trigger update of boundary shape
    Δ *boundary*
else if Δ *boundary*
    // depending on container size/shape options,
    // size and locate the container boundary and contents
    if $z_f \neq \varnothing$
        // creates tight boundary about constrained shapes
        boundary = ϕB² [{ 0 } (∅, *cs*)]
    else if $z_{sh} \neq \varnothing$
        if $z_s \neq \varnothing$
            xscale = *boundary.length* / *cs.bounds.x*
            yscale = *boundary.height* / *cs.bounds.y*
            // scale constrained shapes about origin
            cs = ΔD⁰ [{ *cs.origin*, (*xscale*, *yscale*), 1 } (*cs*) $^{<1>}$]
        // if clip contents from boundary
        else if $z_{co} \neq \varnothing$
            cs = ΩD[{ }(*cs* – *boundary*) $^{<1>}$]
        // if clip boundary to contents
        else if $z_{cc} \neq \varnothing$
            z_{sh} = ΩD[{ }(*boundary* – *cs*) $^{<1>}$]
        // create updated boundary
        boundary = ϕB² [{ 0 } (z_{sh})]

// set boundary offset
// the difference between display offset and content offset is that
// shape sizes and orientation are not affected in display offset;
// it is meant as a visual aid for viewing containment relationships

if $d_o \neq \varnothing$

$boundary_2 = \Xi\ \mathrm{E}\ [\{\ o, d_o\ \}\ (\ boundary\ )\ ]$

// set boundary display options

// if offset boundary exists, display it instead if boundary is to be displayed

if $boundary_2 \neq \varnothing$

$boundary_2 = \phi\mathrm{Q}\ [\ \{\ display, d_{cr}\ \}\ (\ boundary_2\ )\ ]$

$boundary = \phi\mathrm{Q}\ [\ \{\ display, no\ \}\ (\ boundary\ )\ ]$

else

$boundary = \phi\mathrm{Q}\ [\ \{\ display, d_{cr}\ \}\ (\ boundary\ )\ ]$

// set contained elements display options

$cs = \phi\mathrm{Q}\ [\ \{\ cs.\mathrm{display}, d_{cd}\ \}\ (\ cs\ )\ ]$

$Container = boundary \wedge cs \wedge axes$

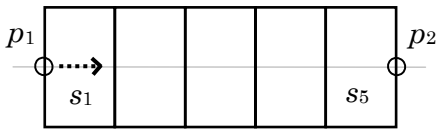
Relationships:

$t == p_1 \rightarrow p_2$

LinearSequence

Signature:

LinearSequence [p_1, t, d, n, s, [c, z, a, l, d_1, o]]



LS [p_1, $p_1 \rightarrow p_2$, 0, 5, *rect*]

Also known as:

LS, Row

Description:

A LinearSequence positions each shape_{i+1} after each shape_i. It also supports rotation of each contained shape which is different than rotation of the overall LinearSequence. A LinearSequence can arrange any types of shapes in sequence, and is akin to an ordered list data structure utilized in computer science. Interaction definition would include behavior for being able to drag and drop elements within the sequence to change order as well as to insert and remove elements from the sequence.

Required Inputs:

p_1 = origin
t = direction
d = distance apart [min–max | value]
n = quantity of shapes
s = shape/s

Optional Inputs:

c = align contained [align shapes contained in s: $c = y \mid \boldsymbol{n}$],
See Container as remaining optional inputs here are derived from Container optional inputs.

Definition:

// if there is more than one shape in s, optionally constrain offset between shapes
if $c = y$ $s = \Xi$ E [{ *origin,* ϕJ^0 [{ $d_{\min}$, $d_{\max}$, $\varnothing$ } (s) $^{<1>}$] } (s)]

// make n number of shapes, constraining offset between shapes;
// it is possible to create an array of arrays of shapes with this approach

$$contained = \phi J^0 [\{ d_{\min}, d_{\max}, \varnothing \} (\sum_{i=1}^{n} s)^{<1>}]$$

// take resultant shapes and place in a container
// this allows interactive insertion within the sequence as well as along the sequence
LinearSequence = Container [p_1, *t, contained, z, a, l,* d_1, *o*]

Relationships:

$t == p_1 \rightarrow p_2$

// the length of this container is the distance between the two points

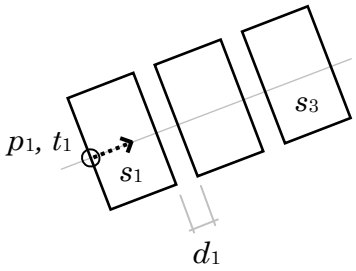
$length == p_2 - p_1$

// or it can be derived from it's content's length, quantity, intershape distance,

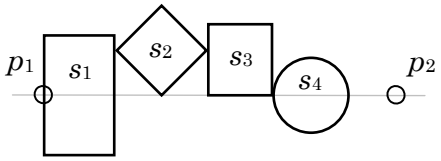
// and the start and end offsets

$length == (n * s.axisPrimary.length) + (n * (d\text{-}1)) + (ox_s + ox_e)$

Examples:



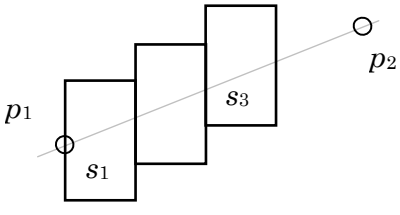
LS [p_1, t_1, d_1, 3, *rect*]



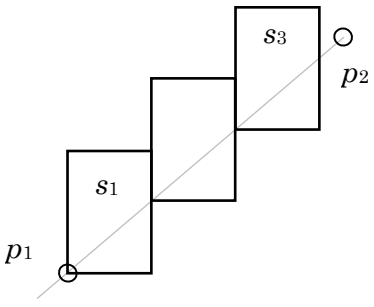
shapes = LS [p_1, $p_1 \rightarrow p_2$, 0, 1, (*rect, diamond, square, circle*)]

// use exception regulator to reassign alignment attributes

Ξ E [{ *l, ljb* } (*shapes*[2], *shapes*[3])]



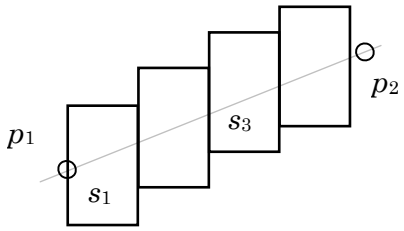
LS [p_1, $p_1 \rightarrow p_2$, 0, 3, *rect*, lr_a=0]



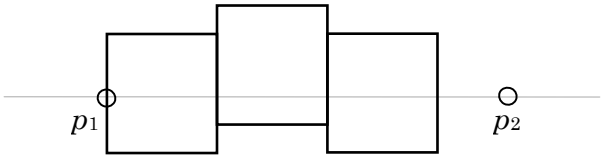
// three shapes to fit within p_1 and p_2

// alignment to vertical bottom

LS [p_1, $p_1 \rightarrow p_2$, 0, 3, *rect*, lj_b, lj_f, lr_a=0]



// as many of rect that will fit within p_1 and p_2
// recall that p_1 and p_2 can be edited interactively
LS [p_1, $p_1 \rightarrow p_2$, 0, 0 – ∞, *rect*, lr_a=0]



Pattern used in FY06 Fort Lewis Barracks design.
// create linear sequence
$bldg_1$ = LS [p_1, $p_1 \rightarrow p_2$, 0, 3, *barracksModule*]

// move center shape distance *d*
$bldg_1$.shape[2] = ΔT [{ p_1, $p_1.y + d$ } ($bldg_1$.shape[2])] [1]

DoubleLoadedCorridor

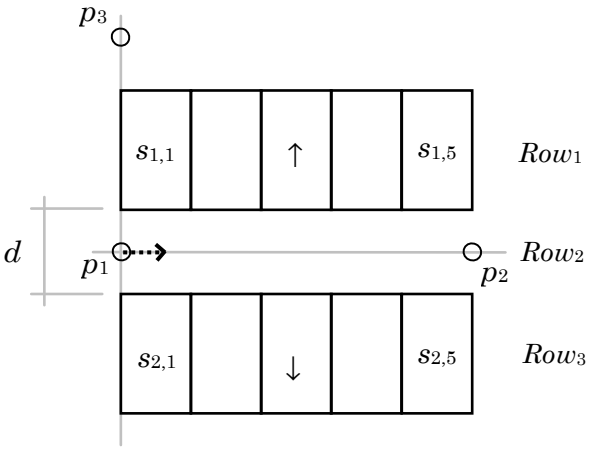
Signature:

DoubleLoadedCorridor [p_1, t, d, LS_1, LS_2, [p_3, c, m]]

SingleLoadedCorridor [p_1, t, d, LS_1, [p_3, c, m]]

Also known as:

RowOfRows, RR, DLC; SLC



LS_1, LS_2 = LS [p, t, d, 5, *rect*]

RR [p_1, $p_1 \rightarrow p_2$, d, LS_1, LS_2, $c=c_n$, $m=m_y$]

Description:

The double loaded corridor is a commonly utilized topological circulation pattern, examples of which can be found in most buildings. In involves a circulation function flanked on either side by another series of functional spaces. Its common use stems from the inherent circulation efficiency achieved by maximizing the ratio of usable functional space to circulation space. The double loaded corridor can be generalized as a linear sequence of linear sequences where the rows of spaces are organized in a sequence with a corridor in between. A common option within this pattern includes fitting a fixed number of spaces with variable length within an overall distance. Each row of spaces can be referenced and transformed independently to create variation within this pattern. This can enable rotation or a repeating sequence of arrangements such as A B B C B B A. A single loaded corridor is a derivative where one of the row of spaces is omitted.

Required Inputs:

p_1 = origin

t = direction

d = distance apart

LS = LinearSequence

Optional Inputs:

p_3 = secondary axis direction from p_1

c = corridor options

[display corridor: c_y = **yes** | c_n = no],

m = mirror option

[mirror option: m_y = mirror LS_1 to create LS_2 | m_n = **don't mirror**]

Definition:

// align origin with t

$p_1 = \Delta R^0 [\{ (p_1, p_1 \rightarrow t, 1 \} (p_1)^{<1>}]$

// put LS_1 and LS_2 in containers; one reason being to change the LSs' p_1 and t to
// incoming p_1 and t and secondly to allow for future prepending and appending
// of shapes within the corridors and thirdly to top and bottom align the LSs
// without changing their original alignments

$Row_1 = \text{LS} [p_1, t, 0, 1, LS_1, lj_t]$

$Row_3 = \text{LS} [p_1, t, 0, 1, LS_2, lj_b]$

// if mirror option specified

if $m_y \neq \varnothing$

// if no Row_3 then mirror Row_1 to create Row_3

if $Row_3 = \varnothing \quad Row_3 = \Delta M^1 [\{ Row_1.p_1, Row_1.p_1 \rightarrow Row_1.p_2, 1 \} (Row_1)^{<1>}]$

// else mirror Row_3 about itself

else $Row_3 = \Delta M^1 [\{ Row_3.p_1, Row_3.p_1 \rightarrow Row_3.p_2, 1\} (Row_3)^{<1>}]$

// Determine angle between primary and rotated axes

$angle = | p_1,p_2 | \sim | p_1,p_3 |$

// set rotation of shapes in Row_1 and Row_3

$Row_1, Row_3 = \Xi \text{ E} [\{ lr_a, angle \} (Row_1, Row_3)]$

// calculate offsets to accommodate rotated axis by rotating origin, moving
// origin half of offset distance and then determining its distance to the original axis

$p_{temp} = \Delta R^0 [\{ p_1, \text{angle}, 1 \} (p_1)^{<1>}]$

$p_{temp} = \Delta T_1 [\{ p_{temp}, p_{temp} \rightarrow (p_{temp}.x, p_{temp}.y + d/2), 0, 1 \} (p_{temp})^{<1>}]$

// get distance from secondary orthogonal axis to point,
// which will return a positive or negative distance

$offset = | (p_1.x - \infty, p_1.y), (p_1.x + \infty, p_1.y) | - p_{temp}$

// set start and end offsets, respecting reversed distance offsets for upper and lower;
// start offset is negative in order to get inset instead of offset

$Row_1 = \Xi \text{ E} [\{ (ox_s, ox_e), (- offset, offset) \} (Row_1)]$

$Row_3 = \Xi \text{ E} [\{ (ox_s, ox_e), (offset, - offset) \} (Row_3)]$

// rotate entire Row about its origin to orient it correctly
// in the linear sequence it is about to be contained within

$Row_1, Row_3 = \Delta R^0 [\{ (Row_1.p_1, Row_3.p_1), 90, 1 \} (Row_1, Row_3)^{<1>}]$

// create linear sequence of Row 1, 2, and 3 using Row_1, d, and Row_3
// in order to create the corridor shape
LS_{tmp} = LS [p_1, t, d, 1, (Row_1, Row_3), c]

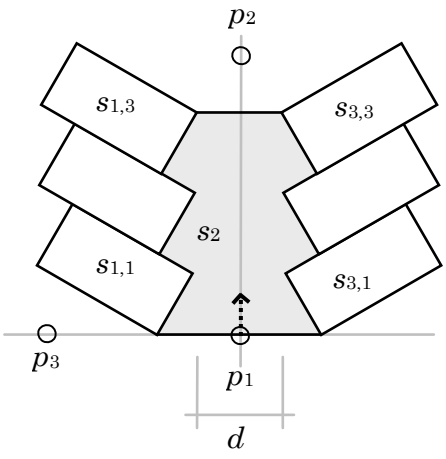
// create corridor shape by applying symmetric difference of rows from boundary
if $c_y \neq \varnothing$
    Row_2 = ΩM [{ } ($LS_{tmp}.boundary$ – ($LS_{tmp}.Row_1 \wedge LS_{tmp}.Row_3$))]
    // if corridor is created then it is included and offset distance is 0
    // the approach is that the double loaded corridor is a vertical row of horizontal rows
    $DoubleLoadedCorridor$ = LS [p_1, t, 0, 1, ($LS_{tmp}.Row_1$, Row_2, $LS_{tmp}.Row_3$), c=y]
else
    $DoubleLoadedCorridor = LS_{tmp}$

// rotate entire container back to realign corridor axial length with the primary axis
$DoubleLoadedCorridor$ = ΔR^0 [{ (p_1), -90, 1 } ($DoubleLoadedCorridor$) $^{<1>}$]

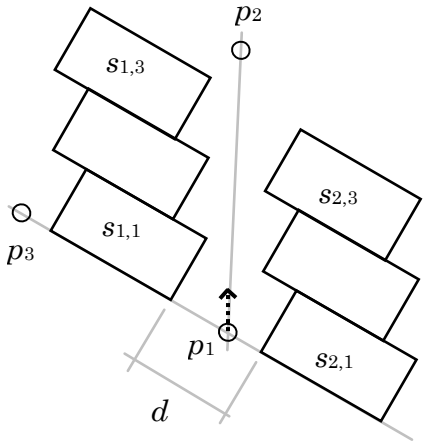
Relationships:

$t == p_1 \rightarrow p_2$

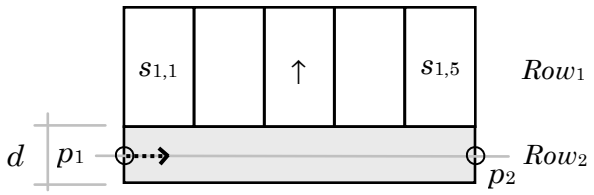
Examples:



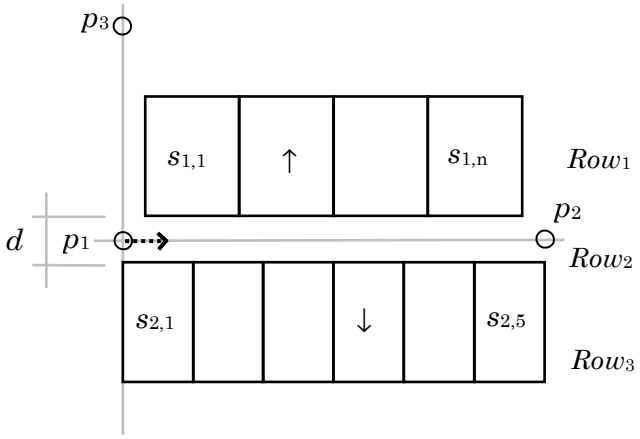
LS_1 = LS [p, t, d, 3, $rect$, l_{ra}= α]
RR [p_1, $p_1 \rightarrow p_2$, d, LS_1, m=m_y]



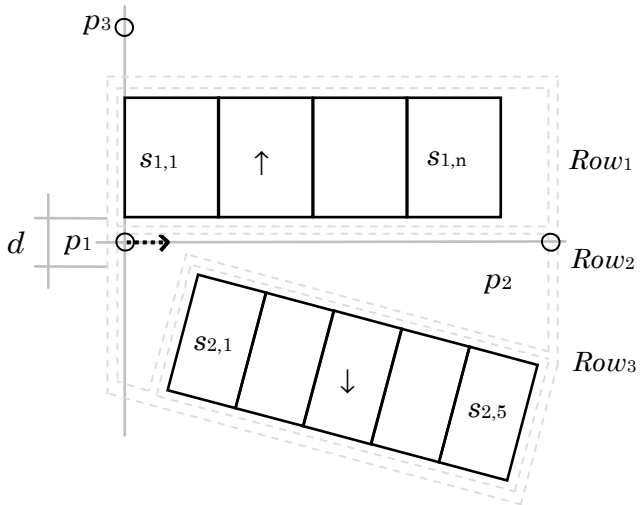
LS_1, LS_2 = LS [p, t, d, 3, $rect$]
RR [p_1, $p_1 \rightarrow p_2$, d, LS_1, LS_2 p_3=p_3, c=c_n]



LS_1 = LS [*p, t, d,* 5, *rect*]
RR [p_1, $p_1 \rightarrow p_2$, *d*, LS_1, ∅]



LS_1 = LS [*p, t, d,* 3 – ∞, $rect_1$, $l=lj_c$]
LS_2 = LS [*p, t, d,* 5, *rect*, ox_s=*offset*, $l=lj_r$]
RR_1 = RR [p_1, $p_1 \rightarrow p_2$, *d*, LS_1, *LS*2, $c=c_n$, $m=m_y$]



LS_1 = LS [*p, t, d,* 3 – ∞, $rect_1$]
LS_2 = LS [*p, t, d,* 5, $rect_2$, ox_s=*offset*, $l=lj_r$]
RR_1 = RR [p_1, $p_1 \rightarrow p_2$, *d*, LS_1, *LS*2, $c=c_n$, $m=m_y$]
$RR_1.container.d_{cr} = y$

TConfig

Signature:

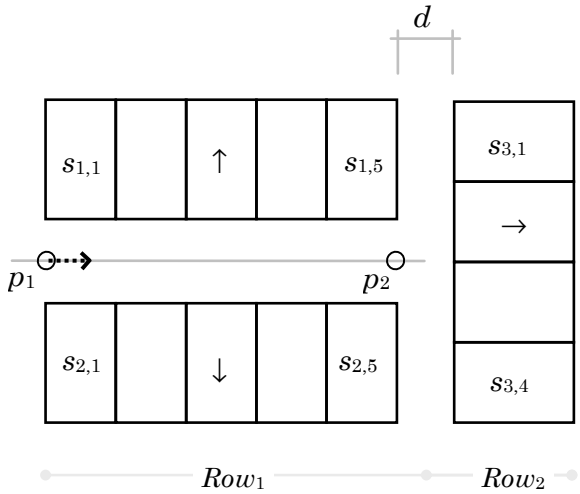
TConfig [p_1, t, DLC_1, d, LS_1, [c, l]]

Also known as:

UConfig, TC, T

Description:

The T configuration is a common circulation pattern. It involves a double loaded corridor with a linear sequence aligned perpendicularly. Corridor widths between the double loaded corridor and the linear sequence can be different. This configuration is not much different than stacking a row of rows where the first row is a double loaded corridor and the second is a rotated single loaded corridor, with the exception that this pattern places the double loaded corridor's corridor width in relation to the difference remaining between the height of the overall linear sequence and the widths of the rows in the double loaded corridor. Vertical alignment in this case refers to Row_2's location in relation to Row_1 rather than in relation to the axis.



TConfig [p_1, $p_1 \rightarrow p_2$, DLC_1, d, $LinearSequence_1$]

Required Inputs:

p_1 = origin
t = direction
d = corridor width
DLC_1 = DoubleLoadedCorridor
LS_1 = LinearSequence

Optional Inputs:

c = corridor options
[display corridor: c_y = **yes** | c_n = no],
l = single loaded corridor's alignment/justification
// align single loaded corridor to the left or right of the double loaded corridor
[horizontal alignment: lj_l = left of DLC | lj_r = **right of DLC**]
// align single loaded corridor in relation to the double loaded corridor
[vertical justification: lj_t = top | lj_c = **center** | lj_b = bottom | lj_f = full]

Definition:

// align origin with t
$p_1 = \Delta R^0 [\{ (p_1, p_1 \rightarrow t, 1 \} (p_1)^{<1>}]$

// rotate LS_1 to be perpendicular to DLC_1 taking left/right side into account
if $l = l_{jr}$ *rotation* = -90 else *rotation* = 90
$LS_1 = \Delta R^0 [\{ (p_1, \text{rotation}, 1 \} (LS_1)^{<1>}]$
// turn LS_1 from a linear sequence into a single loaded corridor
$LS_1 = S\text{LC} [LS_1.p_1, LS_1.p_1 \rightarrow LS_1.p_2, d, LS_1]$

// put DLC_1 and LS_1 in containers; one reason being to change the alignments
// and secondly to allow for future prepending and appending of shapes
$Row_1 = \text{LS} [p_1, t, 0, 1, DLC_1, lj_l]$
$Row_2 = \text{LS} [p_1, t, 0, 1, LS_1, lj_r]$

// sets alignment with consideration for rotated linear sequence;
// note that container's alignment options takes a sweepline along the
// axis to determine rightmost or leftmost y point to align to
if $l = lj_t$ then $l = lj_{tops}$ (Row1) else if $l = lj_b$ then $l = lj_{bottoms}$ (Row1)

// determine which row comes first depending on left/right
if $l = l_{jr}$ r*ows* = (Row_1, Row_2)
else r*ows* = (Row_2, Row_1)

// create linear sequence of Row_1 and Row_2
$TConfig = \text{LS} [p_1, t, 0, 1, rows, l]$

// union the two corridors
$TConfig = \Xi\ \text{E} [\{ (TConfig.Row_1.Row_2, TConfig.Row_2.Row_2),$
$\Omega\text{D}[\{ \} (TConfig.Row_1.Row_2 - TConfig.Row_2.Row_2)^{<1>}] \} (TConfig)]$

if c = no
TConfig = ϕQ [{ ($TConfig.Row_1.Row_2.display$, $TConfig.Row_2.Row_2.display$), *no* } (*TConfig*)]

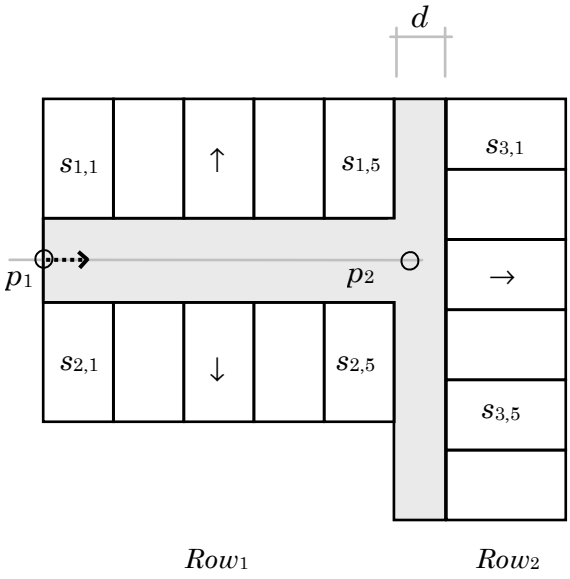
Relationships:

$t == p_1 \rightarrow p_2$

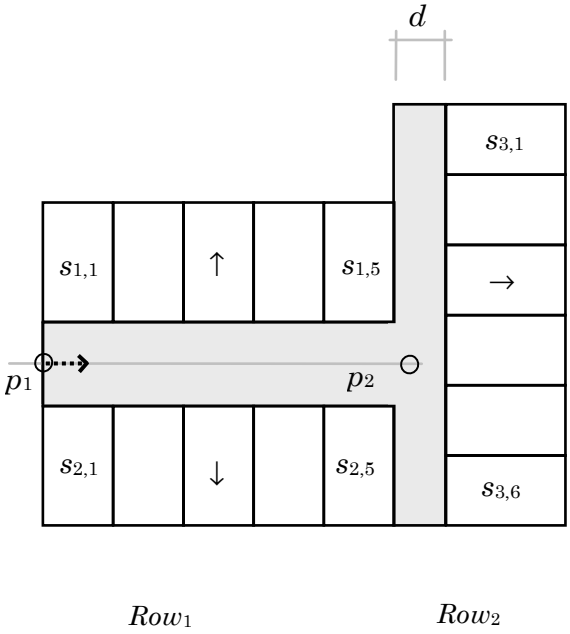
// double loaded corridor width can be variable is in relation to remaining distance
// derived from LS height and Row widths
$DLC_1.d == LS_1.y - DLC.Row_1.x + DLC.Row_3.x$

// single loaded corridor width can also be derived from remaining
// distance between p1 and p2 and LS height and Row widths
$LS_1.d == |p2 - p1| - LS_1.y - DLC.x$

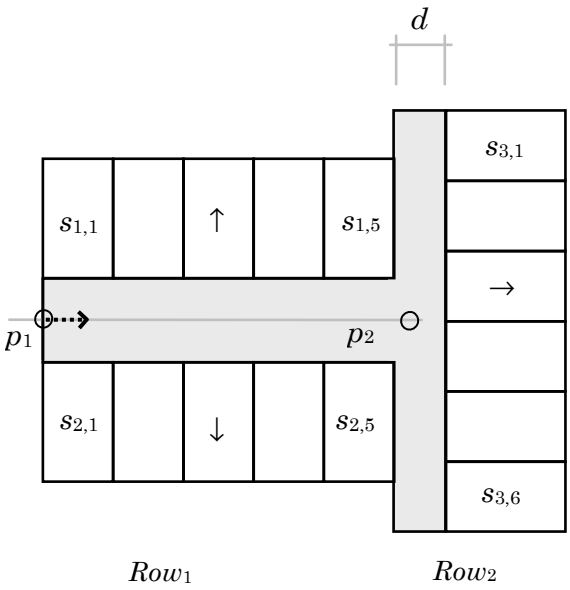
Examples:



TConfig [p_1, $p_1 \rightarrow p_2$, DLC_1, d, $LinearSequence_1$, $l = lj_t$]



TConfig [p_1, $p_1 \rightarrow p_2$, DLC_1, d, $LinearSequence_1$, $l = lj_b$]



TConfig [p_1, $p_1 \rightarrow p_2$, DLC_1, d, $LinearSequence_1$, $l = lj_c$]

LConfig.DLC2SLC

Signature:

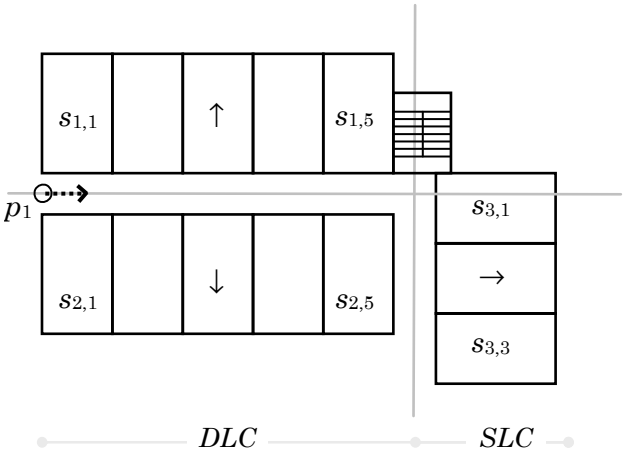
LConfig.DLC2SLC [p_1, *t, DLC*, *st, LS,* [*c, o*$_f$*, s, l, d*]]

Also known as:

LC, L

Description:

The L configuration is different than the T configuration in that it places a stairwell at the corner intersection between the double loaded corridor and a single loaded corridor. It also ensures that the space adjacent to the stairs on the linear sequence side is wide enough for access.



LConfig.DLC2SLC [p_1, *t, DLC*$_1$, *st*, LS$_1$, c=no, s_v]

Required Inputs:

p_1 = origin

t = direction

DLC = DoubleLoadedCorridor

LS = LinearSequence

st = stairs

Optional Inputs:

c = corridor options

[display corridor: c_y = **yes** | c_n = no]

o_f = minimum single loaded corridor width

[offset: o_f = **4**]

s = stair attributes

[stair location: s_d = end of DLC | s_s = **end of SLC**, s_h = **horizontal** | s_v = vertical]

l = alignment

// align single loaded corridor to the left or right of the double loaded corridor

[horizontal alignment: l_l = left of DLC | l_r = **right of DLC**]

d = direction

[SLC direction: d_l = left turning | d_r = **right turning**]

Definition:

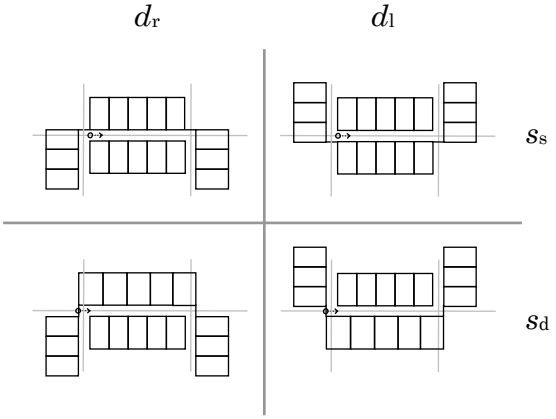
// align origin with t

$p_1 = \Delta R^0 [\{ (p_1, p_1 \rightarrow t, 1 \} (p_1)^{<1>}]$

// if DLC corridor width is greater than minimum offset, use it
if DLC.d > o_f width = DLC.d else width = o_f

// create TConfig
tc = TConfig [p_1, t, DLC, width, LS, c, l]

// if stairs at end of DLC then adjust DLC to extend to end of SLC corridor
if $s_d + d_r$ $rowToModify = tc.Row_1.Row_1$
if $s_d + d_l$ $rowToModify = tc.Row_1.Row_3$
$tc = \Xi$ E [{ (ox_s, ox_e), $tc.Row_2.LS.width$ } ($rowToModify$)]



// modify TConfig to align LS with DLC corridor
// right cases then left cases illustrated in margin
if $s_s + d_r$ or $s_d + d_l$
$tc = \Xi$ E [{ $origin$, $tc.Row_1.Row_2.top$ } ($tc.Row_2.LS$)]
else // $s_d + d_r$ or $s_s + d_l$ case
$tc = \Xi$ E [{ $origin$, $tc.Row_1.Row_2.bottom$ } ($tc.Row_2.LS$)]

// locate and orient stairs in relation to corridor
if $s_s + d_l$ or $s_d + d_r$
$st = \Delta T_1$ [{ p_1, $p_1 \rightarrow tc.Row_1.Row_2.bottom$, 0, 1 } (st) $^{<1>}$]
else // $s_s + d_r$ or $s_d + d_l$ cases
$st = \Delta T_1$ [{ p_1, $p_1 \rightarrow tc.Row_1.Row_2.top$, 0, 1 } (st) $^{<1>}$]

// move stairs to correct side
if l_r $st = \Delta T_1$ [{ $st.p_1$, $st.p_1 \rightarrow (tc.Row_1.Row_2.p_2.x, st.p_1.y)$, 0, 1 } (st) $^{<1>}$]

// rotate stairs accordingly
if d_r
if l_l
if s_v $st = \Delta T_1$ [{ $st.p_1$, $st.p_1 \rightarrow$ ($st.p_1$.x – $st.width$, $st.p_1$.y), 0, 1 } (st) $^{<1>}$]
else if s_h $st = \Delta R^0$ [{ ($st.p_1$, 90, 1 } (st) $^{<1>}$]
if l_r
// if s_v do nothing
if s_h
$st = \Delta T_1$ [{ $st.p_1$, $st.p_1 \rightarrow$ ($st.p_1$.x – $st.width$, $st.p_1$.y), 0, 1 } (st) $^{<1>}$]
$st = \Delta R^0$ [{ ($st.p_1$, -90, 1 } (st) $^{<1>}$]
else // d_l case
if l_l
if s_v $st = \Delta R^0$ [{ ($st.p_1$, 180, 1 } (st) $^{<1>}$]
else if s_h
$st = \Delta T_1$ [{ $st.p_1$, $st.p_1 \rightarrow$ ($st.p_1$.x – $st.width$, $st.p_1$.y), 0, 1 } (st) $^{<1>}$]

$st = \Delta R^0 [\{ (st.p_1, 90, 1 \} (st)^{<1>}]$

if l_r

if s_v

$st = \Delta T_1 [\{ st.p_1, st.p_1 \rightarrow (st.p_1.x - st.width, st.p_1.y), 0, 1 \} (st)^{<1>}]$

$st = \Delta R^0 [\{ (st.p_1, 90, 1 \} (st)^{<1>}]$

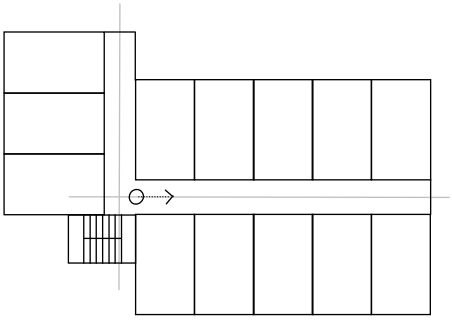
else if s_h $st = \Delta R^0 [\{ (st.p_1, -90, 1 \} (st)^{<1>}]$

$LConfig = tc \wedge st$

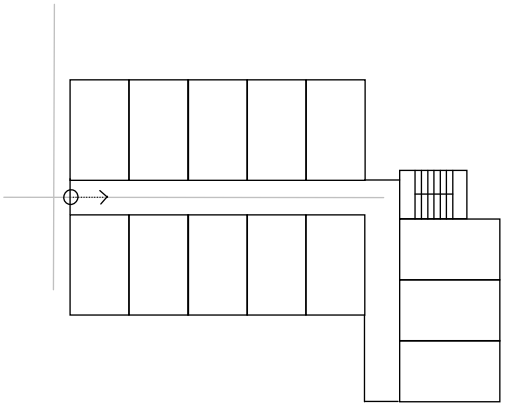
Relationships:

$t == p_1 \rightarrow p_2$

Examples:



LConfig.DLC2SLC [p_1, t, DLC_1, st, LS_1, $s_{s+}s_{h,}$ $l_{l,}$ d_l]



LConfig.DLC2SLC [p_1, t, DLC_1, st, LS_1, $s_{d+}s_{h,}$ $l_{r,}$ d_r]

5 Pattern Analysis

Introduction

This section is a follow on to the more detailed pattern definitions described in Chapter 4 with a more general analysis of other patterns which were identified in military contexts or are predicted would be of value within a military context. Examples of these patterns were distilled from and can be seen within the layouts in Section 7.

Patterns Discussed

BooleanContainer
LConfig.DLC2DLC
UConfig (also: C, L, T, H, O, and Y)
RacetrackCorridor
CornerCondition

BooleanContainer

The BooleanContainer pattern is a response to the commonly observed procedure of including one shape within the context of other shapes such that the end result consists of a Boolean subtraction operation between a combination of the shapes. For example, in Figure 7 a potential use case example would include the following sequence. s_1 and s_2 are created within a BooleanContainer as regular, adjacent rectangles (interactively or otherwise). s_3 is then placed within the container on top of the two rectangles. Then, when s_3 requires modification, s_1 and s_2 automatically reapply the Boolean operation on their original definitions to create updated versions of themselves.

This pattern reflects a common user interaction sequence which could be automated saving numerous steps over manual techniques. Again, while apparently rather simple conceptually and one would think simple to achieve on a computer, currently there is no system which is known of which can support a designer in this manner. The pattern approach would allow a designer to define

and interactively modify this configuration in real time (30+ solutions per second).

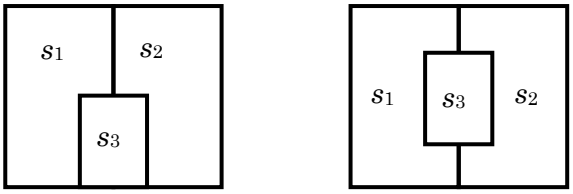


Figure 7: BooleanContainer examples.

An inventory of steps required to create the Boolean configuration manually:

1. initiate rectangle command
2. select s1 origin
3. keyin s1 length/width (or pick p2)
4. select s1
5. select copy command
6. select origin
7. select destination
8. initiate rectangle command
9. tentative snap to s1/s2 shared corner
10. move cursor ½ distance of desired s3 width
11. click to accept s3 origin
12. keyin s3 length/width
13. initiate Boolean subtract command
14. select s1 and s2, accept "elements to be subtracted" selection
15. select s3, accept "subtracting element" selection

With regards to modification, the left case in Figure 7 presents an interesting topological condition which does not present a straightforward method to accomplish. A more formal usability analysis utilizing a tool would be required to determine a more accurate average for this case. More than likely the user would delete and recreate the elements in this case.

An inventory of steps required to modify the right Boolean configuration manually:

1. initiate fence stretch/modify command
2. select origin of fence around s3
3. select p2 of fence around s3
4. select origin of move origin
5. select p2 of destination

Note: this set of modification steps only achieves a subset of desirable transformations, which illustrates that the modification of this configuration in a manual sense is very difficult and time consuming (if even possible according to the desired intent), with recreation from scratch very likely.

Assuming that the BooleanContainer pattern was defined, tested, documented, enhanced, and troubleshooted, an inventory of steps required to create the BooleanContainer would be:

1. instantiation of the BooleanContainer pattern
2. keyin length of s1/s2
3. keyin width of s1/s2
4. select origin of s1/s2
5. keyin length of s3
6. keyin width of s3
7. select origin of s3

An example inventory of steps required to modify the BooleanContainer would be:

1. drag s3 from origin to destination

LConfig.DLC2DLC

The LConfig.DLC2DLC pattern is a variant of the LConfig.DLC2SLC in that it adjoins two double loaded corridors together instead of a double loaded corridor and a single loaded corridor. Some interesting issues to note are 1) the increase of the end space size on the outside edge to provide a more regularized overall building perimeter, 2) the increase of the end space size on the inside edge to provide access to the exterior, 3) the alignment and modularity of the spaces horizontally and vertically, and 4) the substitution of one of the modules for a pair of adjacent modules, commonly utilized as utility and storage rooms.

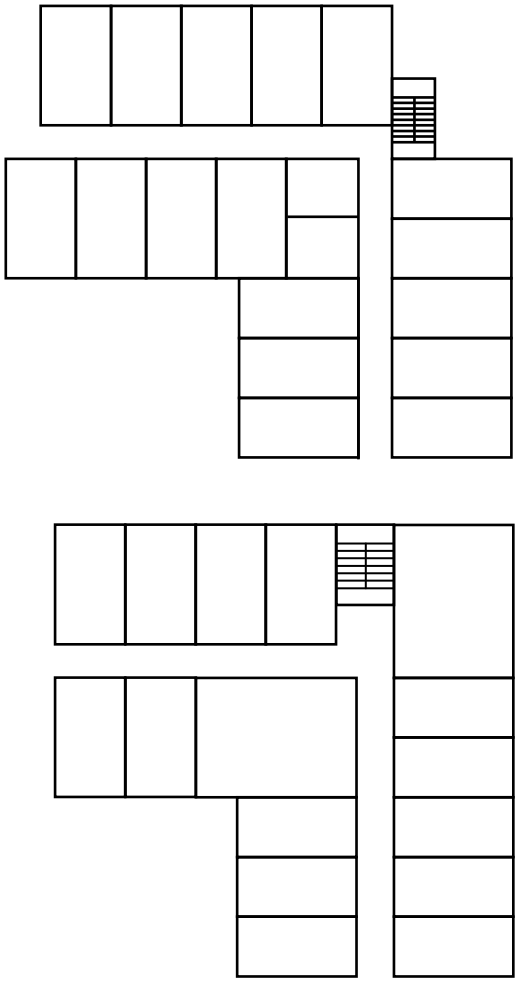


Figure 8: LConfig.DLLC2DLC examples.

UConfig

The U Configuration (or UConfig) pattern is another alphabetically associated geometric configuration in the family of C, L, T, H, O, and Y, types of patterns. Treatment of this morphological concept has been described by a number of researchers, for example by Ching (Ching, 1996), however there are no known algorithmic descriptions of how these could be generated and modified. As illustrated in Figure 9, the UConfig can be thought of as having a top row that extends across two rows which run perpendicular, and which have a circulation channel that connect them. Figure 10 illustrates an instance of the pattern configuration as was utilized in the basic module for the Ft. Lewis FY06 Barracks.

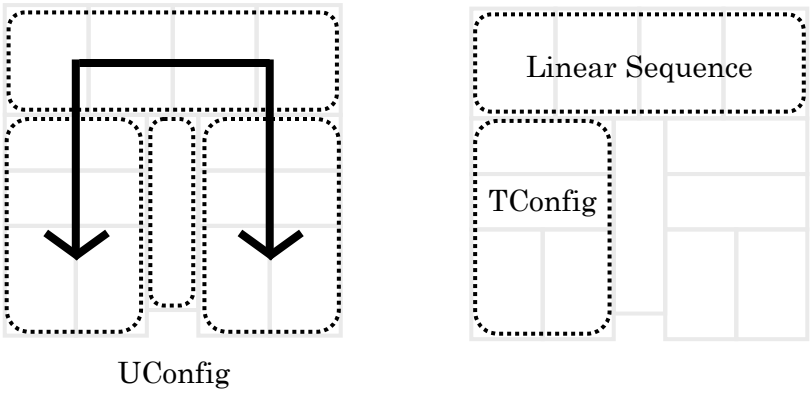


Figure 9: UConfig pattern examples.

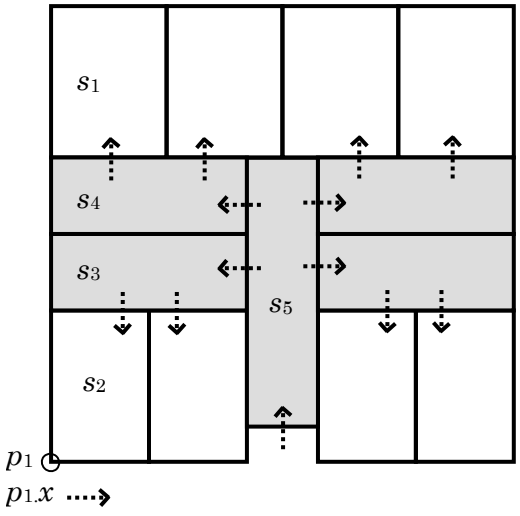


Figure 10: Ft. Lewis FY06 Barracks module configuration as instantiation of UConfig.

RacetrackCorridor

The racetrack corridor is a pattern which consists of an outer set of functions which wrap around an inner set of functions. This circulation pattern introduces the efficiency of the double loaded corridor into a continuous building layout. Two variants exist, one in which the outer functions are stacked inside or outside of each other, and the second in which the outer functions are interlocked. In facility types that require daylighting, the inner function can be utilized as a courtyard, or as a set of functions which encase a courtyard.

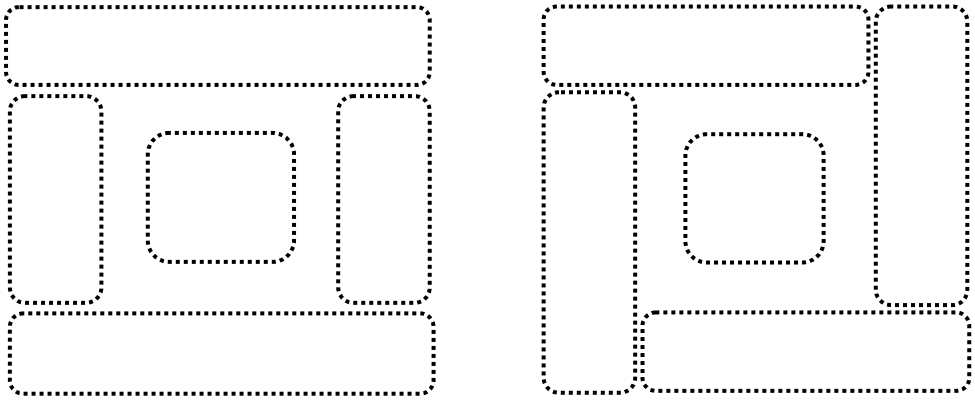


Figure 11: Racetrack corridor pattern examples

Of particular concern with this pattern is the impact of change due to circular dependencies. Different strategies such as conditional existence, quantity, and size variations can all be used as stopping conditions.

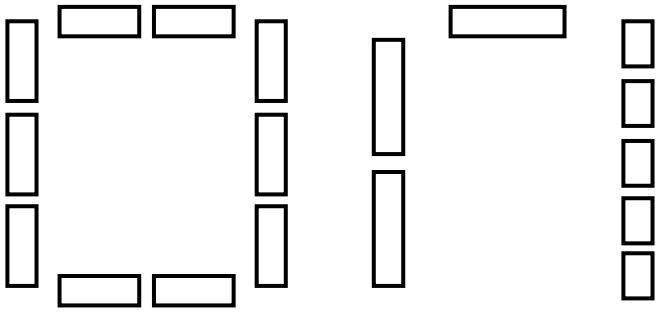


Figure 12: Strategies for circular dependencies.

Circulation Patterns

Circulation patterns play a dominant role in many functionally driven designs. Many of the military design solutions for spatial organization which were analyzed were observed to be very dependant on the ranges of options available around some primary circulation pattern. What follows is a non-exhaustive set of circulation pattern topologies which were observed, sorted by the number of segments involved, up to four. This is an area that may be of interest for further future research.

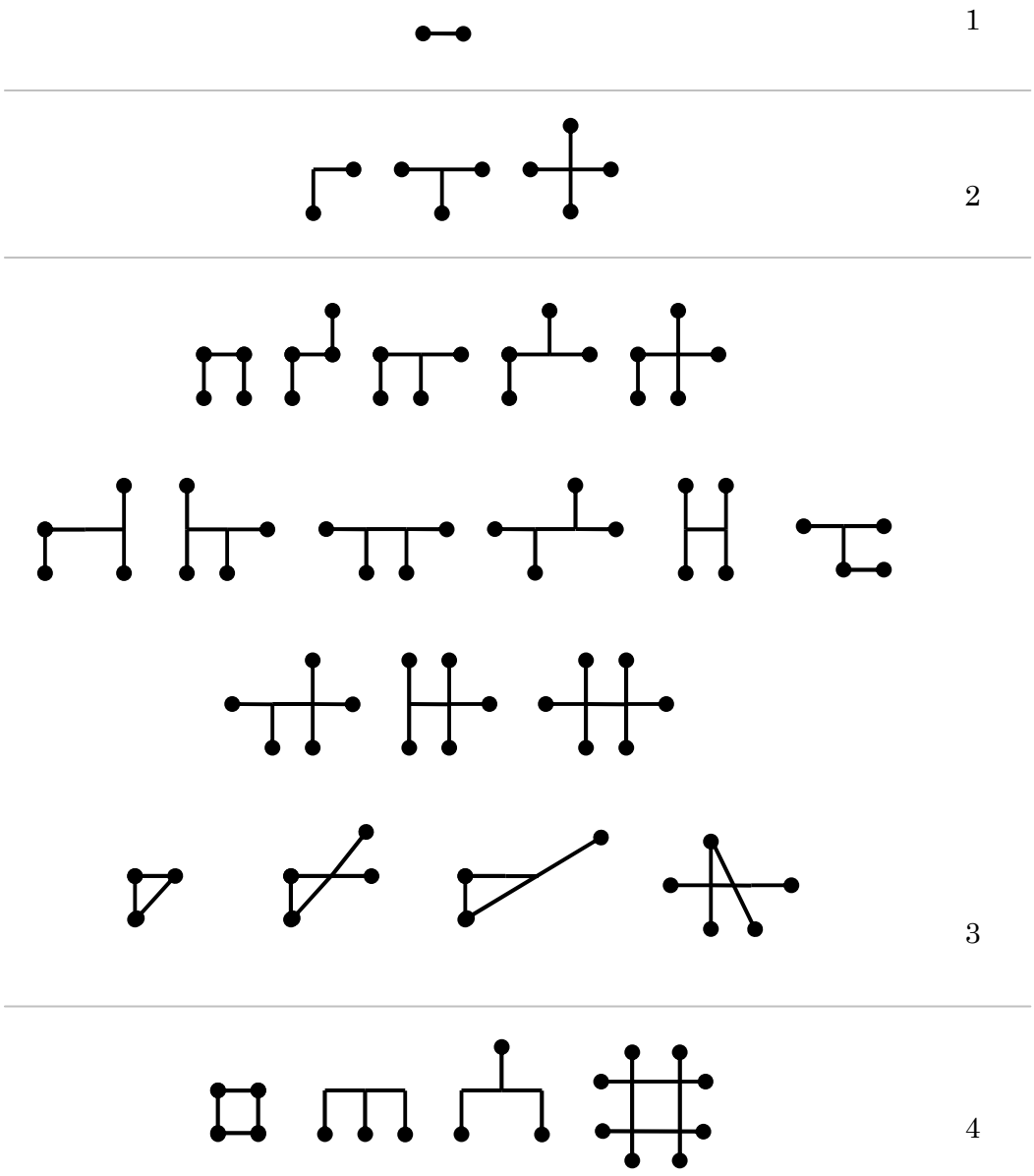


Figure 13: Circulation pattern topologies by number of segments.

Pattern Interaction

Of notable interest to this work, and an area for potential future research, is the question of interaction techniques with patterns. The direct feedback approach provided by today's graphical interfaces is a very efficient method for graphical interaction. However this has catered to interaction with geometry which include very basic geometric transformation operations which can be applied. The question of interaction with more semantically rich geometric structures in an efficient and natural fashion is an important unresolved issue.

In this work, an immersive environment was proposed where users would interact with a directed acyclic graph data structure which represented the inter and intra-pattern dependencies. A prototype immersive environment was developed in partnership with the University of Illinois through the integration of the Cube (a six-sided immersive virtual reality environment) and MicroStation, the Corps of Engineers standard CAD system. A system architecture diagram is included on the following page which outlines the various components and their interactions.

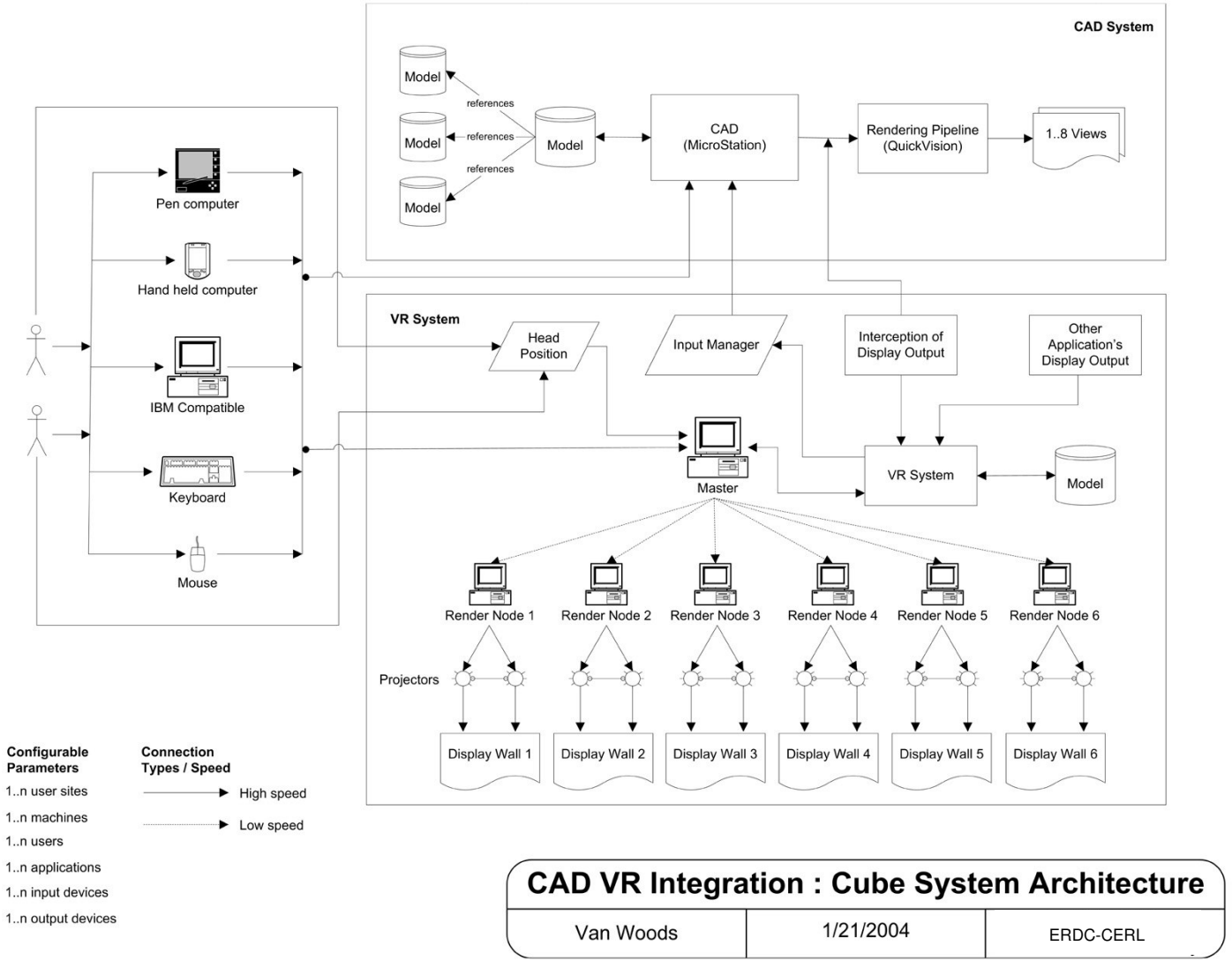


Figure 14: System architecture diagram of Cube virtual reality environment and CAD Integration

6 Conclusion

Summary

Military operations introduce the need for spatial planning capabilities in many contexts. Within the scope of this work, the impact of limitations to current capabilities in the space layout problem was addressed specifically. Optimization, shape grammars, case-based reasoning, and parametric modeling techniques and research were reviewed.

From the algorithmic perspective the space layout optimization problem is considered Nondeterministic Polynomial Hard (or NP-Hard). This results in unrealistically long solution search times as the search time rises to a power of the number of independent variables. Conversely, manual techniques are not robust enough to either allow exploration of alternative options in a rapid, predictable fashion, nor do they allow for user-defined behaviors to be encoded to be used and reused.

The basic premise of this work is that a hierarchical pattern-based design language provides a more robust approach. A pattern language was designed utilizing components of the Interactive Configuration Exploration (ICE) as the primary geometric and generative description, with the following extensions: programmatic control structure constructs, an equation-like approach to parameter definitions, an overall pattern-like syntax, and Interaction Object Graphs (IOG) for extended interaction definition.

Specifically, the pattern approach to design illustrated the following benefits:

1. Shift in problem approach from search to exploration. Rather than relying on a small set of globally or locally optimized parameters, the pattern based approach utilizes contextually relevant parameters and behavioral definitions. The pattern approach is specifically geared towards defining contextually compatible solutions in a flexible manner. It overcomes the common problems associated with implicit, tacit, and emergent constraints and requirements, conditions common and significant to design problems.

2. Strategic progression with localized optimization. The primary end-user utilization of the pattern based approach is envisioned to be as a user-directed process moving strategically through a series of steps to produce a desirable result. Progression leads to predictable changes that the designer expects and has full control over. This subjugates the final evaluation of value of solution quality in design to human evaluation while the computer is responsible for producing the expected result as quickly as possible.

3. Reduction of search space size. This approach trades exhaustive search for localized search, and trades constraint solving for parameter reduction and degree of freedom analysis. This results in more deterministic behavior and the likelihood of polynomial performance.

4. Parameterized solutions can be used to improve existing pure optimization methods. Many optimization routines are very sensitive to their start point. Utilizing a pattern with its default values as a starting point enables the algorithm to narrow the breadth of alternatives to avoid meaningless possibilities, therefore guiding the solver to a more likely solution space. Additionally, design state transformations within optimization routines could be controlled by pattern definitions therefore encouraging more realistic progression through the search space. Transformation of the actual pattern definition could also be employed to mutate the generation mechanism if novel solutions were desired.

5. While not rigorously analyzed, a design solution expressed in this notation is much more compact than lower-level traditional computational geometry techniques of persistence, such as storing coordinates of all entities. Instead what is Instead what is described and persisted are the relationships (geometric and logical) and the rules for maintaining those relationships in a desirable (from a design perspective) manner.

Future Research

Qualitative Feature Based (QFB) analysis is a promising avenue for future research as a means of potentially mining existing facilities for recurring patterns. While the process of identifying and encoding patterns will always require human intervention, it is possible that an automated process like QFB could statistically identify recurring geometric features and configurations. The

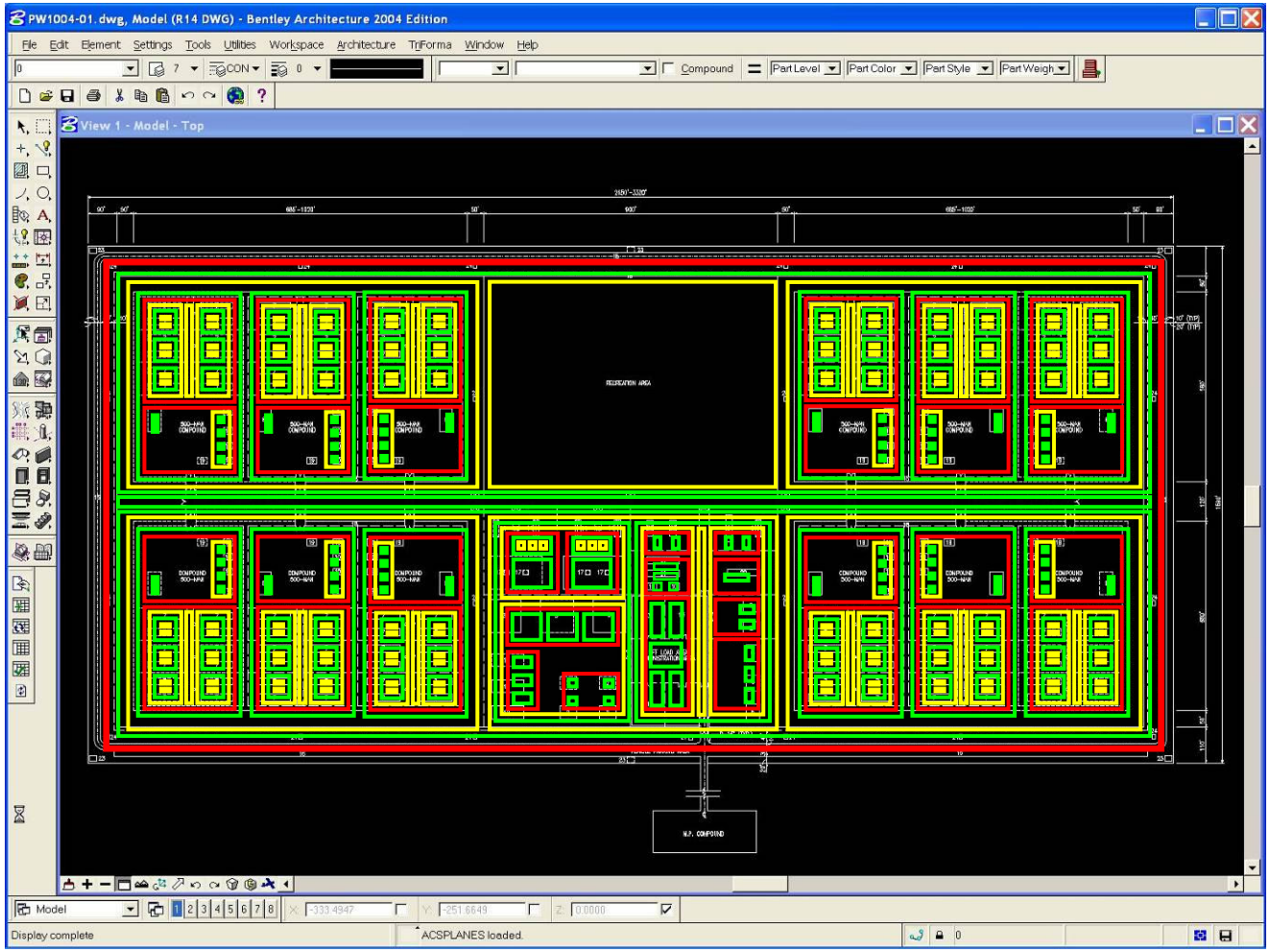
appeal with this approach is that it provides a dimension–independent and automated method for analysis of geometric configurations.

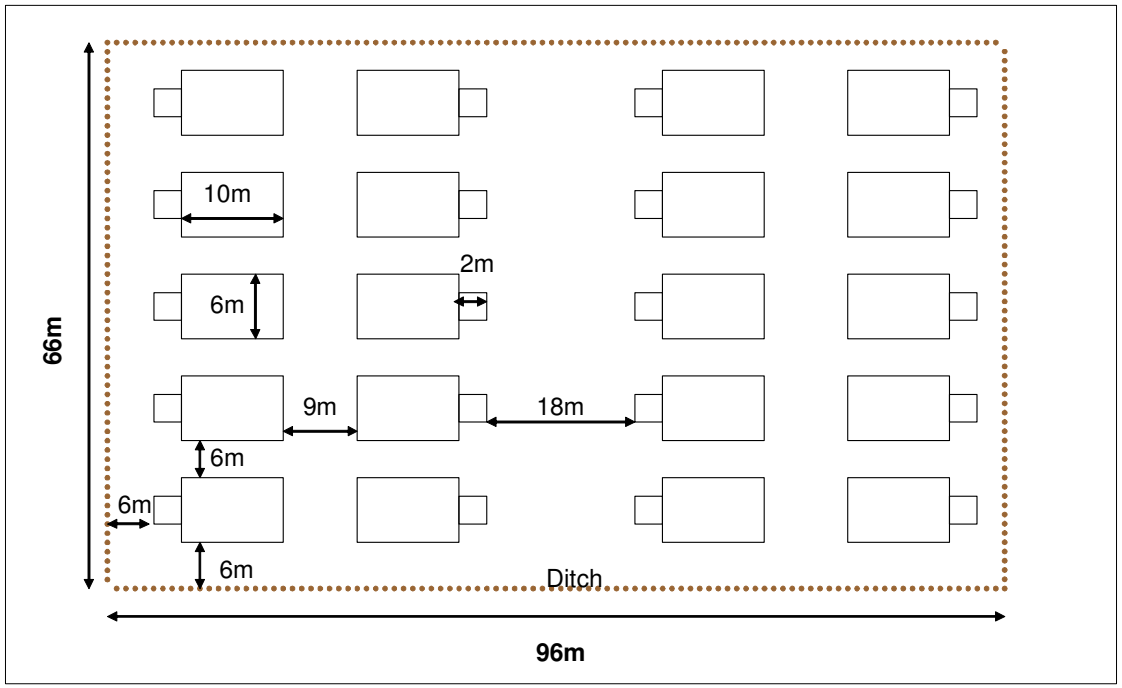
Additionally, as it was observed with the example building layouts, the following spatial constraints were identified as commonly recurring and may, depending on their computational characteristics, may or may not assist with traditional existing optimization approaches: symmetry constraint, axial/alignment constraint, spatial segment adjacency, and size/shape equivalence constraint.

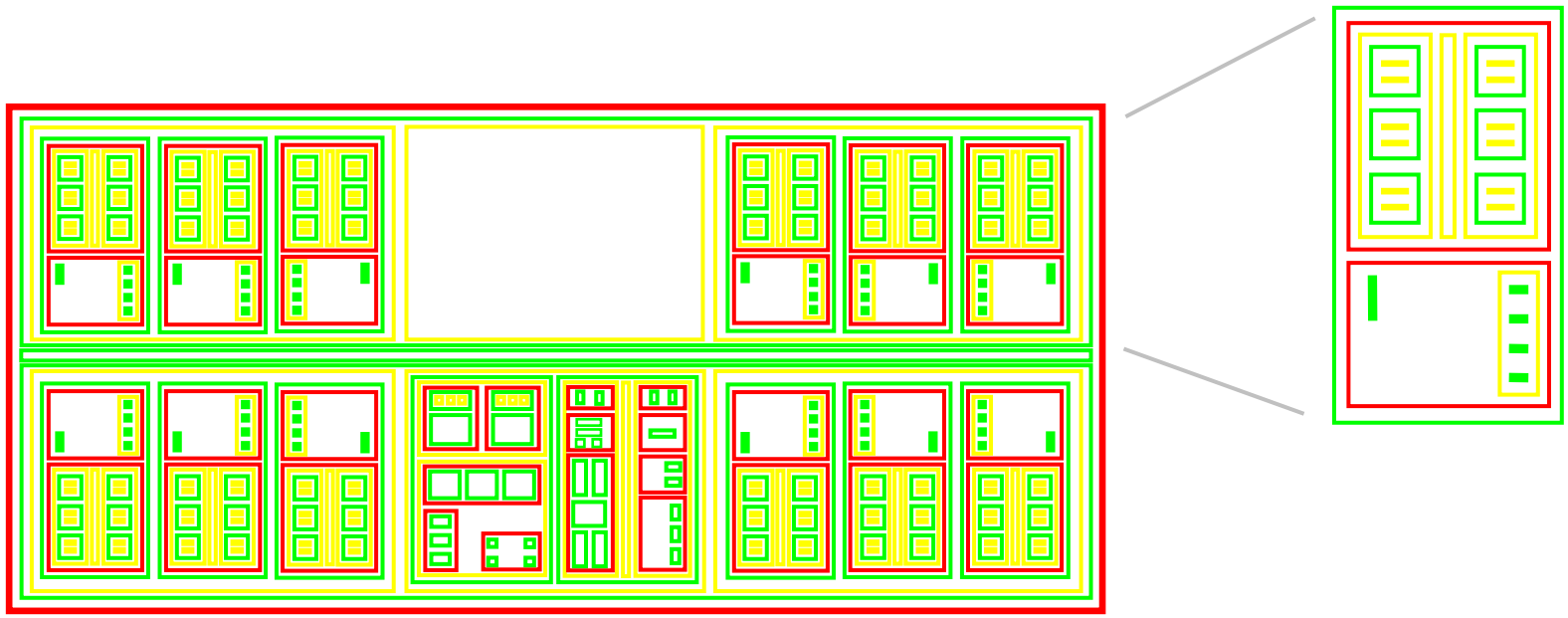
7 APPENDIX: Building Catalog

For the purposes of security, individual actual spatial functions will not be identified but rather identified uniquely by number, with the exception of the general category of circulation which will, when necessary, be differentiated.

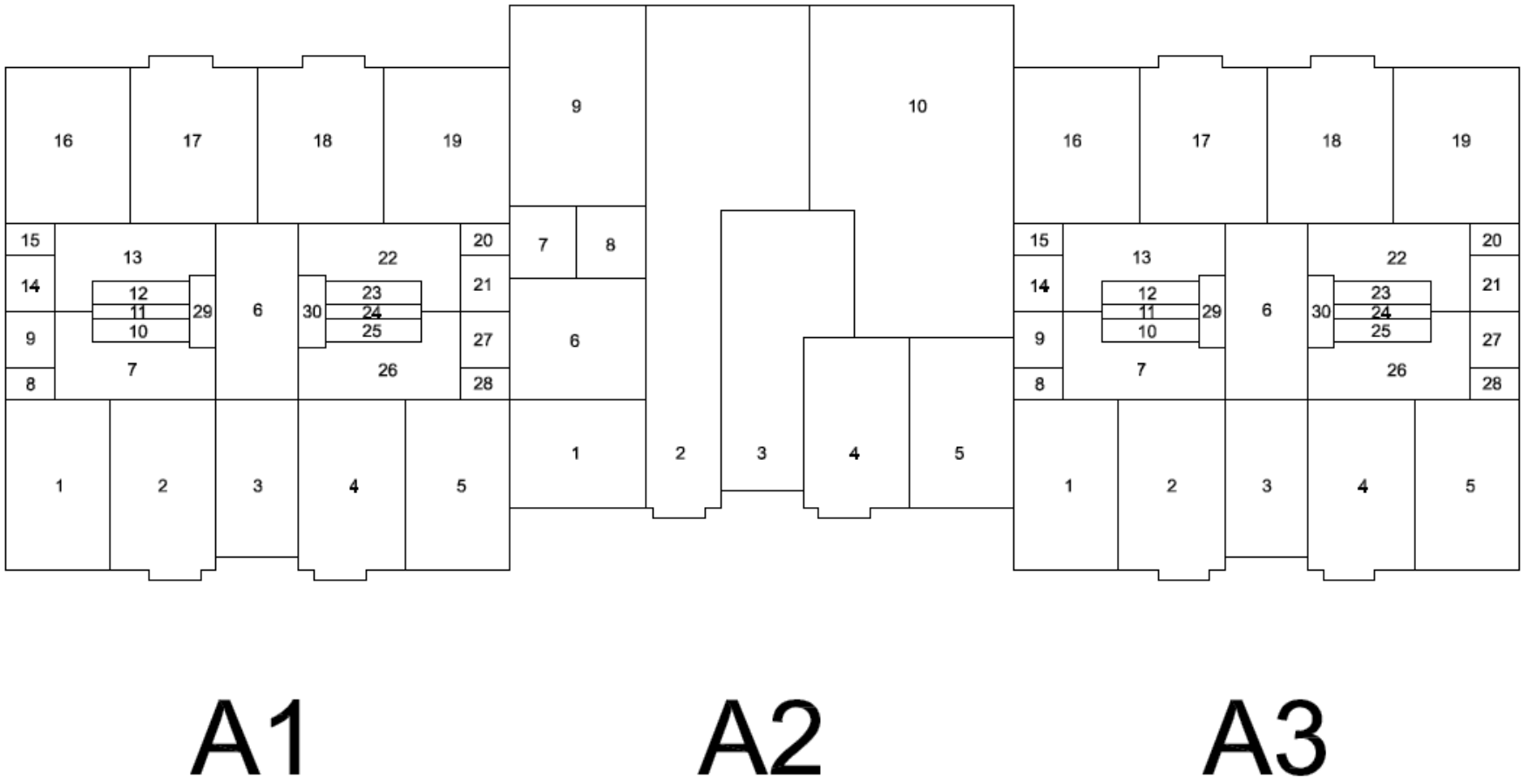
Basecamp Layout



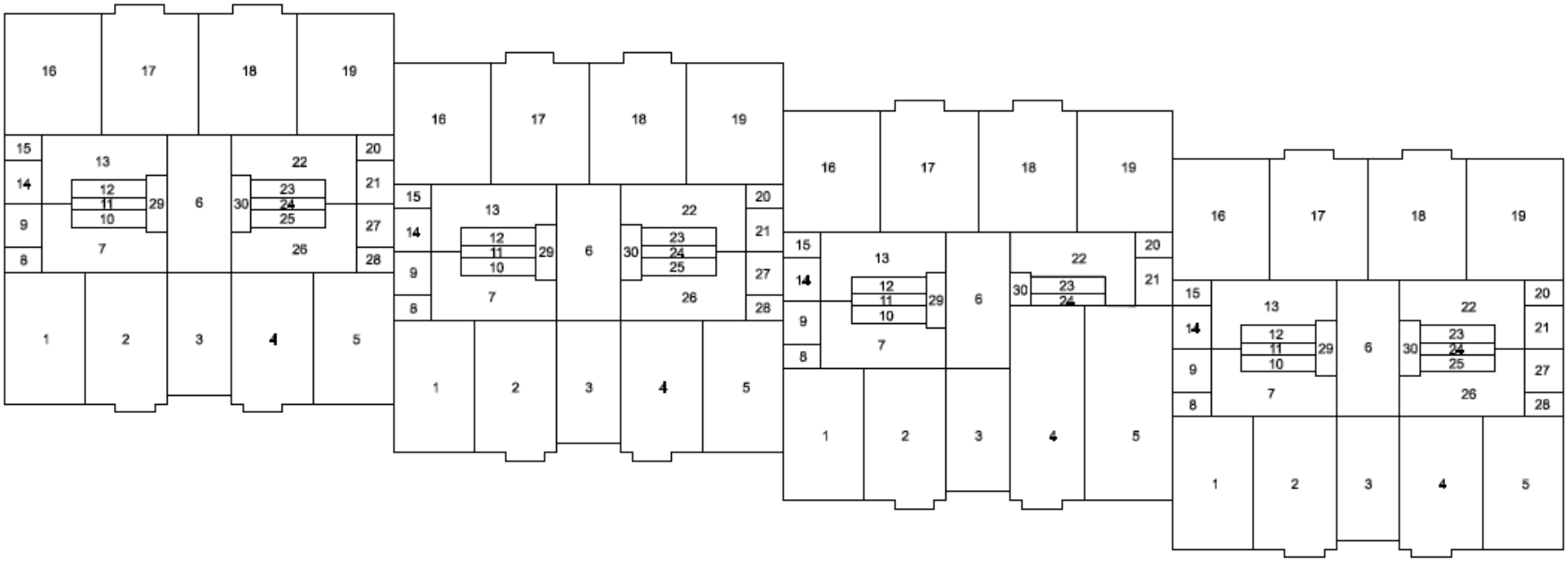

66m
10m
6m
2m
9m
18m
6m
6m
6m
Ditch
96m



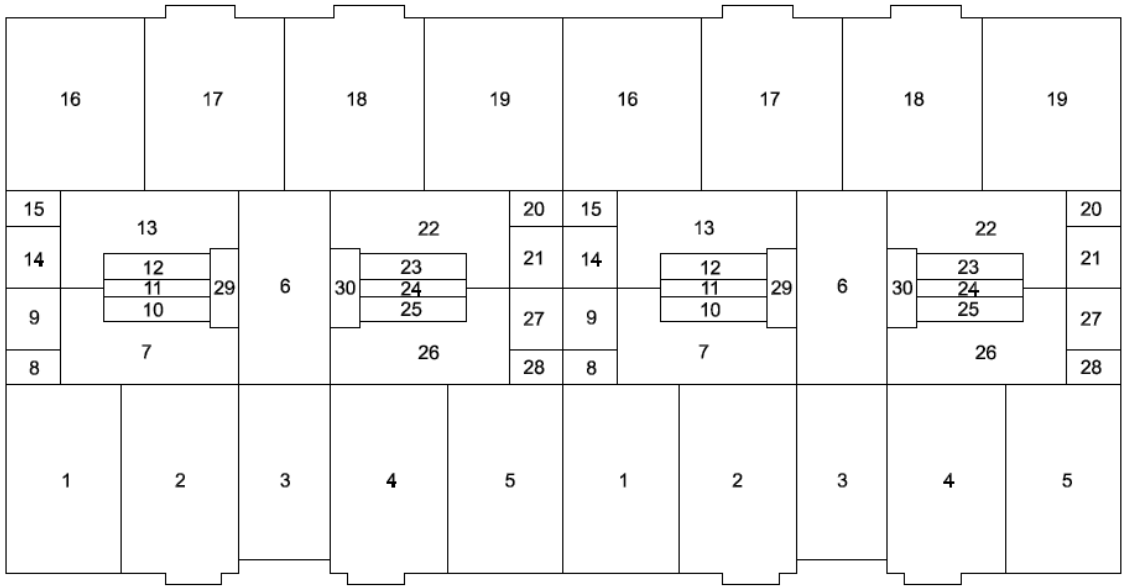
Building Name: FY06 Barracks Bldg A
FILE: Barracks-analysis-FL6B-1.dgn



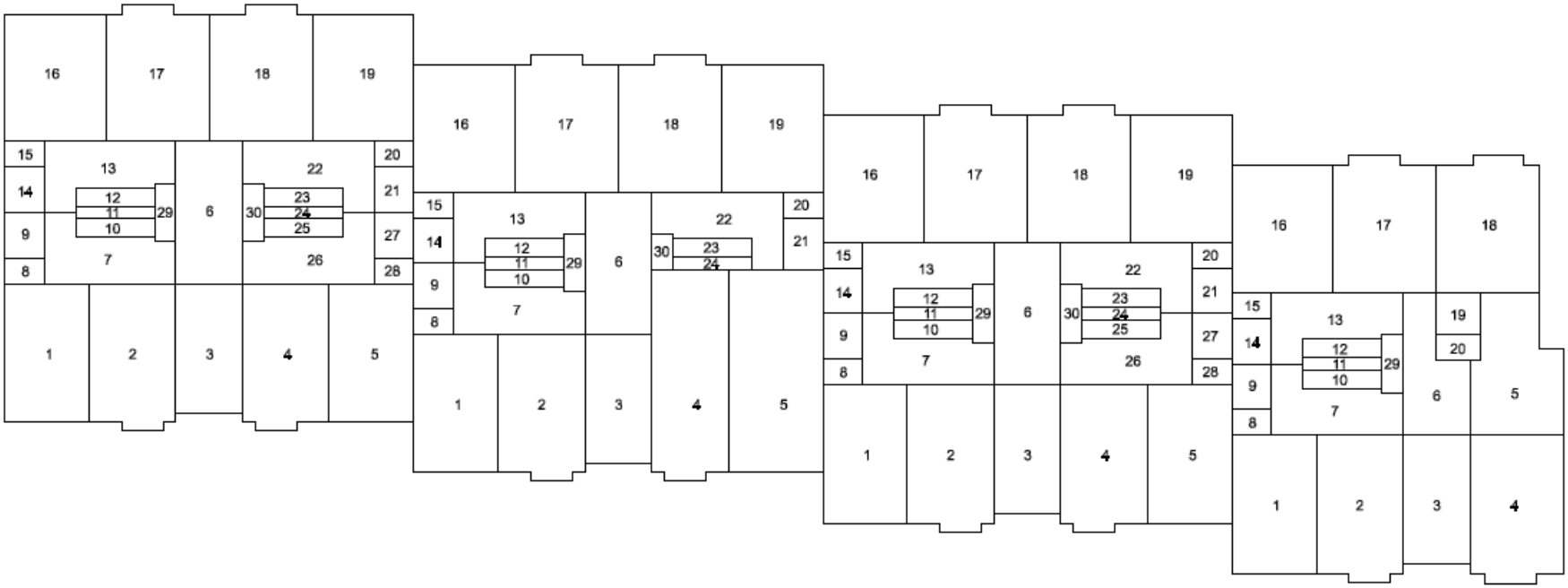
Building Name: FY06 Barracks Bldg B
FILE: Barracks-analysis-FL6B-2.dgn



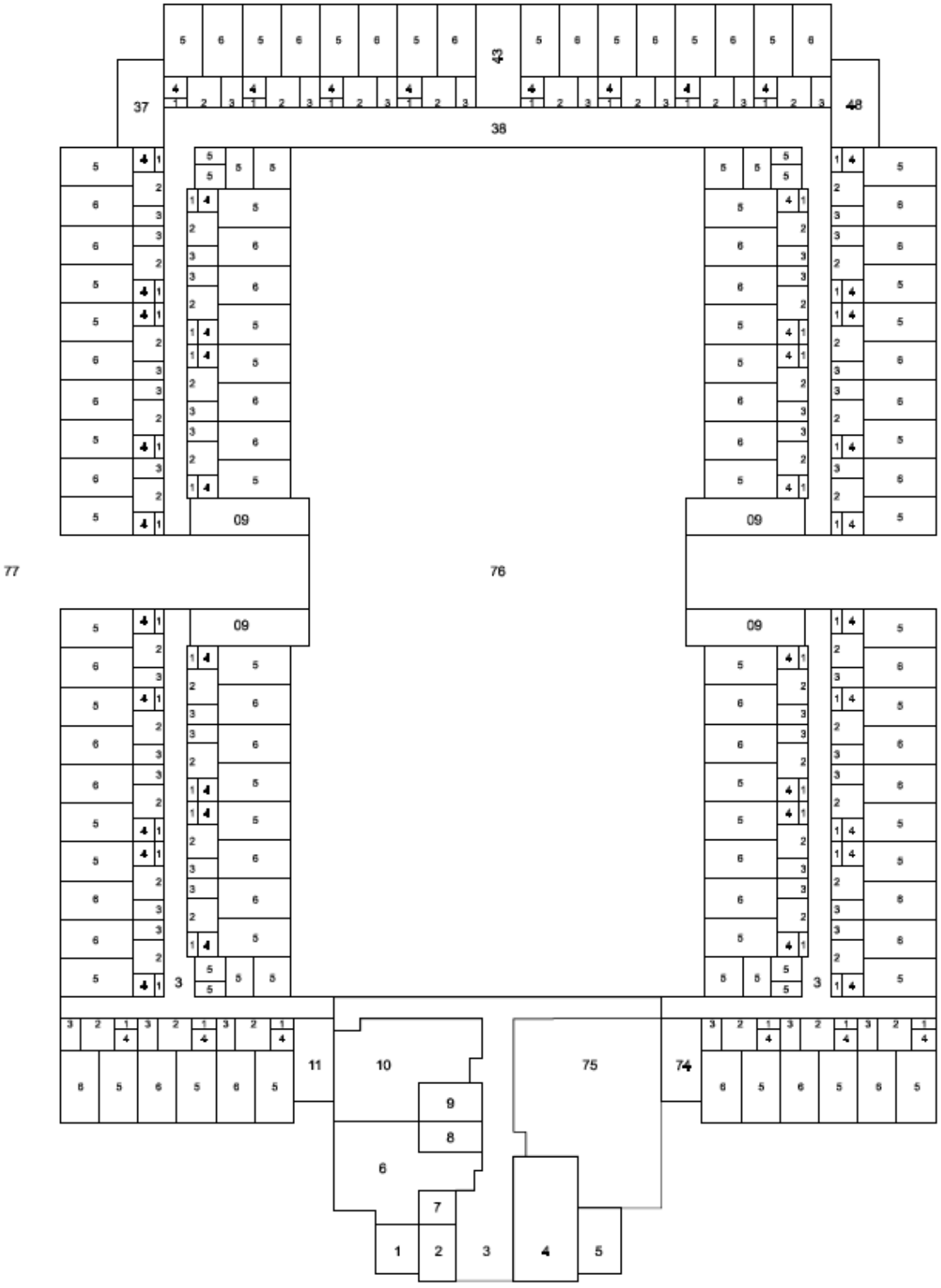
Building Name: FY06 Barracks Bldg C
FILE: Barracks-analysis-FL6B-3.dgn



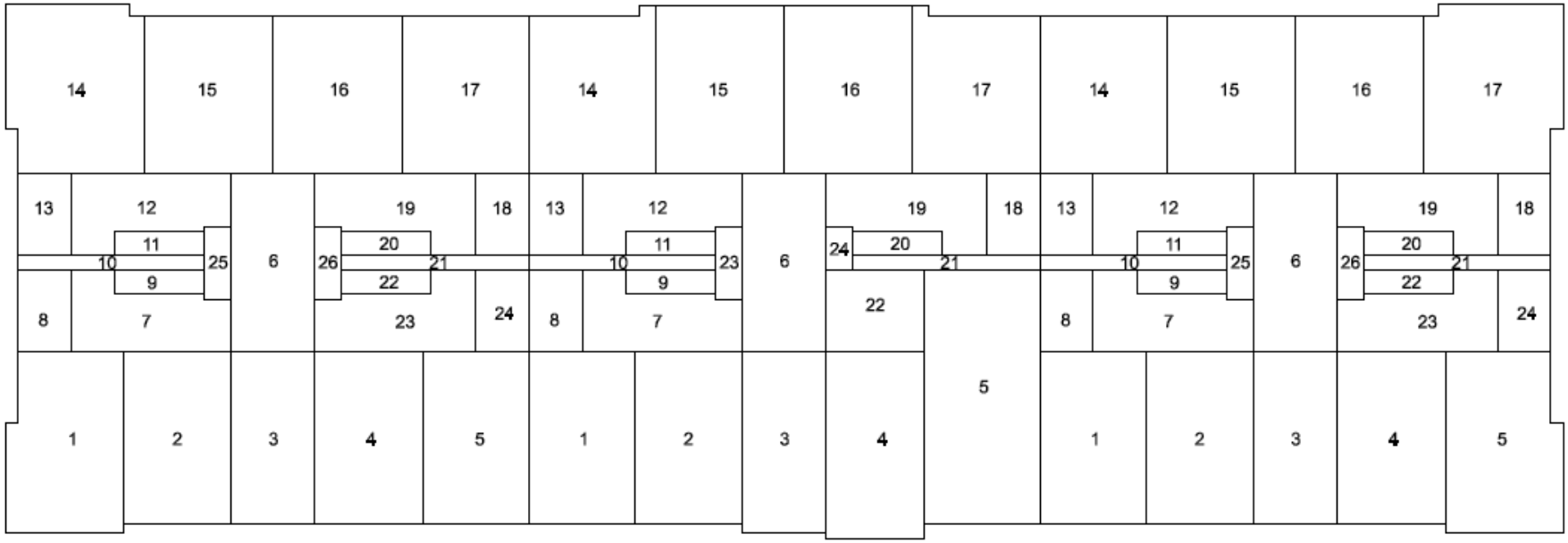
Building Name: FY06 Barracks Bldg D
FILE: Barracks-analysis-FL6B-4.dgn



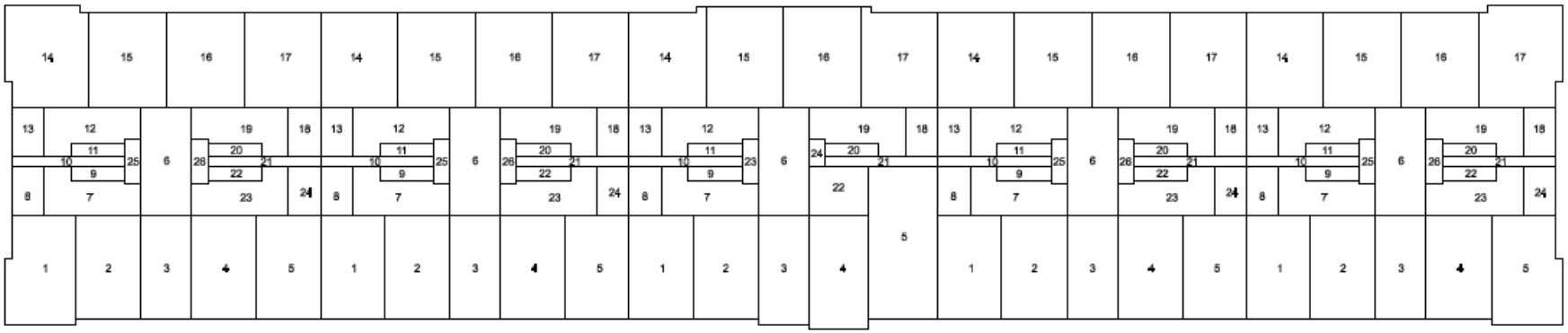
Building Name: FY02 Barracks
FILE: Barracks-analysis-FLW2.1.dgn



Building Name: FY03 Barracks Bldg A
FILE: Barracks-analysis-FL6W3.1.dgn

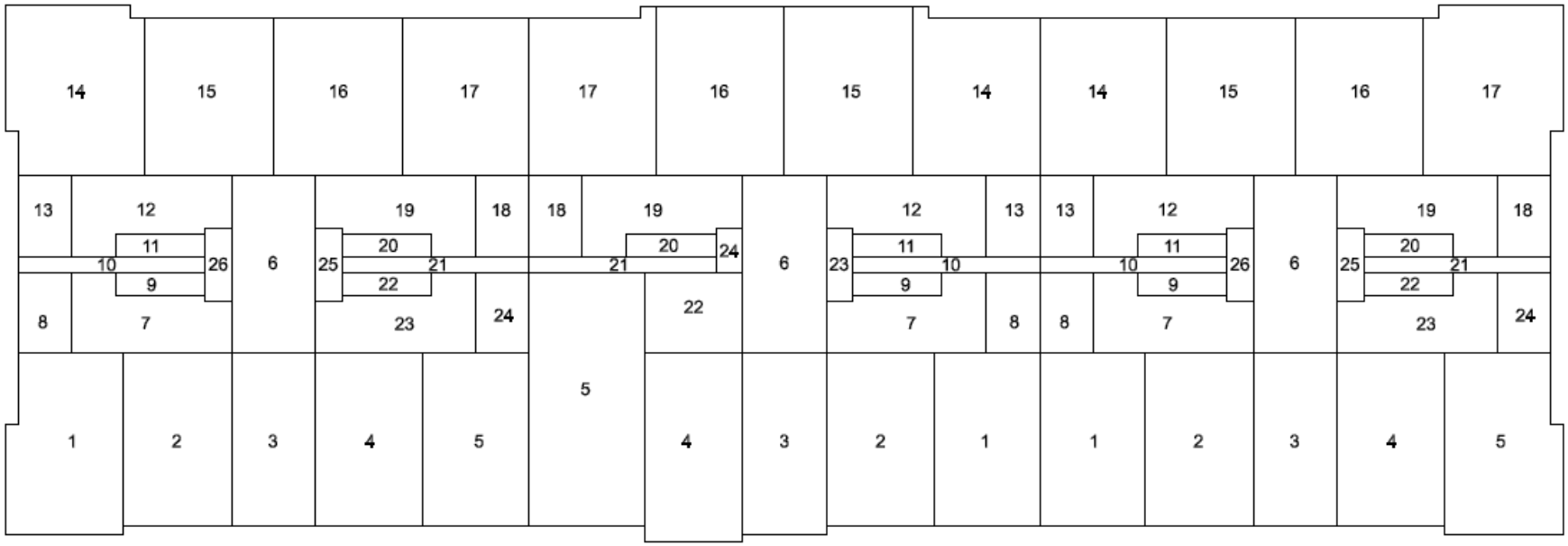


Building Name: FY03 Barracks Bldg B
FILE: Barracks-analysis-FL6W3.2.dgn

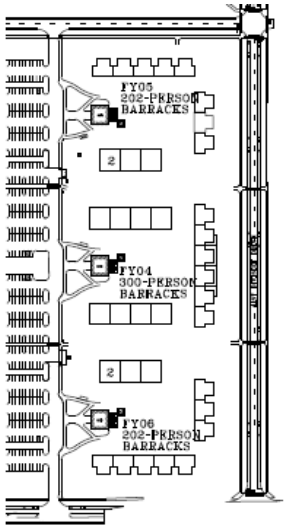


B1 B2 B3 B4 B5

Building Name: FY03 Barracks Bldg C
FILE: Barracks-analysis-FL6W3.3.dgn



C1 C2 C3



OPTION NO.1
E5/E6 AND E1/E4 MODULES

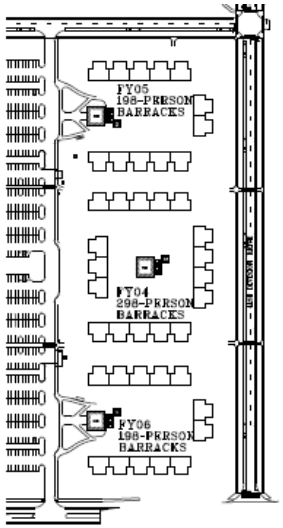
SCALE 1:2000

FY04 300-PERSON BARRACKS:

14	3-STORY CLUSTERS
3	SERVICE UNITS
30	E5/E6 MODULES (1 PN)
135	E1/E4 MODULES (2 PN)

FY05 & 06 202-PERSON BARRACKS:

10	3-STORY CLUSTERS
1	2-STORY CLUSTERS
3	SERVICE UNITS
48	E5/E6 MODULES (1 PN)
77	E1/E4 MODULES (2 PN)



OPTION NO.2
E5/E6 AND E1/E4 MODULES

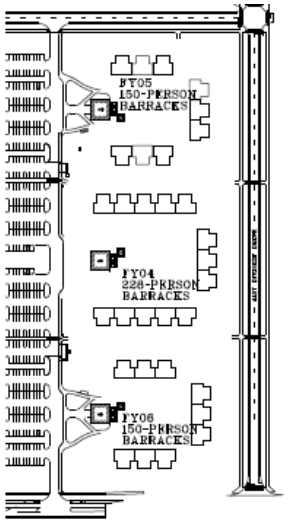
SCALE 1:2000

FY04 298-PERSON BARRACKS:

17	3-STORY CLUSTERS
4	SERVICE UNITS
102	E5/E6 MODULES
196	E1/E4 MODULES

FY05 & 06 198-PERSON BARRACKS:

10	3-STORY CLUSTERS
2	2-STORY CLUSTERS
3	SERVICE UNITS
68	E5/E6 MODULES
130	E1/E4 MODULES



OPTION NO.3
E5/E6 AND E1/E4 MODULES

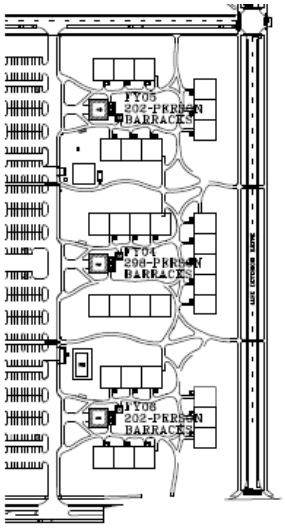
SCALE 1:2000

FY04 228-PERSON BARRACKS:

13	3-STORY CLUSTERS
3	SERVICE UNITS
78	E5/E6 MODULES
150	E1/E4 MODULES

FY05 & 06 150-PERSON BARRACKS:

8	3-STORY CLUSTERS
1	2-STORY CLUSTERS
3	SERVICE UNITS
52	E5/E6 MODULES
98	E1/E4 MODULES



OPTION NO.4
ALL MODULES SIMILAR

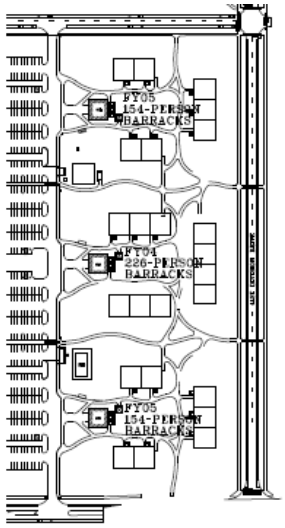
SCALE 1:2000

FY04 298-PERSON BARRACKS:

12	3-STORY CLUSTERS
1	2-STORY CLUSTERS
3	SERVICE UNITS
298	SIMILAR MODULES

FY05 & 06 202-PERSON BARRACKS:

8	3-STORY CLUSTERS
1	2-STORY CLUSTERS
3	SERVICE UNITS
202	SIMILAR MODULES



OPTION NO.5
ALL MODULES SIMILAR

SCALE 1:2000

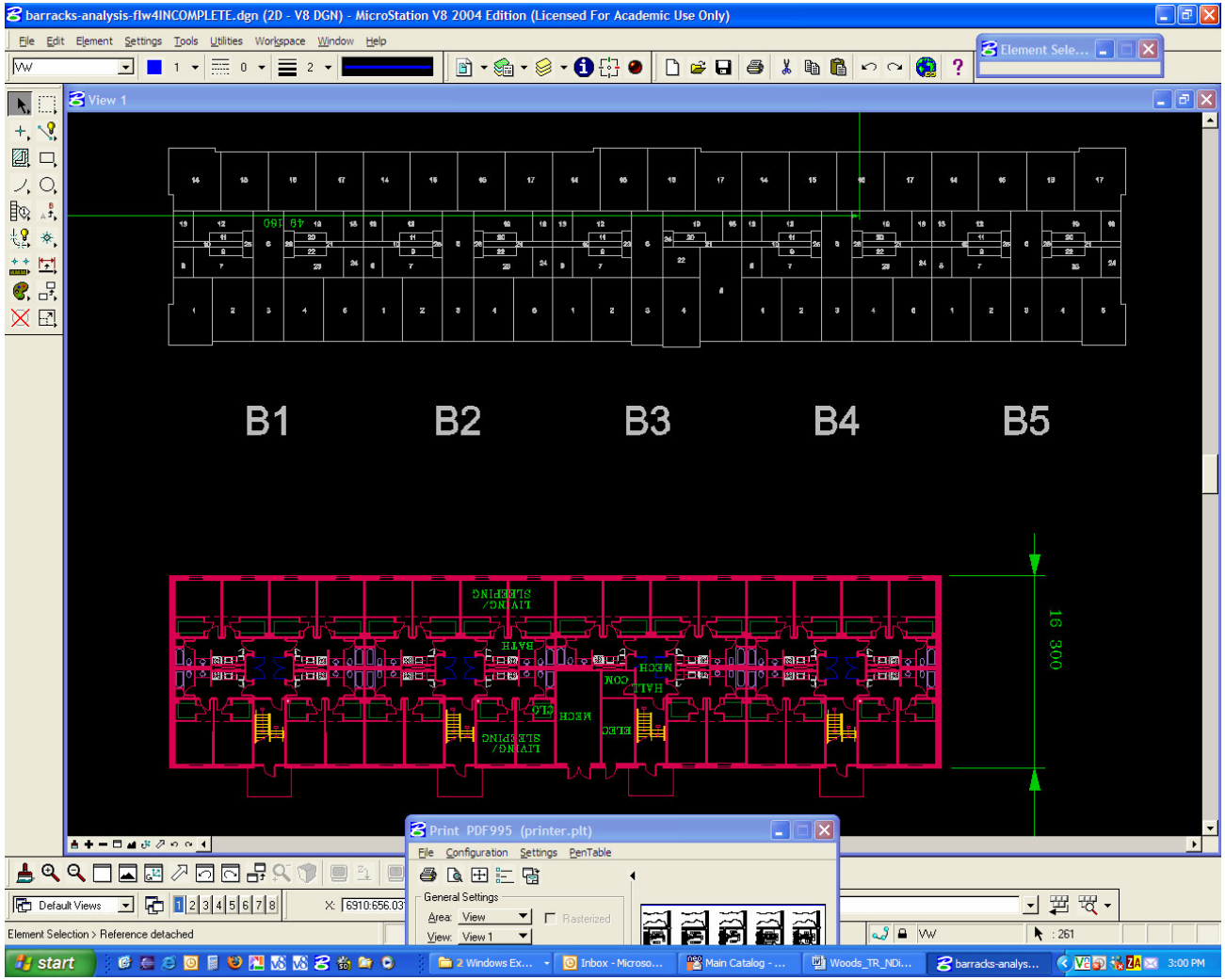
FY04 226-PERSON BARRACKS:

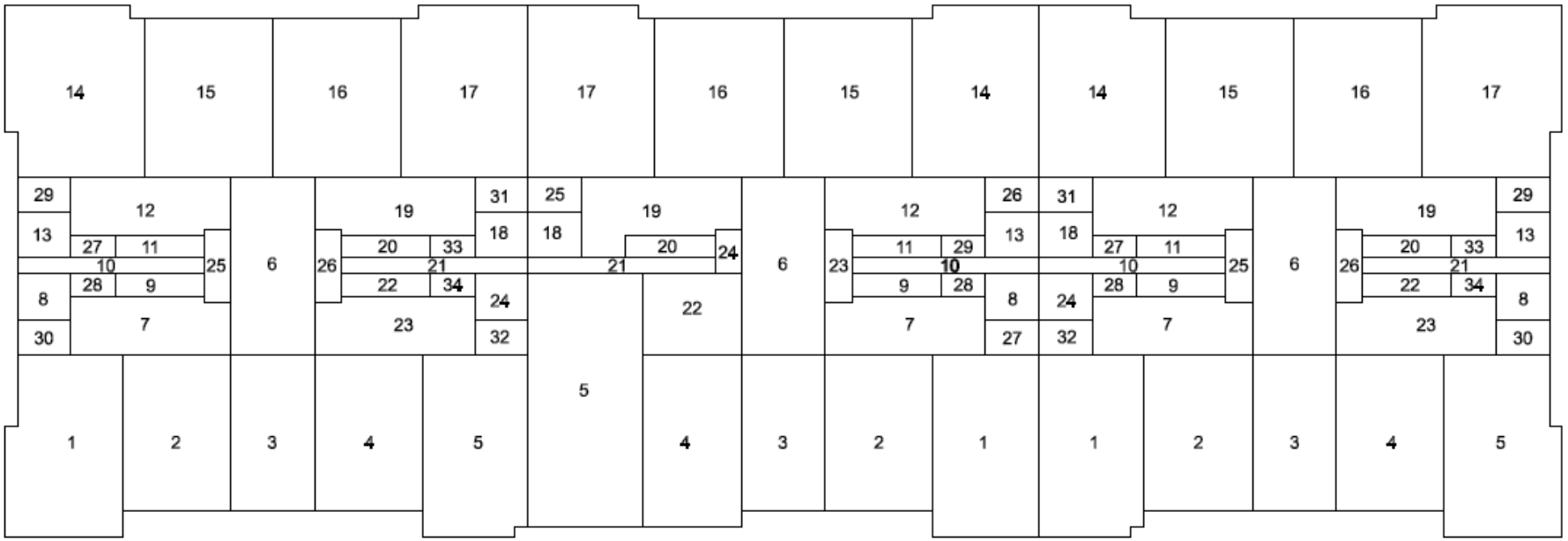
9	3-STORY CLUSTERS
1	2-STORY CLUSTERS
3	SERVICE UNITS
226	SIMILAR MODULES

FY05 & 06 154-PERSON BARRACKS:

6	3-STORY CLUSTERS
1	2-STORY CLUSTERS
3	SERVICE UNITS
154	SIMILAR MODULES

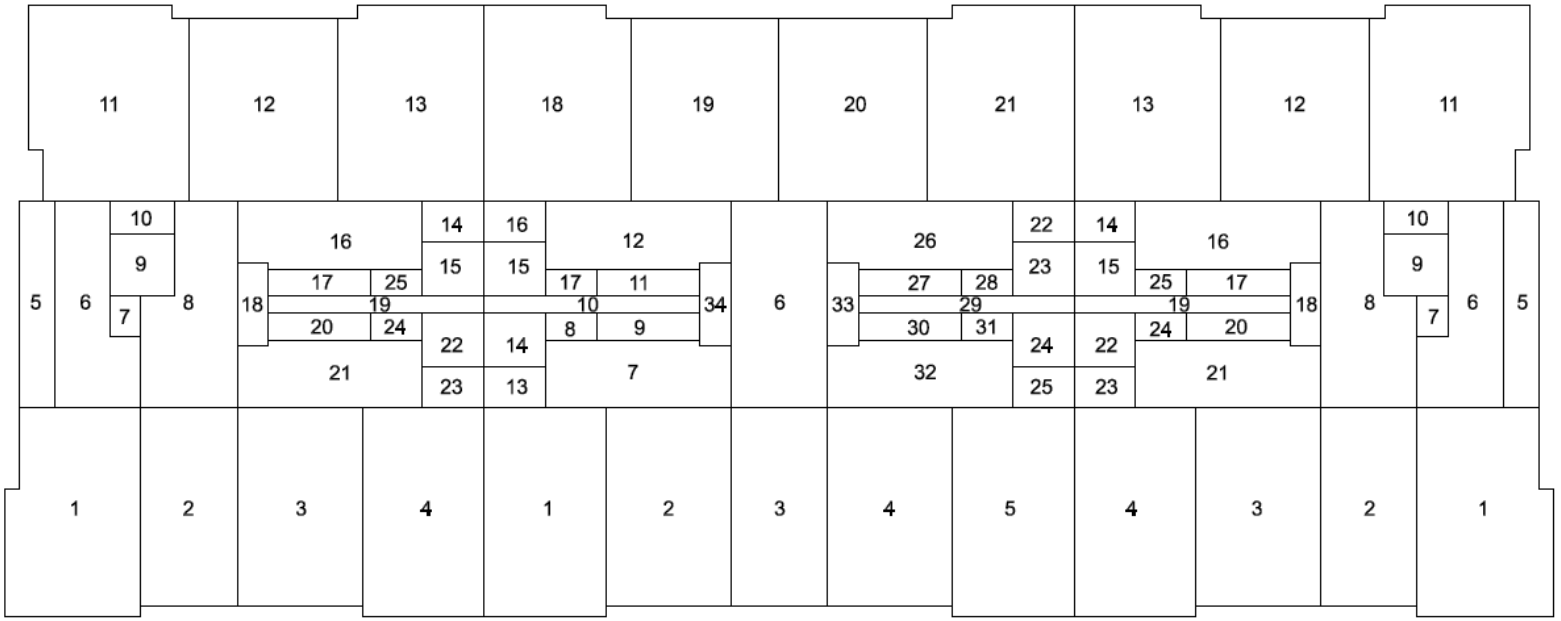
Building Name: FY05 Barracks Bldg A
FILE: Barracks-analysis-FLW51.dgn



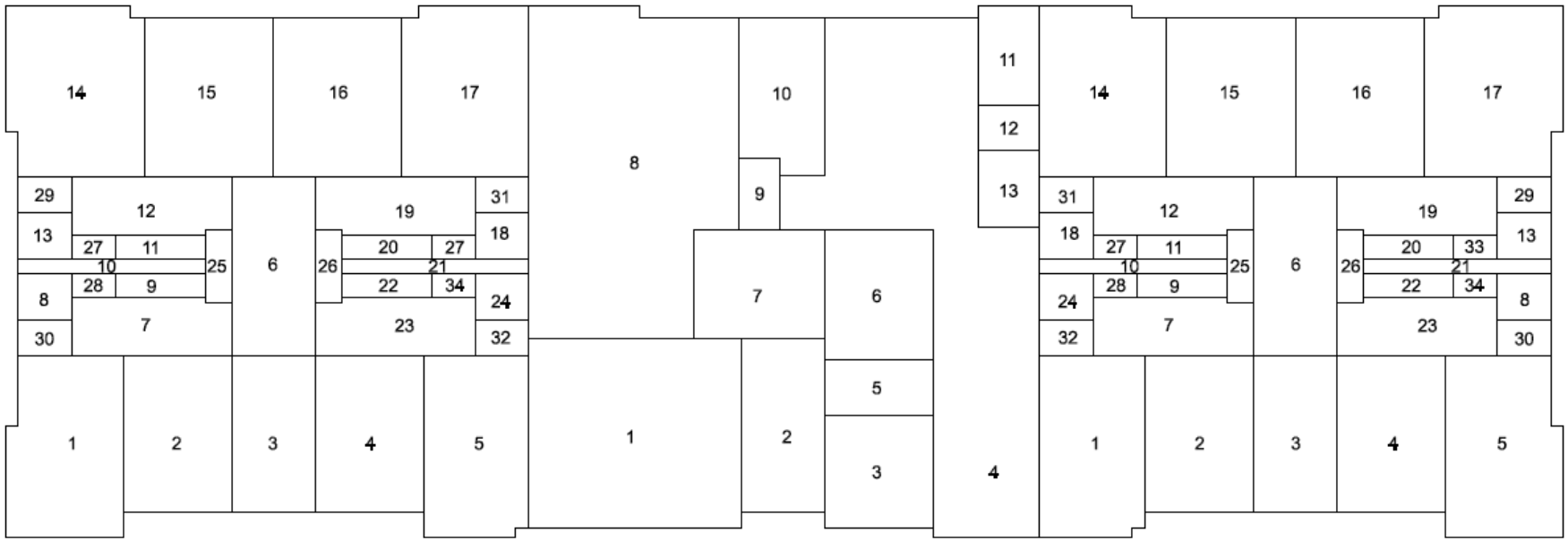

14 15 16 17 17 16 15 14 14 15 16 17
29 12 19 31 25 19 12 26 31 12 19 29
13 27 11 18 18 20 24 11 29 13 18 27 11 20 33 13
10 25 6 26 20 33 21 21 6 23 10 10 25 6 26 21
8 28 9 22 34 24 9 28 8 24 28 9 22 34 8
30 7 23 32 22 7 27 32 7 23 30
1 2 3 4 5 5 4 3 2 1 1 2 3 4 5

A1 A2 A3

Building Name: FY05 Barracks Bldg B
FILE: Barracks-analysis-FLW52.dgn

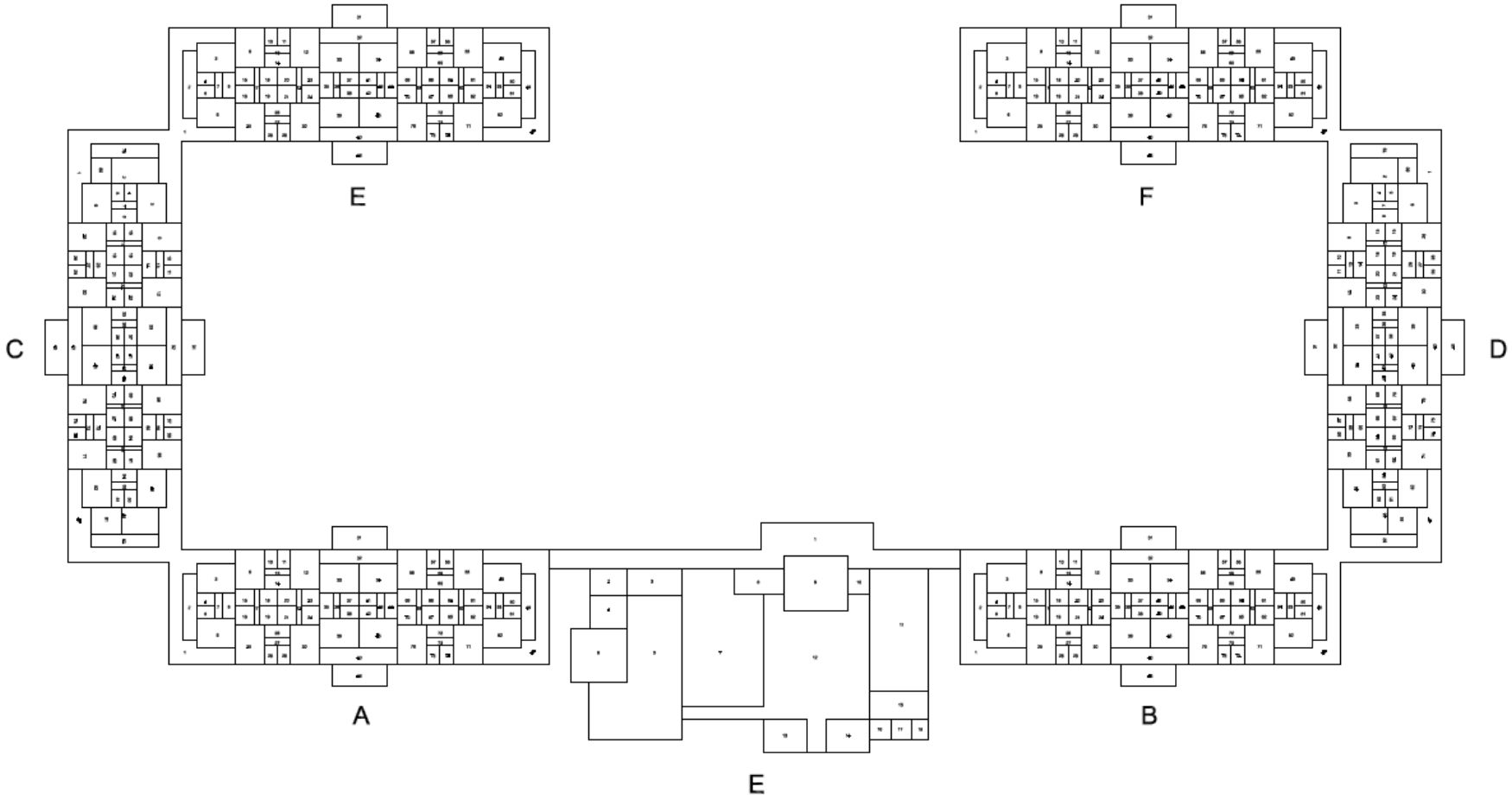


Building Name: FY05 Barracks Bldg C
FILE: Barracks-analysis-FLW53.dgn

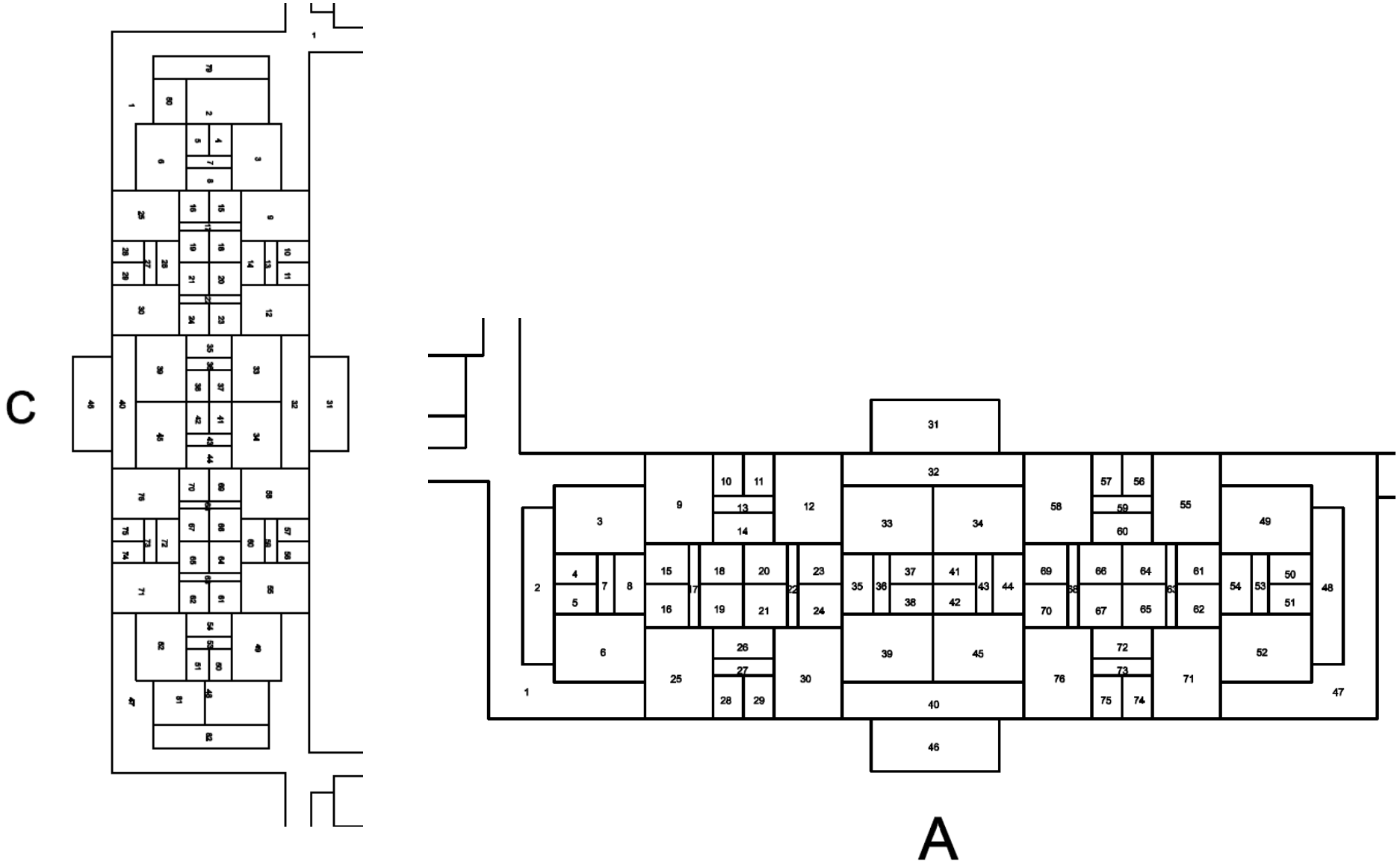


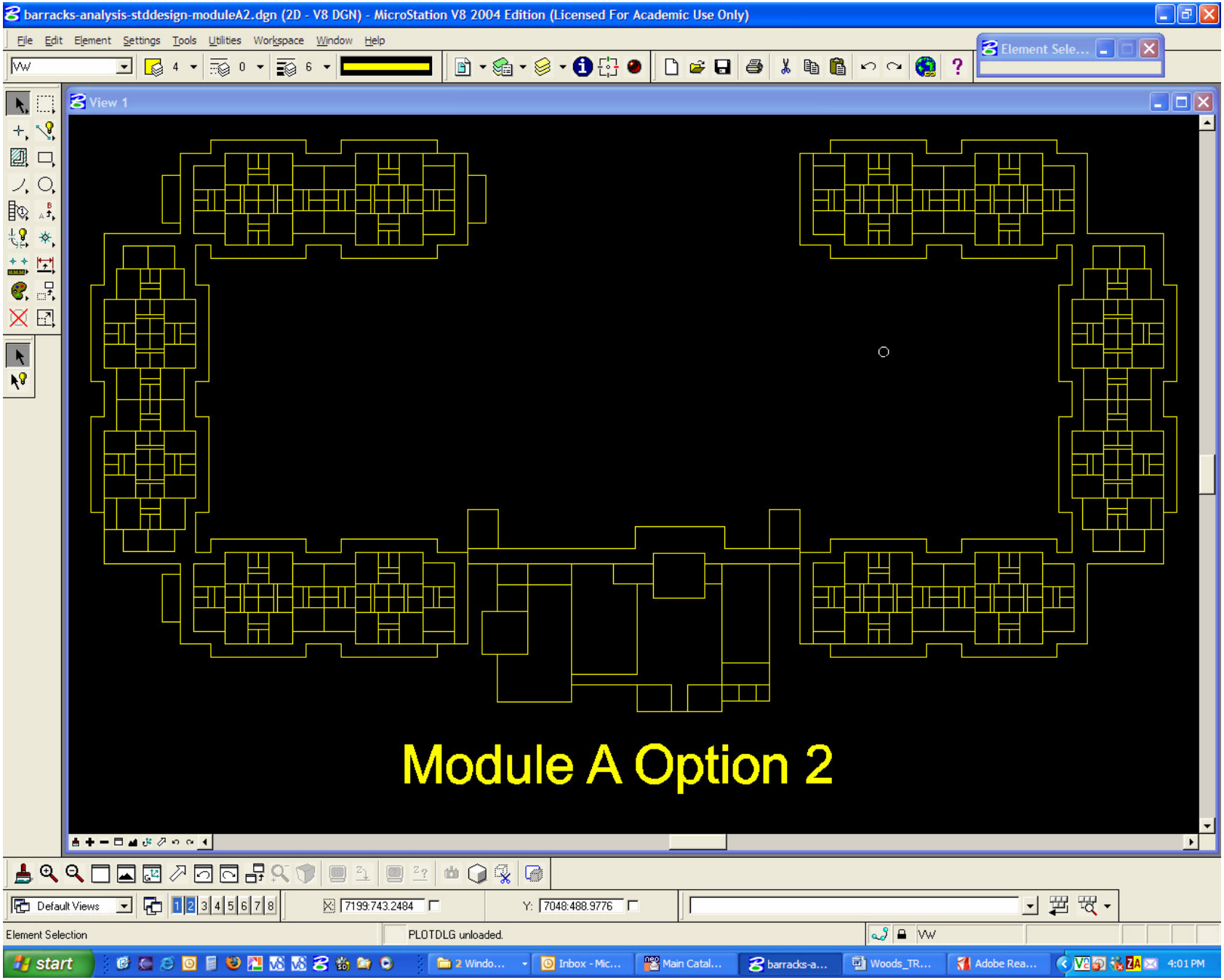
C1 C2 C3

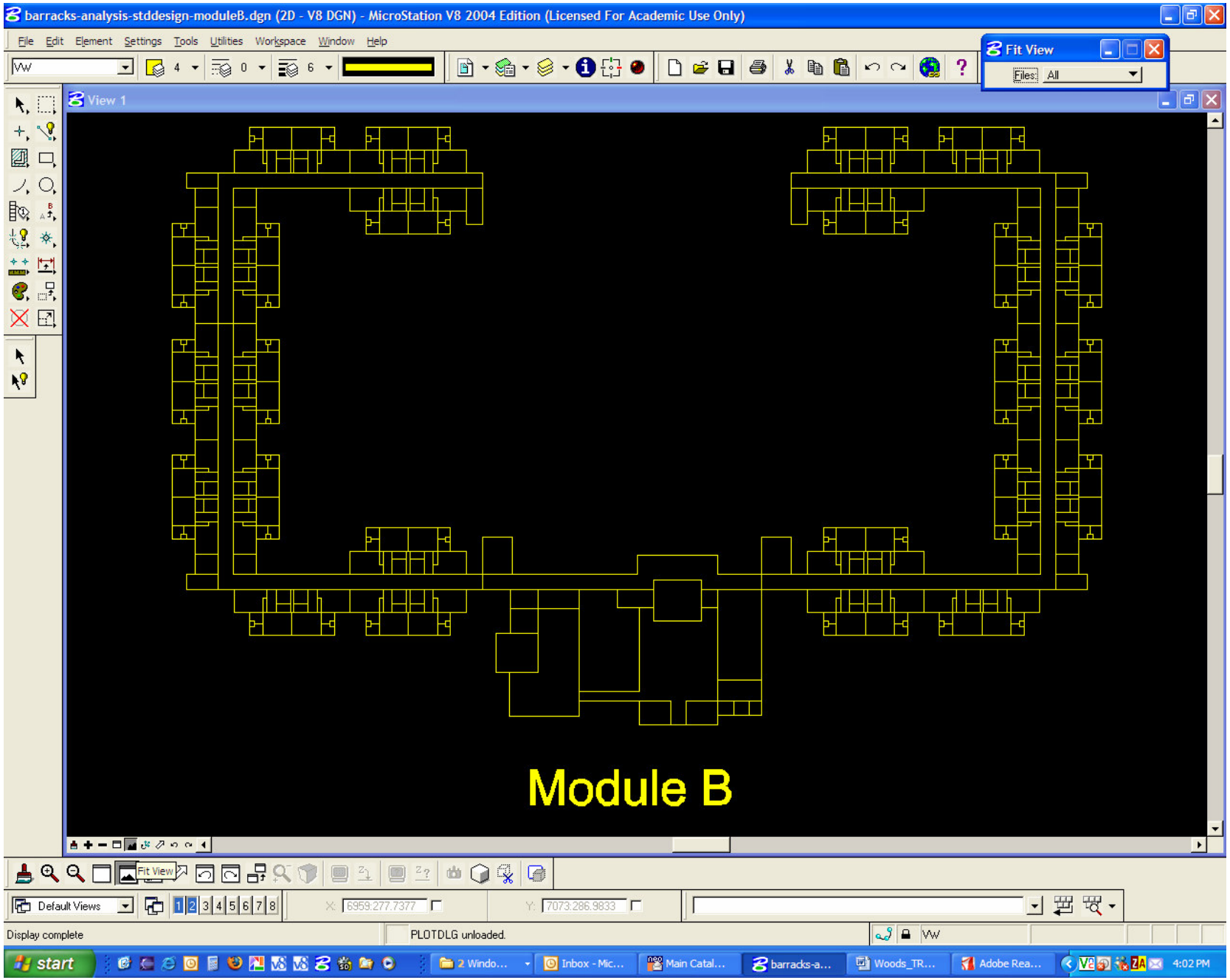
Building Name: FY05 Barracks Bldg C
FILE: Barracks-analysis-stddesign-moduleA1.dgn

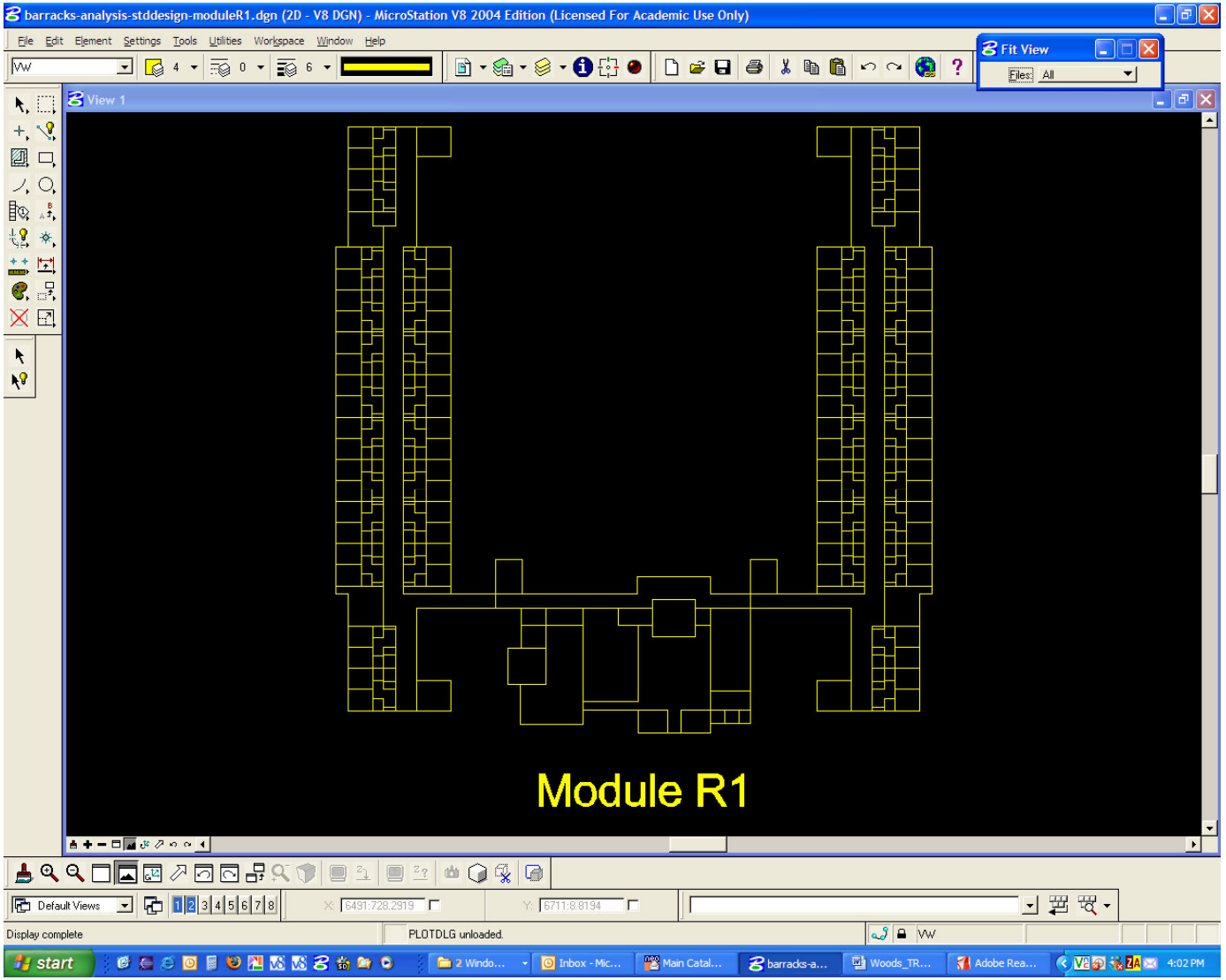


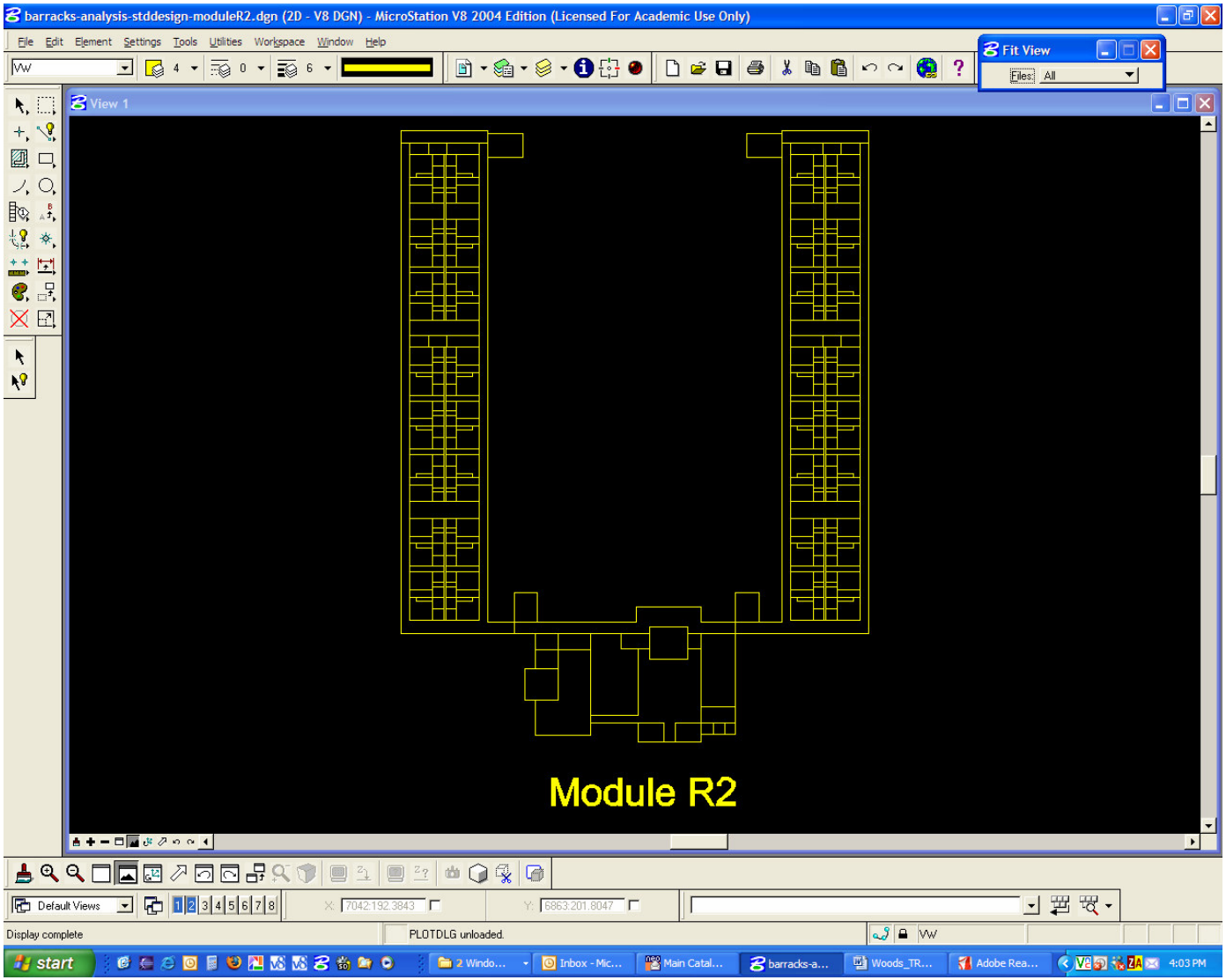
Building Name: Standard Design A1 (Large scale view)
FILE: Barracks-analysis-stddesign-moduleA1.dgn

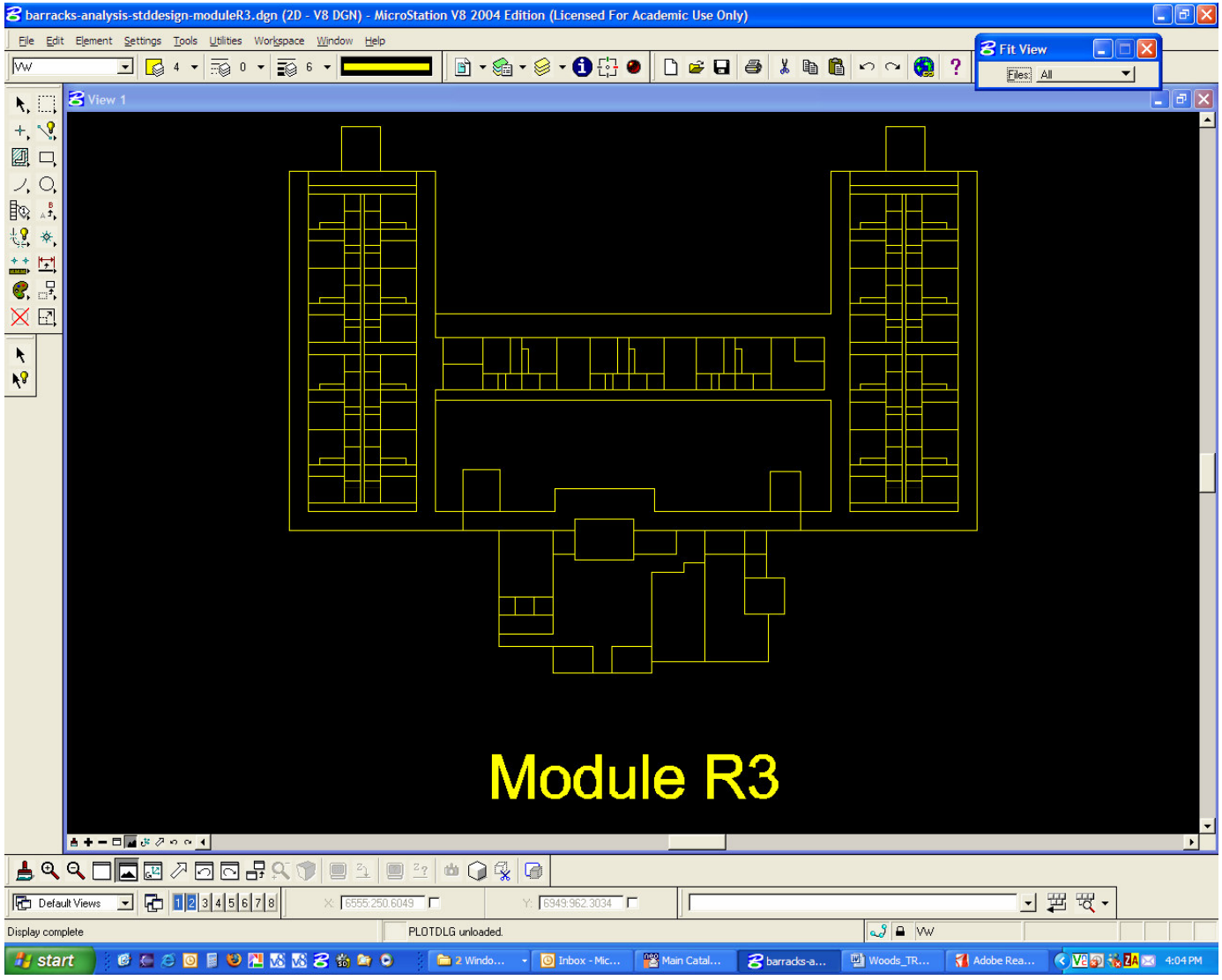


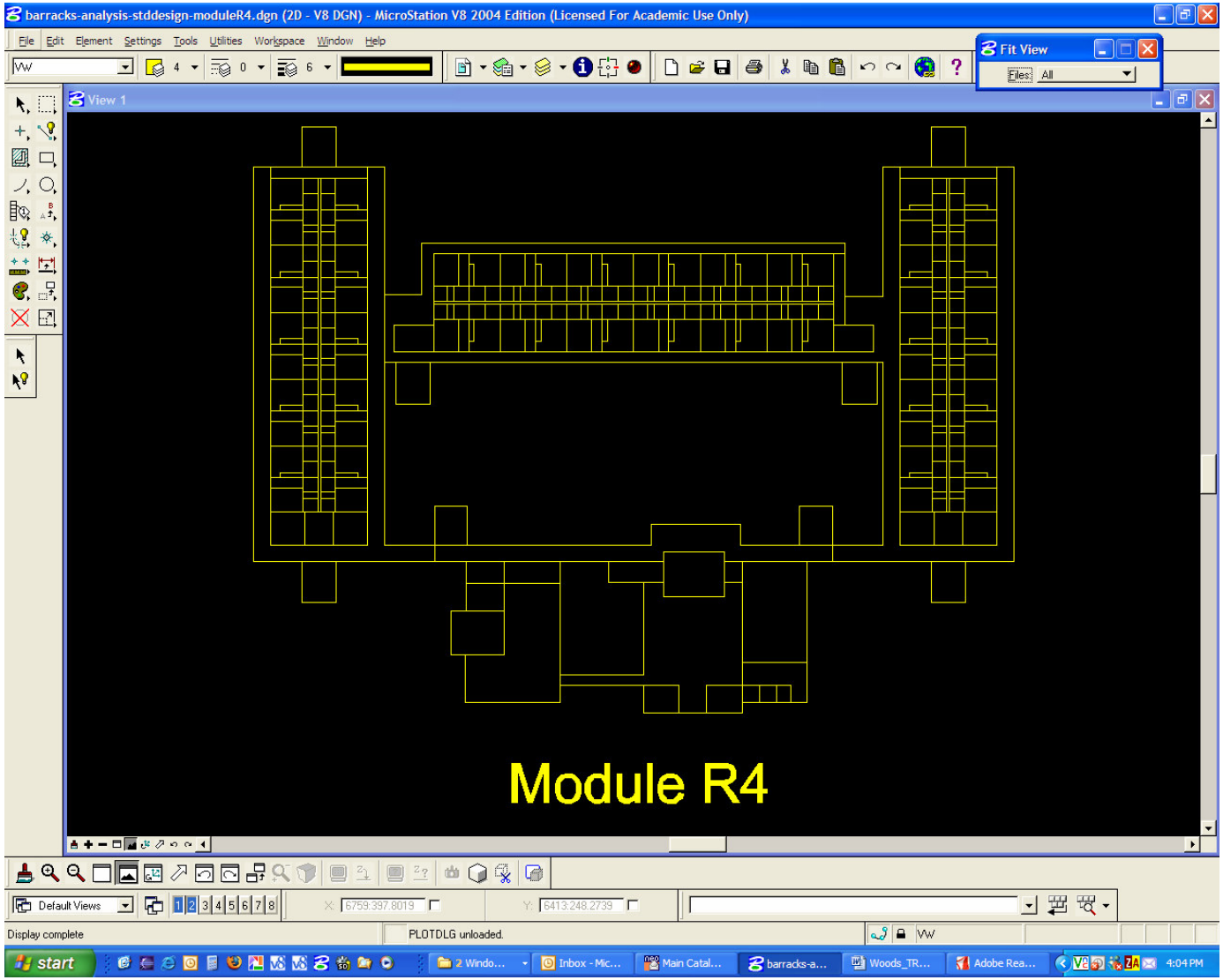

barracks-analysis-stddesign-moduleA2.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
View 1
Module A Option 2
Default Views
Element Selection
PLOTDLG unloaded.
start

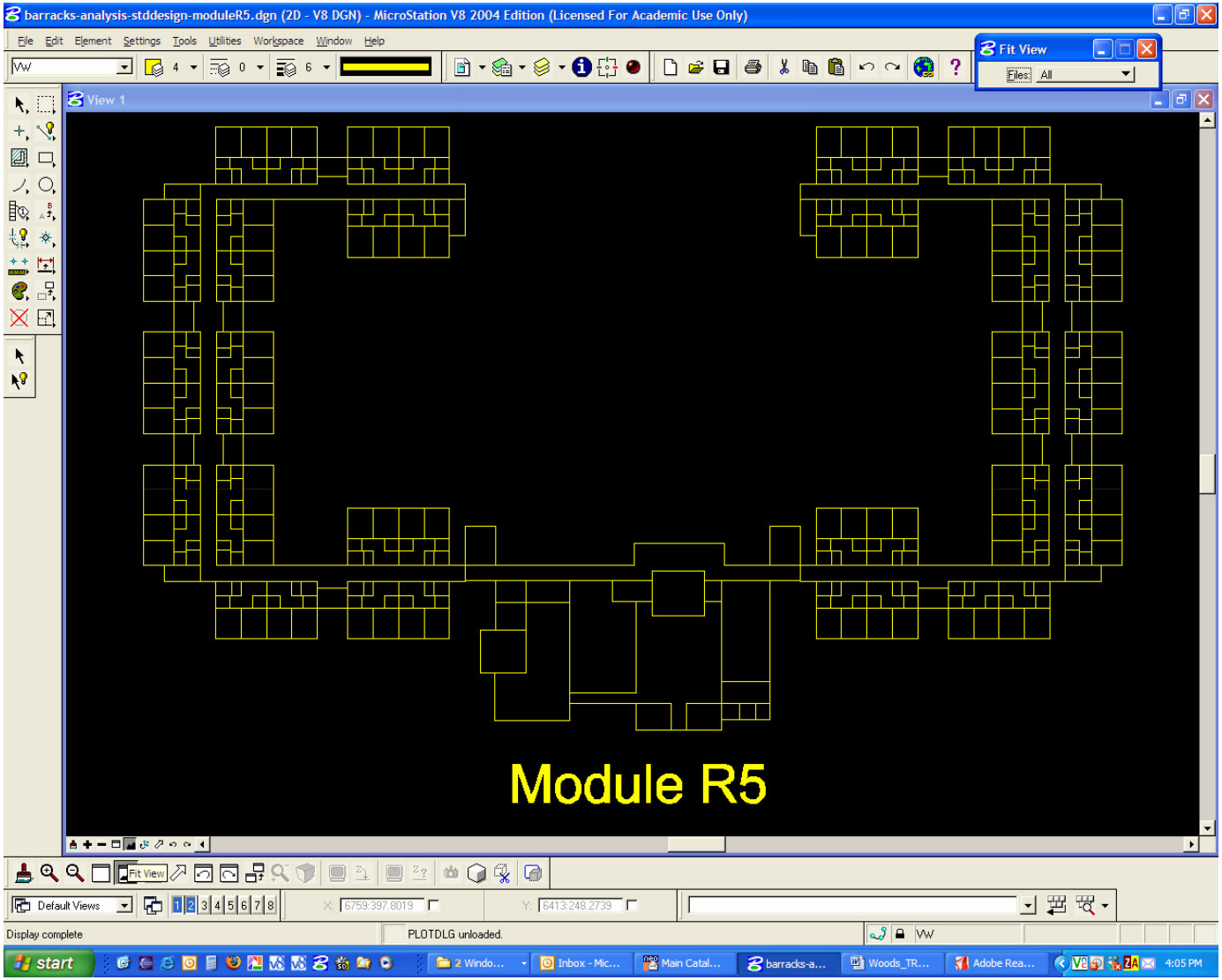

barracks-analysis-stddesign-moduleB.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
Fit View
View 1
Module B
Default Views
Display complete
PLOTDLG unloaded.
start

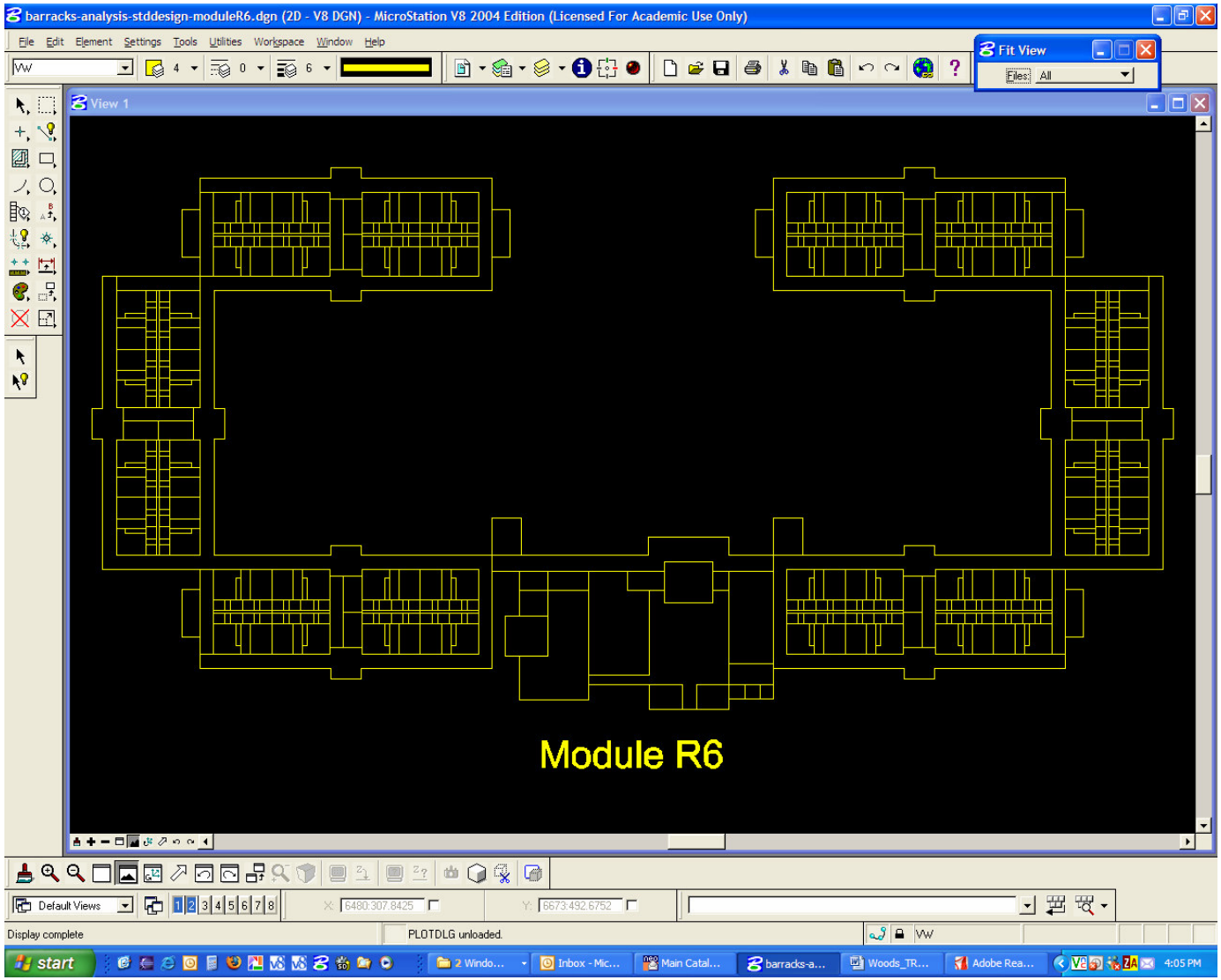

barracks-analysis-stddesign-moduleR1.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
Fit View
Files: All
View 1
Module R1
Default Views
Display complete
PLOTDLG unloaded.
start
4:02 PM


barracks-analysis-stddesign-moduleR2.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
Fit View
Files All
View 1
Module R2
Default Views
Display complete
PLOTDLG unloaded.
start
4:03 PM


barracks-analysis-stddesign-moduleR3.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
Fit View
Files All
View 1
Module R3
Default Views
Display complete
PLOTDLG unloaded.
start


barracks-analysis-stddesign-moduleR4.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
Fit View
Files: All
View 1
Module R4
Default Views
Display complete
PLOTDLG unloaded.
start
4:04 PM


barracks-analysis-stddesign-moduleR5.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
Fit View
Files All
View 1
Module R5
Fit View
Default Views
Display complete
PLOTDLG unloaded.
start
4:05 PM


barracks-analysis-stddesign-moduleR6.dgn (2D - V8 DGN) - MicroStation V8 2004 Edition (Licensed For Academic Use Only)
File Edit Element Settings Tools Utilities Workspace Window Help
Fit View
View 1
Module R6
Default Views
Display complete
PLOTDLG unloaded.
start

8 APPENDIX: QFB

QFB Encoder-Analyzer

An automated Qualitative Feature Based (QFB) representation analysis Encoder-Analyzer (EA) was provided from Jupp and Gero from the University of Sydney, Australia. Input data for the Encoder/Analyzer was prepared using MicroStation v08.05.02.27 by tracing the spatial centerlines from each building in the case base using closed shapes (rectangles and line strings, specifically). A QFB-EA output file generator was developed which produces a file in the QFB file format from the spatial representations in MicroStation.

Source Code for QFB Output File Generator

```
Option Explicit

Sub Module1_Initialize()
    UserForm1.Show
End Sub


Sub processDrawing()
    Dim oScanCriteria As ElementScanCriteria
    Set oScanCriteria = New ElementScanCriteria

    oScanCriteria.ExcludeAllTypes
    oScanCriteria.IncludeType msdElementTypeShape

    Dim oScanEnumerator As ElementEnumerator
    Set oScanEnumerator = ActiveModelReference.Scan(oScanCriteria)

    Dim oElement As ShapeElement
    Dim vertices() As Point3d
    Dim vX, vY, minX, minY As Long
    Dim i As Integer

    minX = 99999999
    minY = 99999999

    'find minX and minY to use to translate all vertices
    Do While oScanEnumerator.MoveNext
        Set oElement = oScanEnumerator.Current
        'If oElement.IsVertexList Then
            'With oElement.AsVertexList
                'Set vList = oElement
                'vertices = vList.GetVertices
                vertices = oElement.GetVertices
```

```
            For i = LBound(vertices) To UBound(vertices)
                vX = CInt(vertices(i).X)
                If vX < minX Then
                    minX = vX
                End If

                vY = CInt(vertices(i).Y)
                If vY < minY Then
                    minY = vY
                End If
            Next
        'End With
    'End If
Loop

minX = minX - 20
minY = minY - 20

'flag for processing shapes as polygons or lines
Dim asPolygons As Boolean
asPolygons = True

'create output file
Dim fileNum As Long
Dim fileName As String
fileNum = FreeFile

If asPolygons Then
    fileName = "-byPolys"
Else
    fileName = "-byLines"
End If
fileName = "C:\QFBModel\qfb files\" + Mid$(ActiveDesignFile.Name, 1,
           Len(ActiveDesignFile.Name) - 4) + fileName + ".qfb"
Open fileName For Output As #fileNum

Dim vertexStr As String
Dim vertexVal As Integer

oScanEnumerator.Reset

'make polygon list using polygons
If asPolygons Then
    Print #fileNum, "Lines"
    Print #fileNum, "Polygons"
    Do While oScanEnumerator.MoveNext
        Set oElement = oScanEnumerator.Current
        vertices = oElement.GetVertices
        'make polygon list by polygon vertices
        For i = LBound(vertices) To UBound(vertices)
            vertexStr = vertices(i).X
            vertexVal = CInt(vertexStr)
            vertexVal = vertexVal - minX
            Print #fileNum, CStr(vertexVal)

            vertexStr = vertices(i).Y
            vertexVal = CInt(vertexStr)
            vertexVal = vertexVal - minY
            Print #fileNum, CStr(vertexVal)
        Next
    Loop
' make polygon list by line vertices
' note: bug in EA when utilizing Lines approach
Else
    Print #fileNum, "Lines"
```

```
        Do While oScanEnumerator.MoveNext
            Set oElement = oScanEnumerator.Current
            vertices = oElement.GetVertices
            For i = LBound(vertices) To UBound(vertices) - 1
                vertexStr = vertices(i).X
                vertexVal = CInt(vertexStr)
                vertexVal = vertexVal - minX
                Print #fileNum, CStr(vertexVal)

                vertexStr = vertices(i).Y
                vertexVal = CInt(vertexStr)
                vertexVal = vertexVal - minY
                Print #fileNum, CStr(vertexVal)

                vertexStr = vertices(i + 1).X
                vertexVal = CInt(vertexStr)
                vertexVal = vertexVal - minX
                Print #fileNum, CStr(vertexVal)

                vertexStr = vertices(i + 1).Y
                vertexVal = CInt(vertexStr)
                vertexVal = vertexVal - minY
                Print #fileNum, CStr(vertexVal)
            Next
        Loop
        Print #fileNum, "Polygons"
    End If

    Close #fileNum
End Sub
```

9 APPENDIX: Rectangle Code

Description

The following MicroStation/J code exhibits the same behavior as the detailed example pattern RectanglePattern in Section 3. Note the verbosity compared to the pattern implementation required to get the same behavior.

```
/*-------------------------------------------------------------------------+
|                                USACERL                                  |
|               Engineering Research and Development Center               |
|              US Army Construction Engineering Research Labs             |
|                             Champaign, IL                               |
|                                                                         |
|     FILE NAME:    CADPlaceSpaceRectangularCommand                       |
|                                                                         |
|     APPLICATION:  Building Composer                                     |
|                                                                         |
|     PROGRAMMERS:  Van Woods, William Zwicky                             |
|                                                                         |
|                                                                         |
+-------------------------------------------------------------------------*/

+-------------------------------------------------------------------------=>*/

package LayoutComposer.MSJ.Commands;


/*-------------------------------------------------------------------------+
|
|   Imports
|
+-------------------------------------------------------------------------*/
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.io.*;
import java.text.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;

import com.bentley.dgn.*;

import SDAI.SIfc2x_final.*;
import SDAI.COM.steptools.SIfc2x_final.*;

import BCCore.BuildingComposer;
import BCCore.GUIBeans.*;
import BCCore.Utils.*;

import LayoutComposer.*;
import LayoutComposer.Common.*;
import LayoutComposer.Common.Controllers.*;
import LayoutComposer.Common.Events.*;
import LayoutComposer.Common.Exceptions.*;
import LayoutComposer.Common.Geometry.*;
import LayoutComposer.MSJ.Geometry.*;
import LayoutComposer.MSJ.Controllers.*;



/*-------------------------------------------------------------------------**//**
* Dynamics-only interface for Place Space Rectangular command.  This collects
* parameters, then sends them in an event to BC for processing.
*
+---------------+---------------+---------------+---------------+-------------*/
public class CADPlaceSpaceRectangularCommand extends PrimitiveCommand {
     // Spaces of this size or smaller will be rejected.
     public static final double ZERO_AREA = 0.00001;

     /** true if everything has been initialized satisfactorily (params valid, cell exists, etc.) */
     private boolean bIsValid = false;

     private DPoint m_startPoint = null;

     private LayoutComposer.Common.Geometry.Area m_area = null;
```

```
/** Descriptive name for GUI friendly messages.  Space will be created with this name. */
private String m_spaceName = "";

/** Space name requested by caller.  Space may get a different name. */
private String m_functionName = "";

/**
 * GUID of space being placed, OR function that will be instantiated.
 * m_guid can be either a space or a function, but it's not a problem cuz we don't
 * currently draw shapes for functions.
 */
private String m_guid = null;


/*-----------------------------------------------------------------
*   Space Criteria
*-----------------------------------------------------------------*/

 //We don't currently support hard limits; if m_bLockSize is true, we lock space
 //size to Min values.  If we add  support for hard limits where Min != Max, we'll
 //want to add m_bLimitRange, and reconsider m_bLockSize by testing Min vs. Max.
 //private boolean m_bLimitRange;    //not used
 private boolean m_bLockSize;        //locks space size to Min values.
 private double  m_lengthMin;        //permissible X size range
 private double  m_lengthMax;        // "
 private double  m_widthMin;         //permissible Y size range
 private double  m_widthMax;         // "


/*-----------------------------------------------------------------
*   Graphic and related data
*-----------------------------------------------------------------*/

/** Solution graphic name (within default library for this project) */
 private String  m_graphic;

/** List of available solutions for this space */
private FunctionSizes m_sizes;

/** The graphic itself, in the *wrong* place (i.e. exactly as copied from lib) */
private CellElement m_cellElement = null;

/** Placement handle for graphic cell.  May be different from m_spaceHandle, we need to compensate. */
private Point3d m_cellOrigin;

/** Where to place cell in drawing so it appears on top of space */
private Point3d m_cellTranslate;


/*-----------------------------------------------------------------
*   Basic Topology
*-----------------------------------------------------------------*/

 /**
 * Generated perimeter geometry.  If generated from a cell, this perim will be in the
 * *right* place, so only the cell needs to be tranlated into place.
 */
 private LayoutComposer.Common.Geometry.Polygon m_geo = new LayoutComposer.Common.Geometry.Polygon();

 //not the same as length, as these can be negative;
 //must be used in conjunction with cw or ccw considerations
 private double m_xSize = 0;
 private double m_ySize = 0;
 private double m_zSize = 0;   //this needs to be tied into the default value in ObjModel
 private double m_angle = 0;


/*-----------------------------------------------------------------
*   Space Placement Data
*-----------------------------------------------------------------*/

 /** Determines location of rectangle relative to origin.  Does NOT determine mirroring or rotation. */
 private DPoint m_quadrant = new DPoint();
 /** Determines mirroring of space within area given my m_geo. */
 private DPoint m_mirror = new DPoint();


 //tool settings
 private ToolPanel toolPanel;
 private ToolSettingsDlg toolDlg = null;

 //utils
 private CADCommandsController m_controller = null;
 private ElementCommandsControllerMSJ m_eleCommands = null;
 private SystemCommandsControllerMSJ m_CADSystem = null;
 private ObjectModelControllerLC m_omCtl = null;

 /** Properties for graphics in dynamics. */
 private HierarchicalProperties s_repDynProps;

 /** Element drawn during dynamics, to make erasing easier */
 private Element dynamics_lastDrawn = null;

 /** Test incoming parameters before saving to database */
 private ValidateMSJ m_validator = new ValidateMSJ();
```

```
     //parameters (should be read from a .properties file)
     // -general
     private boolean p_swipeFocus = true;

     private boolean m_commandCompleted = false;


/*--------------------------------------------------------------------------**//**
*   Constructors
+---------------+---------------+---------------+---------------+-------------*/
     public CADPlaceSpaceRectangularCommand(CADCommandsController controller) {
          m_controller = controller;
          m_eleCommands = new ElementCommandsControllerMSJ();
          m_CADSystem = new SystemCommandsControllerMSJ();
          m_omCtl = new ObjectModelControllerLC();

          //Load properties for graphics in dynamics.
          s_repDynProps = new
HierarchicalProperties(LayoutComposerApp.s_SLSystemProperties,"representation.placement.dynamics");
     }

//temporary methods to register command for testing purposes
     /*
     public CADPlaceSpaceRectangularCommand() {
               toolDlg = ToolSettingsDlg.getInstance();
     }
     public static void  main ( String args[] ) {
          CADPlaceSpaceRectangularCommand cmd = new CADPlaceSpaceRectangularCommand ();
          cmd.setKeyin ("PlaceSpaceRectangular");
          DgnKernel.registerCommand (cmd);
          System.out.println("registered PlaceSpaceRectangular");
     }
     */


/*--------------------------------------------------------------------------**//**
*
*  Local Methods
*
+---------------+---------------+---------------+---------------+-------------*/

     //============================================================================
     /**
      * Instances are recycled; this must clean up for new run.
      * @param spaceName descriptive name for space
      * @param guid database object that we're going to place
      * @param zSize overall space height
      */
     public void setCommandParams(String functionName, String guid, double zSize) {
        m_functionName = functionName;
          m_guid = guid;
          m_zSize = zSize;

          m_lengthMin = 0;
          m_lengthMax = 0;
          m_widthMin  = 0;
          m_widthMax  = 0;
          m_graphic   = null;
          m_bLockSize = false;

          bIsValid = true;

          //disable graphic rendering
          s_repDynProps.set("graphic", "0");

        m_sizes = getPredefinedSpaces();
        if( toolPanel != null )
            toolPanel.reset( m_sizes );
     }



     /**
      * Set up for placement of fixed-size space.
      * Instances are recycled; this must clean up for new run.
      *
      * @param zSize overall space height
      * @param lengthMin lengthMax: range of length.  if same, length is disabled in dialog
      * @param widthMin widthMax: range of width.  if same, width is disabled in dialog
      * @deprecated length/width ranges are going away; perimeter will be derived from graphic.
      */
     public void setCommandParams(String spaceName, String guid, double zSize,
                                  double lengthMin, double lengthMax, double widthMin, double widthMax ) {
          setCommandParams(spaceName, guid, zSize);

          m_lengthMin = lengthMin;
          m_lengthMax = lengthMax;
          m_widthMin  = widthMin;
          m_widthMax  = widthMax;
          m_graphic   = null;
          m_bLockSize = true;

          bIsValid = true;
     }
```

```
    protected FunctionSizes getPredefinedSpaces() {
        MDSAppInst[] fiAndStory = m_omCtl.getFunctionInstance(m_guid);
        if (fiAndStory!=null && fiAndStory[0]!=null ) {
            // 'tis a function; get its code
            String code = m_omCtl.getFunctionID( (MDSFunction)fiAndStory[0] );
            return getPredefinedSpaces(code);
        }
        // quietly ignore everything else
        return null;
    }


    protected FunctionSizes getPredefinedSpaces( String functionID ) {
        FunctionSizes sizes = null;

        try {
            GraphicManager.initGraphicLib();
            Object scanner = FunctionSizes.buildScanner();
            if( scanner == null )
                return null;
            sizes = FunctionSizes.getSizes(scanner, functionID);
        }
        catch (IOException ex) {
            // bad/missing file .. ignore
        }
        catch (java.lang.Exception ex) {
            ex.printStackTrace();
        }

        return sizes;
    }


/*-------------------------------------------------------------------------**//**
*
*  JMDL Methods
*
+---------------+---------------+---------------+---------------+-------------*/

     /*-----------------------------------------------------------------------------+
     *  Called when this Command is made active.
     +---------------+---------------+---------------+---------------+-------------*/
     public void start(String oStrUnparsed) {
          if( !bIsValid ) {
               m_CADSystem.showErrorMsg("Space cannot be placed.");
               abort();
               throw new RuntimeException( "(placeSpaceRect.start) command aborted; some parameter is not valid."
);
          }

          m_area = new LayoutComposer.Common.Geometry.Area(0.0, m_omCtl.getUnits().startsWith("Eng") ?
m_area.English : m_area.Metric);

        if( toolPanel == null ) {
            toolDlg = ToolSettingsDlg.getInstance();
            toolPanel = new ToolPanel( m_sizes );
            toolPanel.setSpanValue( new SpaceStorySpan("1 story") );
            toolPanel.setAngleValue( m_angle );
        }

          //lock length and width if requested
          if (m_bLockSize) {
               if (m_lengthMin == m_lengthMax) {
                    // lock length
                    toolPanel.setLengthValue( m_lengthMin );
                    toolPanel.setLenLocked( true );
                    toolPanel.setLenEnabled( false );
               }
               else {
                    // unlock length
                    toolPanel.setLenLocked( false );
                    toolPanel.setLenEnabled( true );
               }

               if (m_widthMin == m_widthMax) {
                    // lock width
                    toolPanel.setWidthValue( m_widthMin );
                    toolPanel.setWidthLocked( true );
                    toolPanel.setWidthEnabled( false );
               }
               else {
                    // unlock width
                    toolPanel.setWidthLocked( false );
                    toolPanel.setWidthEnabled( true );
               }
          }
          else {
               // unlock all
               toolPanel.setLenLocked( false );
               toolPanel.setLenEnabled( true );
               toolPanel.setWidthLocked( false );
               toolPanel.setWidthEnabled( true );
          }

          //Is element already placed?
          if (m_eleCommands.findElement(m_guid) == true) {
```

```
               //System.out.println("(placeSpaceRect.start) command aborted; a space with this GUID already
exists.");
               m_CADSystem.showErrorMsg("Space already placed.");
               abort();
               throw new RuntimeException( "(placeSpaceRect.start) command aborted; a space with this GUID already
exists." );
          }

          //Show tool panel, give it focus
          toolDlg.showDlg(toolPanel, "Place Space");
          if (p_swipeFocus)
               toolDlg.requestFocus();

          m_startPoint = null;
        m_cellElement = null;

          //don't change mode
          //if (accudrawHints.isActive())
          //    accudrawHints.setRectangular();

          //wipe old overlap highlights
          // -only necessary because stop() is not called when a new command is started
          m_validator.startNewHighlights();

          m_CADSystem.showCommandMsg("(" + m_functionName + ") Place Space Module");
          m_CADSystem.showPromptMsg("Enter start point");

          //start dynamics immediately; if length and width are locked, we'll have something to show
          startDynamics();
     }



     /*-------------------------------------------------------------------------**//**
     *  Called when a data point occurs in a view during the execution of this Command.
     *  @param        oPoint  the data point
     *  @param        oView   the view where this data point occurred.
     +---------------+---------------+---------------+---------------+-------------*/
     public void dataPoint(DPoint oPoint, View oView) {
          updateToolSettings( oPoint, oView );

          // Can we commit?  Yes, if:
          // -this is 2nd data point, or
          // -width and length are locked, or
          // -user is placing a cell.
          if (m_startPoint != null
            || (toolPanel.isLenLocked() && toolPanel.isWidthLocked())
            || m_cellElement != null) {

               DPoint[] msjVerts = Convert.toMSJ(m_geo.getVertices());
               ShapeElement msjPerim;

               {
                    //Test if space is valid on its own terms.
                    //This should be moved into
                    //  Validate.isValidSpace(m_guid, Convert.toMSJ(m_geo.getVertices())
                    //Some other tests we'll want:
                    //  doesn't overlap self
                    //  no dup points at all

                    try {
                         msjPerim = new ShapeElement(null, msjVerts, msjVerts.length);
                    }
                    catch( Element.Exception ex ) {
                         m_CADSystem.showErrorMsg("INVALID SHAPE: Space perimeter is not a valid shape.");
                         Toolkit.getDefaultToolkit().beep();
                         return;
                    }

                    //Area must be > 0
                    if( msjPerim.getArea(0) <= ZERO_AREA ) {
                         m_CADSystem.showErrorMsg("INVALID PLACEMENT: Space cannot have zero area.");
                         Toolkit.getDefaultToolkit().beep();
                         return;
                    }
               }

               try {
                    //Test if space is valid relative to rest of building.
                    boolean isValid = m_validator.isValidInPlan( m_guid, msjVerts );

                    // Not valid -> complain
                    if (! isValid) {
                         Toolkit.getDefaultToolkit().beep();
                         m_CADSystem.showErrorMsg("INVALID PLACEMENT: Overlapping placement of spaces is not
allowed.");
                    }

                    // Valid -> commit
                    else {
                         // don't set the accudraw distance hint
                         // MicroStation only sets this for chained placement (like line strings)
                         //if (accudrawHints.isActive())
                         //   accudrawHints.setDistance(m_endPoint.distance(m_startPoint));

                         // Geometry is complete; inform main program
                         notifyObjectCreated();
```

```
                    // restart this command
                    abort();
                    }
               }
               catch (java.lang.Exception e) {
                    e.printStackTrace();
                abort();
               }
          }

          // Not enough info yet
          else {
               // save off start point
               m_startPoint = oPoint;

               // keep dynamics enabled
               startDynamics();

               // change prompt
               m_CADSystem.showPromptMsg("Enter opposite corner");

               // keep focus on the tool settings
               if (p_swipeFocus)
                    toolDlg.requestFocus();
          }
     }


   /*-------------------------------------------------------------------------**//**
    * Called when the cursor is moved over a view during the execution of this command.
    * <P>
    * Some notes on dynamics:  We call updateToolSettings which calls JTextField.setText,
    * which <I>may</I> result in a call to wait().  However, Microstation ignores the
    * state of a thread, and will force another call to dynamics.  The resulting stack
    * trace contains a call to dynamics(), followed by a call to wait(), followed by
    * another call to dynamics().  Obviously, this causes no end of trouble for use, and
    * is made worse by the fact that it can't be prevented with Java code:
    * <UL>
    *   <LI> The <I>synchronized</I> keyword is useless, as Microstation recycles the
    *        same thread and simply injects a new function call.  Since the thread
    *        already has the lock, it's not stopped.
    *   <LI> A boolean lock flag would work, but embedded sleep() calls (if present) will
    *        never terminate if the delay is longer than 32 ms.  A forced call to
    *        dynamics() is not even necessary in this case.
    * </UL>
    * As of this writing (7 May 2003) Bentley has sent me (Bill Zwicky) two .dll files
    * which appear to fully fix these problems.  Hopefully these will be included in
    * the next update of Microstation.
    *
     * @param        oPoint     the new cursor position in Model coordinates.
     * @param        oView      the view that the cursor is currently in.
     * @param        bDrawMode  true indicates that dynamics geometry should be drawn.
     *                          false indicates that geometry should be erased.
     +---------------+---------------+---------------+---------------+-------------*/
    public synchronized void dynamics(DPoint oPoint, View oView, boolean bDrawMode) {
       PlanView_Space_MSJ rep = null;

         updateToolSettings( oPoint, oView );

         if (bDrawMode) {
              //test if we have anything to draw yet
           if( m_cellElement != null ) {
               rep = new PlanView_Space_MSJ( m_spaceName, m_geo, m_zSize, m_graphic, Convert.toSL(oPoint),
m_angle, m_mirror.x<0, m_mirror.y<0 );
           }
              else if (m_geo.getVertices() != null) {
               rep = new PlanView_Space_MSJ( m_spaceName, m_geo, m_zSize );
              }

           if( rep != null ) {
               try {
                    // ------
                    // If dynamics_lastDrawn is not null, I don't really care, it just
                    // means MS forgot to call us to erase element.
                    // ------
                    rep.setProperties( s_repDynProps );
                    dynamics_lastDrawn = rep.construct();
                    DgnKernel.session.redrawElement( dynamics_lastDrawn, View.XORDRAW );
               }
               catch (Element.Exception e) {
                    // do nothing in dynamics
               }
           }
         }
         else {
              // --- Erase if not yet erased --- //
              if (dynamics_lastDrawn != null) {
                   DgnKernel.session.redrawElement (dynamics_lastDrawn, View.XORDRAW);
                   dynamics_lastDrawn = null;
              }
         }
     }
```

```
     /*---------------------------------------------------------------------------**//**
     * Called when the user hits right button
     +---------------+---------------+---------------+---------------+-------------*/
     public void reset() {
          // Emulate default MicroStation command behavior.
          // If command is in process, then the only thing to cancel the command is
          // to select a different command, else a reset restarts the command.
          restart();
     }



     /*---------------------------------------------------------------------------**//**
     * Call this to terminate the current command.  Note this will kill ANY active
     * command, this one need not be active.
     +---------------+---------------+---------------+---------------+-------------*/
    public void abort() {
          DgnKernel.startDefaultCommand();
    }



     /*---------------------------------------------------------------------------**//**
     * Callback called by MSJ when this Command is terminated.  DO NOT CALL!
     * Use abort(), or any DgnKernal function command to stop or start commands.
     +---------------+---------------+---------------+---------------+-------------*/
     public void stop( ) {
          if (BuildingComposer.developerVersion) System.out.println("CADPlaceSpaceRectangularCommand:stop()
called");

          //clear overlap highlights
          // ** This hangs MSJ V7!  Defect submitted to Bentley.
          // ** This now works in V8, however, stop() isn't always called.
          //    For example, clicking the MS native Delete button will skip this function.
          m_validator.startNewHighlights();

          m_startPoint = null;
          toolDlg.setVisible(false);

          if (!m_commandCompleted) {
               notifyAborted();
          }

     }



/*---------------------------------------------------------------------------**//**
*
*  Event methods
*
+---------------+---------------+---------------+---------------+-------------*/

     private Vector m_CADObjectEventListeners = new Vector();

     public synchronized void addCADObjectEventListener(CADObjectEventListener listener) {
          if (!m_CADObjectEventListeners.contains(listener)) {
               m_CADObjectEventListeners.addElement(listener);
          }
     }


     public synchronized void removeCADObjectEventListener(CADObjectEventListener listener) {
          if (m_CADObjectEventListeners.contains(listener)) {
               m_CADObjectEventListeners.removeElement(listener);
          }
     }


     protected void notifyObjectCreated() {
          m_commandCompleted = true;

          CADObjectCreatedEvent evt;

          //store data to reside in object model
          evt = new CADObjectCreatedEvent(this);

        evt.setFunctionName(m_spaceName);
          evt.setGUID(m_guid);

        evt.setSpaceName( m_spaceName );
          evt.setPoints( m_geo.getVertices() );

          evt.setSpan( toolPanel.getSpanValue().toString() );
          evt.setMirrorX( m_mirror.x < 0 );
          evt.setMirrorY( m_mirror.y < 0 );
          //--ceiling height is disabled in toolPanel--
          //evt.setHeight( Double.toString(toolPanel.getCeilingHtValue()) );
          evt.setHeight( null );

        // Fill in data for custom graphic cell
        if( m_cellElement != null ) {
            evt.setSolutionGraphic( m_graphic );
            evt.setAngle( m_angle );
        }

          evt.setCompletionStatus(CADObjectCreatedEvent.COMMAND_COMPLETED);
```

```
     //inform listeners of new object
     Vector v;
     synchronized (this) {
          //preserve current state
          v = (Vector)m_CADObjectEventListeners.clone();
     }

     int cnt = v.size();
     for (int i = 0; i < cnt; ++i) {
          CADObjectEventListener client = (CADObjectEventListener)v.elementAt(i);
          client.objectCreated(evt);
     }
}

protected void notifyAborted() {
     m_commandCompleted = false;

     CADObjectCreatedEvent evt;

     //store data to reside in object model
     evt = new CADObjectCreatedEvent(this);

     evt.setGUID(m_guid);
     evt.setCompletionStatus(CADObjectCreatedEvent.COMMAND_ABORTED);

     //inform listeners of new object
     Vector v;
     synchronized (this) {
          //preserve current state
          v = (Vector)m_CADObjectEventListeners.clone();
     }

     int cnt = v.size();
     for (int i = 0; i < cnt; ++i) {
          CADObjectEventListener client = (CADObjectEventListener)v.elementAt(i);
          client.objectCreated(evt);
     }
}




/*-----------------------------------------------------------------------------+
*  Construct a rotated rectangle, return the corners.
*  p1-p2 are opposite corners; other two corners will be placed so that rect's
*  +X axis (from p1) is at given angle to drawing's.
*  Z-coord is ignored on input, and 0 on output.
*  output is CCW
*  points[0] is LL corner of rect rotated to angle=0
*  points[0] -> points[1] always points in direction of 'angle'
+---------------+---------------+---------------+---------------+--------------*/
private static DPoint[] constructRectanglePoints
(
DPoint      p1,
DPoint      p2,
double      angle
)
{
     Point2D.Double   p2rot;
     DPoint        [] pts;
     Point2D.Double[] p2d;
     AffineTransform  atTrans;
     double           xmin, xmax, ymin, ymax;

     /* --- Build transform --- */
     atTrans = AffineTransform.getRotateInstance ( angle, p1.x, p1.y );

     /* --- Build rectangle --- */
     // -un-rotate corner
     p2rot = new Point2D.Double ( p2.x, p2.y );
     try {
          atTrans.inverseTransform ( p2rot, p2rot );
     }
     catch (NoninvertibleTransformException e) {
          e.printStackTrace();
     }

     // -find LL and UR corners
     if (p1.x < p2rot.x) {
          xmin = p1.x;
          xmax = p2rot.x;
     }
     else {
          xmin = p2rot.x;
          xmax = p1.x;
     }
     if (p1.y < p2rot.y) {
          ymin = p1.y;
          ymax = p2rot.y;
     }
     else {
          ymin = p2rot.y;
          ymax = p1.y;
     }
```

```
     // -build rectangle
     p2d = new Point2D.Double [ 4 ];
     p2d[0] = new Point2D.Double ( xmin, ymin );
     p2d[1] = new Point2D.Double ( xmax, ymin );
     p2d[2] = new Point2D.Double ( xmax, ymax );
     p2d[3] = new Point2D.Double ( xmin, ymax );

     /* --- Rotate everything --- */
     atTrans.transform ( p2d, 0, p2d, 0, p2d.length );

     /* --- Convert from Java to MicroStation type --- */
     pts = new DPoint [ 4 ];
     pts[0] = new DPoint ( p2d[0].x, p2d[0].y );
     pts[1] = new DPoint ( p2d[1].x, p2d[1].y );
     pts[2] = new DPoint ( p2d[2].x, p2d[2].y );
     pts[3] = new DPoint ( p2d[3].x, p2d[3].y );

     return pts;
}




/**
 * updates tool settings dialog and local geometry members.  If tool settings are locked,
 * they're used to determine geometry.  If not, cursor determines geometry, and tool settings
 * are updated accordingly.  Note that this function runs in two modes:
 * <UL>
 *   <LI>If m_startPoint == null and length and width are locked, then the rectangle is
 *       computed at oPoint in the +X+Y quadrant, relative to current angle.  After the
 *       first click, user will be able to change quadrant.
 *   <LI>After m_startPoint has been chosen, then oPoint is used to set the size of unlocked
 *       axes, and the direction of locked axes.
 * </UL>
 *
 * Local variables that are updated:
 * <UL>
 *   <LI> m_angle
 *   <LI> m_xSize
 *   <LI> m_ySize
 *   <LI> m_geo
 *   <LI> m_area
 * </UL>
 *
 * @param oPoint Potenetial end point.  Determines length and width if they're not
 *     locked in the tool panel; determines quadrant in any case.
 * @param oView View that <I>oPoint</I> is in.
 */
private void updateToolSettings( DPoint oPoint, View oView ) {
     double xSize, ySize;                        //computed length and width
     DPoint startPoint;                          //pointer to correct startPoint
     DPoint endPoint = new DPoint(oPoint);   //copy of endPoint so we can adjust it

   // First check for solution graphic
   setGraphic( toolPanel.getGraphic() );

   // Mangle the space name if a cell is selected
   if( m_cellElement != null ) {
       m_spaceName = toolPanel.getGraphicDescription();
   }
   else {
       m_spaceName = m_functionName;
   }

     // --- Fetch and update angle --- //
     if (toolPanel.isAngleLocked())
          // if locked, fetch
          m_angle = toolPanel.getAngleValue();
     else {
          // if not locked, get MSJ's angle, and display in dialog
          RMatrix          rm;
          DPoint           dpTmp = new DPoint();

          /* --- Fetch ACS --- */
          rm = accudrawHints.getRMatrix ( oView );

          /* --- Fetch current angle --- */
          //returns [0,pi); see docs
          m_angle = rm.getRotationAngleAndVector ( dpTmp );
          //fix to [0,2*pi)
          if (dpTmp.z > 0)
               m_angle = 2 * Math.PI - m_angle;
     }


     // --- Fetch mirror status --- //
     m_mirror.x = ( toolPanel.getMirrorX() ? -1 : +1 );
     m_mirror.y = ( toolPanel.getMirrorY() ? -1 : +1 );
     m_mirror.z = +1;


     // --- If start point already clicked, ... --- //
     if (m_startPoint != null) {
          //Use it
          startPoint = m_startPoint;
```

```
          }
          else {
               //If perim is already determined, then let user drag it around
               if( m_graphic != null || (toolPanel.isLenLocked() && toolPanel.isWidthLocked()) ) {
                    startPoint = oPoint;
               }
               //Not locked, so erase everything else
               else {
                    m_xSize = 0;
                    m_ySize = 0;
                    m_area.setArea(0);
                    m_quadrant.x = 1;
                    m_quadrant.y = 1;
                    m_quadrant.z = 0;
                    m_geo.setVertices(null);
                    return;
               }
          }


          // --- If we're placing a graphic cell, then perim has already been provided --- //
          if( m_cellElement != null ) {
            // Get fresh perim
            m_cellOrigin = new Point3d();
            m_geo = GraphicManager.extractPerim( m_cellElement, m_cellOrigin );

            // translate perim into position
               Point3d[]       verts = m_geo.getVertices();
               Point3d[]       newVerts = new Point3d[verts.length];
            AffineTransform atTrans;
            Point2D.Double  temp = new Point2D.Double();

               // rotate to requested angle, translate to cursor
               atTrans = AffineTransform.getRotateInstance ( m_angle, startPoint.x, startPoint.y );
               atTrans.translate( startPoint.x - m_cellOrigin.x, startPoint.y - m_cellOrigin.y );
               for( int i=0; i<verts.length; i++ ) {
                    temp.x = verts[i].x;
                    temp.y = verts[i].y;
                    atTrans.transform ( temp, temp );
                    newVerts[i] = new Point3d( temp.x, temp.y, verts[i].z );
               }
               m_geo.setVertices( newVerts );
          }

          // --- Otherwise, build a new rectangle --- //
          else {
               // --- Compute xSize and ySize, even if rotated. --- //
               // Note that endPoint and startPoint are in drawing's coord system, but
               // xSize and ySize need to be computed in rectangle's.
               {
                    // project endPoint onto rectangle's +X
                    //all is translated to (0,0) to make it easier
                    Point2D.Double dpPt = new Point2D.Double( endPoint.x - startPoint.x, endPoint.y - startPoint.y
);
                    Point2D.Double dpAxis = new Point2D.Double( Math.cos(m_angle), Math.sin(m_angle) );
                    Point2D.Double dpProj;

                    if (toolPanel.isLenLocked()) {
                         m_xSize = toolPanel.getLengthValue();
                    }
                    else {
                         // xSize is just size of projected vector.
                         try {
                              dpProj = GeoToolJ.project( dpPt, dpAxis );
                              xSize = Math.abs( dpProj.distance( 0, 0 ) );
                         }
                         catch (GeometryException ex) {
                              xSize = 0;
                         }
                         m_xSize = xSize;
                    }

                    // project endPoint onto rectangle's +Y; ySize is size of projected vector.
                    dpAxis.x = Math.cos(m_angle + Math.PI/2);
                    dpAxis.y = Math.sin(m_angle + Math.PI/2);

                    if (toolPanel.isWidthLocked()) {
                         m_ySize = toolPanel.getWidthValue();
                    }
                    else {
                         try {
                              dpProj = GeoToolJ.project( dpPt, dpAxis );
                              ySize = Math.abs( dpProj.distance( 0, 0 ) );
                         }
                         catch (GeometryException ex) {
                              ySize = 0;
                         }
                         m_ySize = ySize;
                    }
               }

            // --- //
            // Update the dialog.
            // This really needs to be done here in-thread, the next call to this
            // function will need these values.  Though Swing is not generally thread-
            // safe, JTextField.setText() specifically is, so we're ok.
            // --- //
```

```
        // --- Update xSize --- //
        if (!toolPanel.isLenLocked()) {
            toolPanel.setLengthValue(m_xSize);
        }
        // --- Update ySize --- //
        if (!toolPanel.isWidthLocked()) {
            toolPanel.setWidthValue(m_ySize);
        }
        toolPanel.setAngleValue(m_angle);

           // --- Fetch and update area --- //
           m_area.setArea ( Math.rint(Math.abs( m_xSize * m_ySize )*1000)/1000 );


           // --- Compute quadrant only if user clicked a start point --- //
           if (m_startPoint != null) {
                // ------
                // What quadrant is the cursor in?  This will determine which corners
                // the user clicked, and which direction to draw the rectangle relative
                // to m_startPoint.
                //   This is computed by examining the angle that m_startPoint -> endPoint
                // (the diagnoal) makes to the +X axis, after rotating the rectangle by
                // -m_angle (making it orthogonal.)
                // ------
                DPoint plusX = new DPoint ( 1, 0 );       //+X axis
                DPoint vec   = new DPoint ( endPoint.x - m_startPoint.x, endPoint.y - m_startPoint.y );
                double q;

                // Compute angle, relative to requested rotation, in range 0 .. 2pi
                q = plusX.angleToXY ( vec );
                q = q - m_angle;
                q = Math.IEEEremainder( q, 2 * Math.PI );
                if (q < 0)     q += 2 * Math.PI;

                // Decipher angle
                m_quadrant.z = 0;
                if (q <= Math.PI / 2) {
                     m_quadrant.x = 1;
                     m_quadrant.y = 1;
                }
                else if (q <= Math.PI) {
                     m_quadrant.x = -1;
                     m_quadrant.y = 1;
                }
                else if (q <= 3 * Math.PI / 2) {
                     m_quadrant.x = -1;
                     m_quadrant.y = -1;
                }
                else {
                     m_quadrant.x = 1;
                     m_quadrant.y = -1;
                }
           }
           else {
                m_quadrant.x = 1;
                m_quadrant.y = 1;
           }

           // Build outline shape
           m_geo.setVertices( Convert.toSL( GeoToolMSJ.constructRectangleLWA( startPoint,
                m_xSize*m_quadrant.x, m_ySize*m_quadrant.y, m_angle ) ) );
      }
 }



/**
 * Attach this cell graphic to the space being placed.
 * @param gname Name of cell graphic to attach.
 */
public void setGraphic( String gname ) {
    // handle nulls as legit values
    if( m_graphic != gname || ( m_graphic != null && ! m_graphic.equals(gname)) ) {
        m_graphic = gname;

        // reset members
        m_cellElement = null;
        m_cellOrigin = null;
        m_geo.setVertices(null);

        // reset command
        m_startPoint = null;

        if( m_graphic != null ) {
            //enable graphic rendering
            s_repDynProps.set("graphic", "1");

            try {
                // Get cell
                GraphicManager.initGraphicLib();
                m_cellElement = GraphicManager.getGraphic( m_graphic );
            }
            catch( java.lang.Exception ex ) {
                // cell not found
                ex.printStackTrace();
            }
```

```
            }
            else {
                //disable graphic rendering
                s_repDynProps.set("graphic", "0");
            }
        }
    }



//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////
////       Tool Panel
////
////
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/*-------------------------------------------------------------------------------------+
|
|   This class constructs a command-specific panel to be displayed in the container class ToolSettingsDlg
|
+-------------------------------------------------------------------------------------*/
//if dialog is closed, kill the command (restartDefaultCommand)
//need to work on behavior to mimic accudraw
//start command, track dynamics values (display at 2 decimal places, system setting)
//if shape in dynamics is longer than wide, give dialog length the focus; same for width; user should not ever
     // have the cursor leave dynamics to keyin values
//lock value if keyed in; pause dynamics tracking updating in dialog when keying in
//(not part of accudraw) esc cancels out of/unlocks last entered value
//how to handle equivalent of keying in x or y (lock x or y) in accudraw given that l is mapped to Lock Index; w
     //is currently not used
//if length and width are locked, still need to query user for quadrant
//negative values permitted
//anything else??? check accudraw behavior
//need to consider making an abstract ToolPanel that defines most common useage methods, with the ability to
     //override methods to modify behavior as needed

class ToolPanel
     extends JPanel
     implements ItemListener,
     KeyListener {
        // -- Incoming data
        FunctionSizes m_sizes;

          // -- Size and angle
          NumericTextField m_XSizeField;
          NumericTextField m_YSizeField;
          NumericTextField m_ZSizeField;
          NumericTextField m_angleField;

          JLabel     m_XSizeLabel;
          JLabel     m_YSizeLabel;
          JLabel     m_ZSizeLabel;
          JLabel     m_angleLabel;
          JCheckBox  m_XSizeCheckBox;
          JCheckBox  m_YSizeCheckBox;
          JCheckBox  m_angleCheckBox;

        // -- Solution cell size list
        JLabel     m_cellLabel;
        JComboBox  m_cellField;

          // -- Buttons
          JButton       m_rotateL;
          JButton       m_rotateR;
          JToggleButton m_mirrorX;
          JToggleButton m_mirrorY;

          // -- SpaceStorySpan
          JPanel     m_spanOpts;
          JLabel     m_spanType_label;
          JComboBox  m_spanType;
          JLabel     m_spanAmount_label;

          // change to NumericTextField if decision is made to turn this feature back on
          JTextField m_spanAmount;

          NumberFormat m_formatter;

          public ToolPanel( FunctionSizes sizes ) {
               m_formatter = NumberFormat.getNumberInstance();
               m_formatter.setGroupingUsed( false );        //no commas
               m_formatter.setParseIntegerOnly( false );    //allow decimals
               m_formatter.setMinimumFractionDigits( 2 );   //always 2 sig dig
               m_formatter.setMaximumFractionDigits( 2 );

               try {
                    initPanel();
                reset( sizes );
               }
               catch (Exception ex) {
                    ex.printStackTrace();
               }
          }
```

```
public void initPanel() {
     GridBagLayout      lay = new GridBagLayout();
     GridBagConstraints con = new GridBagConstraints();

     // Create components
     m_XSizeLabel = new JLabel( "Length:" );
     m_XSizeField = new NumericTextField( null, null, 0, 9999, 0, 9999, true );
     finishTextField( m_XSizeField, 8, "Length (X-size) of space." );
     m_XSizeCheckBox = new JCheckBox();

     m_YSizeLabel = new JLabel( "Width:" );
     m_YSizeField = new NumericTextField( null, null, 0, 9999, 0, 9999, true );
     finishTextField( m_YSizeField, 8, "Width (Y-size) of space." );
     m_YSizeCheckBox = new JCheckBox();

     m_ZSizeLabel = new JLabel( "Ceiling:" );
     m_ZSizeField = new NumericTextField( null, null, 0, 9999, 0, 9999, true );
     finishTextField( m_ZSizeField, 8, "Height (Z-size, floor to ceiling) of space." );

     m_angleLabel = new JLabel( "Angle:" );
     m_angleField = new NumericTextField( null, null, 0, 9999, 0, 9999, false );
     finishTextField( m_angleField, 8, "Angle of space's +X axis relative to drawing's." );
     m_angleCheckBox = new JCheckBox();

     // SpaceStorySpan, plus border to group them
     m_spanOpts = makeSpanOpts();
     m_spanOpts.setBorder( BorderFactory.createEtchedBorder() );

  // Pre-Engineered Solutions
  m_cellLabel = new JLabel( "Solution:" );
  m_cellField = new JComboBox();
  m_cellField.setMaximumRowCount( 9999 );
  m_cellField.setEnabled( false );

  m_cellField.addActionListener( new ActionListener() {
                                   public void actionPerformed(ActionEvent e) {
                                       resetEnabled();
                                    }
                                } );

     // Fun buttons
     m_mirrorX = new JToggleButton( "H" );
     m_mirrorY = new JToggleButton( "V" );

     m_rotateL = new JButton( "L" );
     m_rotateL.addActionListener( new ActionListener() {
                                          public void actionPerformed(ActionEvent e) {
                                               setAngleLocked( true );
                                               setAngleValue( Math.IEEEremainder(getAngleValue() +
                                                   Math.PI / 4, 2*Math.PI) );
                                          }
                                     } );

     m_rotateR = new JButton( "R" );
     m_rotateR.addActionListener( new ActionListener() {
                                          public void actionPerformed(ActionEvent e) {
                                               setAngleLocked( true );
                                               setAngleValue( Math.IEEEremainder(getAngleValue() -
                                                   Math.PI / 4, 2*Math.PI) );
                                          }
                                     } );

     // --- Build dialog --- //
     setLayout( lay );

     // Arrange labels
     con.gridx = 0;
     con.gridy = 0;
     lay.setConstraints( m_XSizeLabel, con );
     add( m_XSizeLabel );
     con.gridy = GridBagConstraints.RELATIVE;
     lay.setConstraints( m_YSizeLabel, con );
     add( m_YSizeLabel );
     lay.setConstraints( m_angleLabel, con );
     add( m_angleLabel );
     lay.setConstraints( m_cellLabel, con );
     add( m_cellLabel );

     // Arrange checkboxes
     con.gridx = 2;
     con.gridy = 0;
     lay.setConstraints( m_XSizeCheckBox, con );
     add( m_XSizeCheckBox );
     con.gridy = GridBagConstraints.RELATIVE;
     lay.setConstraints( m_YSizeCheckBox, con );
     add( m_YSizeCheckBox );
     lay.setConstraints( m_angleCheckBox, con );
     add( m_angleCheckBox );

     // Arrange text fields to take all remaining space
     con.gridx = 1;
     con.gridy = 0;
     con.fill = GridBagConstraints.HORIZONTAL;
     lay.setConstraints( m_XSizeField, con );
     add( m_XSizeField );
     con.gridy = GridBagConstraints.RELATIVE;
```

```
        lay.setConstraints( m_YSizeField, con );
        add( m_YSizeField );
        lay.setConstraints( m_angleField, con );
        add( m_angleField );
        lay.setConstraints( m_cellField, con );
        add( m_cellField );

        {
             //Add buttons and switches as 1 row
             LayoutManager _lay = new FlowLayout();
             JPanel        _panel = new JPanel(lay);

             _panel.add( m_rotateL );
             _panel.add( m_rotateR );

             int z = con.fill;
             con.fill = GridBagConstraints.VERTICAL;
             lay.setConstraints( _panel, con );
             add( _panel );
             con.fill = z;
        }

        setMinimumSize( getPreferredSize() );
        setSize( getPreferredSize() );
    }


public void reset( FunctionSizes sizes ) {
    if( sizes == m_sizes )
        return;

    // Try to preserve current selection
    String selected = getGraphic();

    m_sizes = sizes;

    m_cellField.setEnabled( m_sizes != null && m_sizes.descriptions != null );

    m_cellField.removeAll();
    m_cellField.removeAllItems();
    m_cellField.addItem("Dynamic");

    if( m_sizes != null && m_sizes.descriptions != null ) {
        for( int i=0; i<m_sizes.descriptions.length; i++ )
            m_cellField.addItem( m_sizes.descriptions[i] );
    }

    if( selected != null && m_cellField.getItemCount() > 1 ) {
        int idx = -1;
        for( int i=0; i<m_sizes.cells.length; i++ )
            if( m_sizes.cells[i].equalsIgnoreCase(selected) ) {
                idx = i;
                break;
            }
        if( idx >= 0 )
            m_cellField.setSelectedIndex( idx+1 );
    }

    resetEnabled();
}

  public void resetEnabled() {
    boolean dynamic = ( m_cellField.getSelectedIndex() == 0 );
    setLenEnabled( dynamic );
    setWidthEnabled( dynamic );
    setAngleEnabled( true );
}

  private JTextField makeTextField( int width, String tooltip ) {
       JTextField txt = new JTextField();
       finishTextField( txt, width, tooltip );
       return txt;
  }

  private void finishTextField( JTextField field, int width, String tooltip ) {
       field.setColumns( width );
       field.setToolTipText( tooltip );
       field.addKeyListener( this );
  }

  private JPanel makeSpanOpts() {
       GridBagLayout      lay = new GridBagLayout();
       GridBagConstraints con = new GridBagConstraints();
       JPanel panel = new JPanel(lay);

       // Create components
       String[] spanTypeItems = SpaceStorySpan.getItemText();
       m_spanType_label = new JLabel( "Span:" );

       m_spanType = new JComboBox( spanTypeItems );
       m_spanType.setEditable( false );
       m_spanType.addItemListener( this );

       m_spanAmount_label = new JLabel( "Distance:" );

       m_spanAmount = makeTextField( 8, "Distance from top of slab to next top of slab." );
```

```
     // Arrange labels
     con.gridx = 0;
     con.gridy = 0;
     lay.setConstraints( m_spanType_label, con );
     panel.add( m_spanType_label );
     con.gridy = GridBagConstraints.RELATIVE;
     lay.setConstraints( m_spanAmount_label, con );
     panel.add( m_spanAmount_label );

     // Arrange checkboxes
     //con.gridx = 2;
     //con.gridy = 0;
     //-none here!

     // Arrange text fields to take all remaining space
     con.gridx = 1;
     con.gridy = 0;
     con.fill = GridBagConstraints.HORIZONTAL;
     lay.setConstraints( m_spanType, con );
     panel.add( m_spanType );
     con.gridy = GridBagConstraints.RELATIVE;
     lay.setConstraints( m_spanAmount, con );
     panel.add( m_spanAmount );

     panel.setMinimumSize( getPreferredSize() );

     return panel;
}



/*------------------------------
** Listener methods
*------------------------------*/

/**
 * User selected from pop-up list.
 */
public void itemStateChanged( ItemEvent e ) {
     if (e.getStateChange() == e.SELECTED && e.getItemSelectable() == m_spanType) {
          m_spanAmount.setEnabled( true );

          switch (m_spanType.getSelectedIndex()) {
               case 0:
                    //"Height From Slab"
                    m_spanAmount_label.setText( "Distance:" );
                    break;
               case 1:
                    //"# Stories"
                    m_spanAmount_label.setText( "Stories:" );
                    break;
               case 2:
                    //"To Story"
                    m_spanAmount_label.setText( "Story:" );
                    break;
               case 3:
                    //"Top of Bldg"
                    m_spanAmount_label.setText( "-" );
                    m_spanAmount.setEnabled( false );
                    break;
               case 4:
                    //"Height from Top"
                    m_spanAmount_label.setText( "Distance:" );
                    break;
               default:
                    m_spanAmount_label.setText( "Distance:" );
          }
     }
}


/**
 * Hack to intercept GUI events only.  This code triggers only when the user fiddles with
 * the text box; it's NOT triggered when code does a setText or setValue.  This code is
 * called BEFORE the field is updated, so getText/getValue won't be up to date.
 * <P>
 * I'd prefer some sort of ValueChangedEvent, but those are all called on setText/setValue.
 */
public void keyPressed(KeyEvent e) {
     //This code MUST be under keyPressed, keyTyped doesn't get events for DEL, UP, DOWN,
     //LEFT, RIGHT, or any other action key.

     //Lock field if user makes change to content
     // -ignore cursor keys
     // -special case for BS, since it's not actionKey
     //Function needs if/else 'cuz text field is not updated til we return,
     //  so .length() == 0 even if printable key is pressed.

     //If printable, lock field.
     //  BS is not considered an action key, but we don't want to handle it here.
     //  We DO want to handle Up and Down, these increment and decrement the value.
     if( (! e.isActionKey() && e.getKeyChar() != e.VK_BACK_SPACE)
          || (e.getKeyChar() == e.VK_UP || e.getKeyChar() == e.VK_DOWN) ) {
          if (e.getSource() == m_XSizeField) {
               setLenLocked( true );
```

```
          }
          else if (e.getSource() == m_YSizeField) {
               setWidthLocked( true );
          }
          else if (e.getSource() == m_angleField) {
               setAngleLocked( true );
          }
     }

     //If not printable, unlock if empty, lock if unempty.
     else {
          if (e.getSource() == m_XSizeField) {
               if (m_XSizeField.getText().length() == 0)
                    setLenLocked( false );
               else
                    setLenLocked( true );
          }
          else if (e.getSource() == m_YSizeField) {
               if (m_YSizeField.getText().length() == 0)
                    setWidthLocked( false );
               else
                    setWidthLocked( true );
          }
          else if (e.getSource() == m_angleField) {
               if (m_angleField.getText().length() == 0)
                    setAngleLocked( false );
               else
                    setAngleLocked( true );
          }
     }
}

public void keyTyped(KeyEvent e) {
     //ignored
     //Can't use this, it doesn't get events for DEL, UP, DOWN, LEFT, RIGHT.
}

public void keyReleased(KeyEvent e) {
     //ignored
}


/*------------------------------
** Accessor/mutator methods
*------------------------------*/
public boolean isLenLocked() {
     return(m_XSizeCheckBox.isSelected());
}

public boolean isWidthLocked() {
     return(m_YSizeCheckBox.isSelected());
}

public boolean isAngleLocked() {
     return(m_angleCheckBox.isSelected());
}

public void setLenLocked(boolean val) {
     m_XSizeCheckBox.setSelected(val);
}

public void setWidthLocked(boolean val) {
     m_YSizeCheckBox.setSelected(val);
}

public void setAngleLocked(boolean val) {
     m_angleCheckBox.setSelected(val);
}

public void setLenEnabled(boolean val) {
     m_XSizeField.setEnabled(val);
     m_XSizeCheckBox.setEnabled(val);
}

public void setWidthEnabled(boolean val) {
     m_YSizeField.setEnabled(val);
     m_YSizeCheckBox.setEnabled(val);
}

public void setAngleEnabled(boolean val) {
     m_angleField.setEnabled(val);
     m_angleCheckBox.setEnabled(val);
     m_rotateL.setEnabled(val);
     m_rotateR.setEnabled(val);
}


public double getLengthValue() {
     return m_XSizeField.getValue();
}

public double getWidthValue() {
     return m_YSizeField.getValue();
}

public double getCeilingHtValue() {
     return m_ZSizeField.getValue();
```

```
        }

        /** Radians! */
        public double getAngleValue() {
             return m_angleField.getValue() / 180 * Math.PI;
        }

        public SpaceStorySpan getSpanValue() {
             double spanAmount;

             try {
                  spanAmount = m_formatter.parse(m_spanAmount.getText()).doubleValue();
             }
             catch (ParseException ex) {
                  spanAmount = 0;
             }
             return new SpaceStorySpan( m_spanType.getSelectedIndex(), spanAmount );
        }

        public boolean getMirrorX() {
             return m_mirrorX.isEnabled();
        }

        public boolean getMirrorY() {
             return m_mirrorY.isEnabled();
        }


        /** Return cell name selected by user, or null if dynamic mode. */
        public String getGraphic() {
             // skip 'dynamic' option
             int idx = m_cellField.getSelectedIndex()-1;
             if( idx >= 0 )
                  return m_sizes.cells[idx];
             else
                  return null;
        }

        /** Return description for cell name selected by user, or null none. */
        public String getGraphicDescription() {
            // skip 'dynamic' option
            int idx = m_cellField.getSelectedIndex()-1;
            if( idx >= 0 )
                return m_sizes.descriptions[idx];
            else
                return null;
        }


        public void setLengthValue(double d) {
             m_XSizeField.setText(d);
        }

        public void setWidthValue(double d) {
             m_YSizeField.setText(d);
        }

        public void setCeilingHtValue(String val) {
             m_ZSizeField.setText(val);
        }

        /** Radians! */
        public void setAngleValue(double d) {
             m_angleField.setText(d * 180 / Math.PI);
        }

        public void setSpanValue(SpaceStorySpan val) {
             m_spanType.setSelectedIndex( val.getType() );
             m_spanAmount.setText( Double.toString(val.getVal()) );
        }

        public void setMirrorX( boolean val ) {
             m_mirrorX.setEnabled( val );
        }

        public void setMirrorY( boolean val ) {
             m_mirrorY.setEnabled( val );
        }

    }
}
```

10 Bibliography

Alexander, C. (1977). *A Pattern Language*. Harvard University Press, Cambridge, MA.

Akin, Ö. and Moustapha, H. (2003). "Strategic Use of Representation in Architectural Massing" Design Studies, Vol. 25, no 1, Elsevier Ltd, London.

Carr, D. (1995). "A Compact Graphical Representation of User Interface Interaction Objects," Doctoral Dissertation, University of Maryland, College Park, MD.

Cha M., and Gero, J. (2001). Shape Pattern Representation for Design Computation. Unpublished manuscript available from http://www.arch.usyd.edu.au/~john/publications/ChaGero.pdf.

Chase S. C. (1996). "Modeling Designs With Shape Algebras and Formal Logic," Ph.D. Dissertation, University of California, Los Angeles.

Chase, S. (1997). "Emergence, Creativity and Computational Tractability in Shape Grammars" in Preprints of workshop, Interactive Systems for Supporting the Emergence of Concepts and Ideas, CHI '97, Atlanta, GA, Ed E Edmonds, T Moran.

Ching, F. (1996). *Architecture: Form, Space and Order,* Van Nostrand Reinhold: International Thomson Publishing, Inc.: New York.

Choudhary, R., and Michalek, J. (2005). "Design Optimization in Computer Aided Architectural Design," International Conference of the Association for Computer Aided Architectural Design Research In Asia, New Delhi, India.

Damski, J. C. and Gero, J. S. (1997). "An Evolutionary Approach to Generating Constraint-Based Space Layout Topologies," in R. Junge (ed.), CAAD Futures 1997, Kluwer, Dordrecht, pp. 855-874.

Facility Composer (2005). http://fc.cecer.army.mil/ (Last viewed October 1, 2005).

Faltings, B. (1997). "Case Reuse by Model-based Interpretation." in Issues and Applications of Case Based Reasoning in Design, M.-L. Maher, P. Pu, eds. Lawrence-Earlbaum Publishers.

Flemming, U. (1978). "Wall Representations of Rectangular Dissections and their use in Automated Space Allocation," Environment and Planning B: Planning and Design, 5, 215-232.

Flemming, U. (1986). "On the Representation and Generation of Loosely Packed Arrangements of Rectangles", Environment and Planning B, 13, 189-205.

Flemming, U. (1987). "More Than the Sum of Parts: the Grammar of Queen Anne Houses," Environment and Planning B, Vol. 14, 1987, pp. 323-350.

Flemming, U. and Woodbury, R (1995). "Software Environment to Support Early Phases in Building Design (SEED): Overview", in Journal of Architectural Engineering, ASCE, 1(4): 147-152.

Gero, J. S. and Kazakov, V. (1997). "Learning and Reusing Information in Space Layout Problems using Genetic Engineering," Artificial Intelligence in Engineering 11(3): 329-334.

Harada, M., Witkin, A., and Baraff., D. (1995). “Interactive physically-based manipulation of discrete/continuous models,” SIGGRAPH ’95 Conference Proceedings, ACM Siggraph, ACM, 29, 199—208.

Herzog, M. (1994). “The Use of Intelligent Hypermedia in Architectural Design Environments - A Conceptual Framework”
Doctoral Dissertation, Vienna University of Technology, Vienna, Austria.

Jagielski, R. and Gero, J. S. (1997). “A Genetic Programming Approach to the Space Layout Planning Problem,” in R. Junge (ed.), CAAD Futures 1997, Kluwer, Dordrecht, pp. 875-884.

Kim, M. (1980). ‘Countermodeling as a Strategy For Decision Making: Epistemological Problems in Design,” Ph.D. Dissertation, University of California, Los Angeles.

Knight, T. (2003). “Computing With Ambiguity," Environment and Planning B: Planning and Design 30 165-180.

Leyton, M. (2001). *A Generative Theory of Shape*, Springer Verlag, New York.

Liew, H. (2004). “Extending Shape Grammars with Descriptors” in J Gero (ed) Design Computing and Cognition ’04, Kluwer, Dordrecht, pp. 417-436.

Liew, H. (2002). “Descriptive Conventions for Shape Grammars,” Thresholds - Design, Research, Education and Practice, in the Space Between the Physical and the Virtual [Proceedings of the 2002 Annual Conference of the Association for Computer Aided Design In Architecture / ISBN 1-880250-11-X] Pomona (California) 24-27 October 2002, pp. 365-378

Medjdoub, B. and Yannou, B. (2000). “Separating Topology and Geometry in Space Planning". Computer Aided Design; 32(1), p. 39-61.

Michalek, J. J., Choudhary, R., and Papalambros, P. Y. (2002). "Architectural Layout Design Optimization", Engineering Optimization, Vol. 34, No. 5, pp. 461-484.

Michalek, J. J. and Papalambros, P. Y. (2002). "Interactive Design Optimization of Architectural Layouts", Engineering Optimization, Vol. 34, No. 5, pp. 485-501.

Modular Design System (2005). http://www.cecer.army.mil/td/tips/product/details.cfm?ID=577 (Last viewed October 1, 2005).

Moustapha, H. (2004). "A Formal Representation for Generation and Transformation in Design,” Generative CAD Systems Symposium (GCAD’04), Carnegie Mellon University, Pittsburgh.

Oxman, R.M. and Oxman, R.E. (1990). "The Computability of Architectural Knowledge," in M. McCullough, W.J. Mitchell, P. Purcell (Eds.) The Electronic Design Studio, MIT Press, pp. 171-186.

Rittel H and Webber M (1973). “Dilemmas in a General Theory of Planning”, Policy Sciences, Vol. 4, pp. 155-169, Amsterdam, Elsevier.

Simon, H. (1981). *The Sciences of the Artificial*, MIT Press, Cambridge, MA.

Smithers, T. and Troxell, W. (1990). “Design is intelligent behaviour, but what's the formalism?” Journal of Artificial Intelligence for Engineering and Design, Analysis, and Manufacturing, Vol. 4, No. 2, 89-98.

Stiny, G. (1980). “Introduction to Shape and Shape Grammars,” Environment and Planning B, 7, pp. 343-351.

Stiny G. (1994). “Shape Rules: Closure, Continuity, and Emergence,” Environment and Planning B: Planning and Design, 21, 1994, s1-s116.

Stouffs, R. and Krishnamurti, R. (1994). "An Algebraic Approach to Shape Computation," Workshop on Reasoning with Shapes in Design, Artificial Intelligence in Design'94, Lausanne, Switzerland, pp. 50-55.

TCMS (2005). "Theatre Construction Management System" http://www.tcms.net/ (Last viewed October 1, 2005).

Woodbury, R. and Burrow, A. (2005). " Whither Design Space?" Prepublished manuscript to appear in Journal of Artificial Intelligence for Engineering and Design, Analysis, and Manufacturing, Special issue on Design Spaces: the Explicit Representation of Spaces of Alternatives, Vol. 20, No. 2, 2006.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
08-2005	Final	

4. TITLE AND SUBTITLE
N-Dimensional Abstraction Patterns in NP-Hard Design Space

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Van J. Woods

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Research and Development Center (ERDC)
Construction Engineering Research Laboratory (CERL)
PO Box 9005
Champaign, IL 61826-9005

8. PERFORMING ORGANIZATION REPORT NUMBER
ERDC/CERL TR-05

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Copies are available from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

14. ABSTRACT

Current state of the art research has demonstrated the applicability of parametric shape grammar, optimization, and constraint-based interactive techniques for the space layout problem. However, they fall short in the following primary areas: performance, flexibility, and control. The basic premise of this work is based on the principle that a hierarchical pattern-based design method will provide a more robust approach.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Van J. Woods
Unclassified	Unclassified	Unclassified	SAR	0	19b. TELEPHONE NUMBER (include area code) (217) 3526511, X-7675