Michael J. Dalyrmple
University of Colorado
michael.dalyrmple@usa.net

J. Michael Gerzso
Independent Researcher
104164.341@compuserve.com

Executable Drawings: The Computation of Digital Architecture

Architectural designs are principally represented by drawings. Usually, each drawing corresponds to one design or aspects of one design. On the other hand, one executable drawing corresponds to a set of designs. These drawings are the same as conventional drawings except that they have computer code or programs embedded in them. A specific design is the result of the computer executing the code in a drawing for a particular set of parameter values. If the parameters are changed, a new design or design variation is produced.

With executable drawings, a CAD system is also a program editor. A designer not only designs by drawing but also programming. It fuses two activities: the first, drawing, is basic in architectural practice; and the second, progamming, or specifying the relation of outputs from inputs, is basic in computer system development. A consequence of executable drawings is that architectural form is represented by graphical entities (lines or shapes) as well as computer code or programs. This type of architecture we call digital architecture.

Two simple examples are presented: first, the design of a building in terms of an executable drawing of the architects, Sangallo the Younger and Michelangelo, and second, a description of an object oriented implementation of a preliminary prototype of an executable drawing system written in 1997 which computes a simple office layout.

Les dessins exécutables: Le calcul d'une architecture digitale

Le design en Architecture est généralement représenté par des dessins. La plupart du temps, chaque dessin correspond à un design ou à certains aspects d'un design. D'autre part, un dessin exécutable correspond à un ensemble de designs. Ces dessins exécutables sont semblables à des dessins conventionnels, sauf qu'ils comprennent du code pour l'ordinateur, ou encore des programmes. Un design spécifique est le résultat de l'exécution du code à l'intérieur d'un dessin pour un ensemble particulier de valeurs des paramètres. Si on change les paramètres, un nouveau design, ou variation de design est produit.

Avec des dessins exécutables, un système DAO est aussi un éditeur de programmes. Un concepteur ne conçoit pas seulement en dessinant, mais aussi en programmant. Il se produit une fusion des deux activités: la première, le dessin, est primordiale dans la pratique de l'architecture, et la deuxième, la programmation, ou la spécification des résultats des données fournies à l'ordinateur, est fondamental lors du développement de systèmes informatiques. Un résultat des dessins exécutables est que la forme architecturale est représentée par des entités graphiques (lignes ou formes), ainsi que par du code pour l'ordinateur ou des programmes. Ce type d'architecture nous appelons 'architecture digitale'.

Deux exemples simples sont présentés: premièrement, la conception d'un édifice en fonction d'un dessin exécutable produit par les architectes Sangallo le Jeune et Michelange; deuxièmement, une description d'une implantation orientée-objets d'un prototype préliminaire d'un système de dessin exécutable, écrit en 1997, qui calcule un plan simple de bureau.

introduction

For most architectural designers who use CAD systems, the computer has automated many of their drawing activities. Even though they are aware that computers must be programmed to do anything, they rarely engage in programming as they design their buildings. To them, programs reside somewhere in the bowels of the machine or a commercial CAD package.

And yet, a fundamental aspect of the digital computer is that it can be programmed, and for this reason, it is different from any other machine or technology. Nevertheless, most commercial CAD companies define a drawing in their systems as digital counterparts of static paper drawings. They pass up the unique opportunity, which only the computer makes possible, of merging drawing data with code to make drawings executable.

The main purpose of this paper is to present the ideas of executable drawings (ED's) and digital architecture by means of two simple examples: the Palazzo Farnese and an office layout. The objective is to show how code can be embedded in CAD drawings. Some of the ideas have been implemented, such as in the prototype written in object oriented MacIntosh Common LISP (MCL), and others are to be considered specifications of an ED system. At the same, it is recognized that real architectural drawings are much more complicated than those presented here, and that for ED´s to be successful, they must be able to represent them too. The examples do not intend to imply that all buildings and floor plans are based on simple and regular grids or cells. In addition, the code that appears in the examples is pseudo code or fake code in the sense that they have no syntax or semantics as do real computer languages. This is to make it accesible to those readers who are not programmers.

ED's are interesting by the mere fact that they are programs, while CAD drawings are data to be processed by a CAD program. They are also interesting because they have advantages over paramteric shapes and shape grammars. This will also be considered in this paper.

definition of executable drawings

In architecture, there is usually a one to one correspondence between a drawing and a design. In contrast, with executable drawings (ED), there is a one to many correspondece between an ED and a set of designs or variants. This characteristic of ED's makes possible the representation of a design as a set of variants by means of a computer program. Consequently, the architecture represented by ED's is called *digital architecture*. It is an architecture produced by a computer executing an ED program.

To put it another way, an executable drawing is like any other drawing in a CAD system except that each individual area in the drawing can have code embedded in it. This is what makes the drawing "executable." The areas must be well defined by a polygon or other closed figures. The areas may correspond to neighborhoods, streets, city lots, buildings, and rooms within a building.

In addition, an important characteristic of the ED is the way in which the designer uses it or interacts with it. Anytime the drawing is modified, it is recalculated in a similar way as a spreadsheet recalculates by modifying one its cells. The results can be a new drawing or the quantification of entities in the drawing related to the coded areas, such as cost and quantification of construction materials, capital expenditures, parking places, tax revenue, etc.

In summary an executable drawings fuses two activities: the first, drawing, is basic in architectural practice; and the second, programming, or specifying the relation of outputs from inputs, is basic in computer system development.

motivation

One of many needs that could be satisfied by ED's is the calculation of cost and materials specified in a design represented in a drawing. The applications in which this type of calculation would be useful are in architecture, engineering, urban planning, facilities management, and geographical information system, among others. In all of these cases, a drawing not only represents a layout but also physical elements that occupy the spaces with their corresponding sets of attributes.

There are many situations in which ED's could be very useful. A typical one is an architect or engineer who is developing a design in collaboration with a client. The client may frequently change the specification of his project, and requires that the designer inform him of the consequences of those changes in terms of cost, investment, or whatever. Among the many reasons that a client would do this is that he is adjusting to the constant flux of an economic and business environment.

origins of executable drawings

The idea of executable drawings came about in 1991 as a consequence of consulting work in geographical information systems (GIS) in a company called Programacion No Numerica S.A. de C.V. (Pronon). It was an attempt to solve the problem of calculating the consequences of changing property tax laws, property valuation and land use regulations for the Department of Urban Planning of Mexico City, Mexico. It was an urgent need to address these vexing financial problems of the city given the chronic economic crisis that Mexico has suffered since 1980. In addition, the city must negotiate in a matter of days or weeks with neighborhood groups aspects of land use laws. This requires the recalculation of options under consideration in hours or days. Thus, what seemed to be required was some sort of map or drawing in a GIS/CAD system which has code embedded in specific parts of the drawings with correspond to lots, streets, and public areas. For example, each time a parameter in property taxes were needed to be changed, the effect could have been evident in terms of tax revenue associated with particular areas of the city, population density of land use, increased demand for parking areas, among others.

The insight that led to ED's was a result of being exposed to work in two areas. The first was the experience in quad and oct trees applications found in GIS (Sammet 1994) and the other one was the experience attained from designing and implementing an object oriented language called TM (Buchmann 1985).

In 1991, a very preliminary prototype of an ED was implemented at Pronon in AutoCAD V11 under DOS. A second system was design and pre-

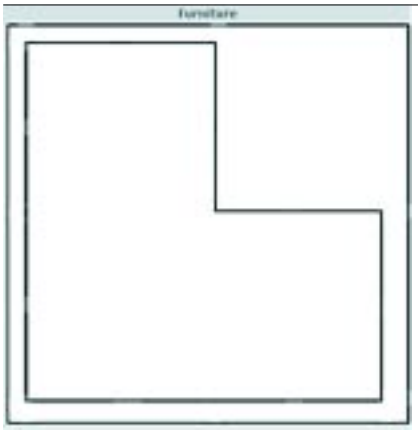

Figure 1. Floor plan boundary.

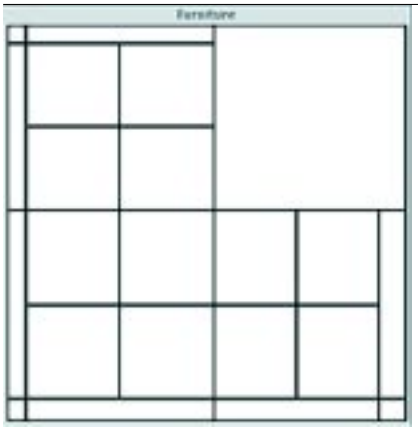

Figure 2. Creation of the grid layout.

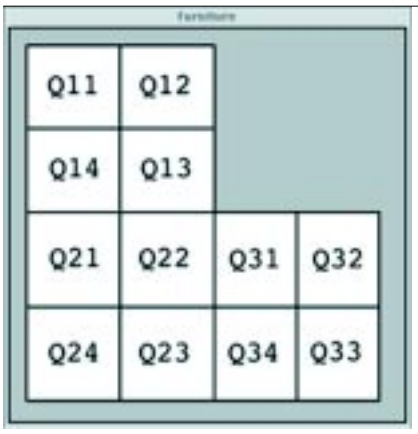

Figure 3. Cells with identifiers.

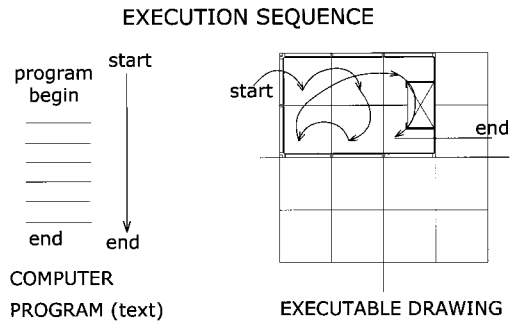

Figure 4. Sequence of execution of ED's versus typical programs.

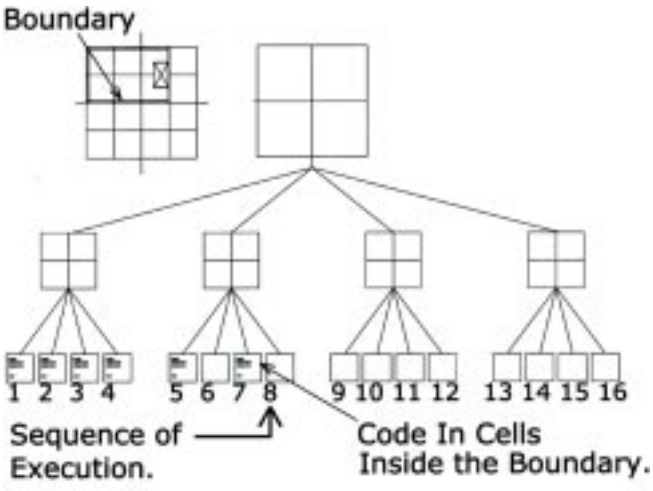

Figure 5. The layout and execution sequence represented as a quad tree.

sented at a conference on computer graphics in Mexico City (Gerzso 1991). A third prototype, which is object oriented, was implemented in 1997 by Michael Dalrymple in MacIntosh Common LISP (MCL) as a student project and is described in the rest of this article.

ED's: what the user sees

From the point of view of the user, designing with an ED has several things in common with the creation and manipulation of graphic elements in standard CAD programs. This is what is called the drawing view. But in addition, the ED has computer code which is assigned to specific areas in the drawing. While editing the code assigned to specific areas, the user uses what is called the code view. The text of the code view can be displayed along with the drawing elements or in a separate window. The user should be able to easily switch between the two views in a CAD system. In the preliminary demonstration version of an ED system described later in this paper, the code is viewed in an individual window for each area in an ED.

The analogy between ED's and spreadsheets serves only to illustrate how an ED recalculates after each modification of ED code. A spreadsheet does not change the graphical appearance of the cells after each execution but only the contents of the cells. However, the graphical characteristics of a particular ED drawing displayed by the CAD system can be the result of its execution. Depending upon the parameters and the code in the ED, the drawing can differ radically between executions. For example, a very simple ED has all of dimensions of the width and length of the cells multiplied by one global variable. If this variable is set to zero, the ED will not appear at all. If it is greater than zero, the ED will appear.

ED's: how the user interacts

The way in which the user interacts with an ED is a combination of the way he interacts with a CAD drawing and a spreadsheet. In the first case, all of the drawing creation and modification should be identical to the operations of AutoCAD, for example. In addition to the CAD functions, the ED system should have a spreadsheet like menu, such as Excel, which permits the user to create and edit code as he embeds it in a drawing.

As a result of implementing the preliminary prototype, the operations for the creation and use of an ED became apparent. First, the user starts with a blank area. It has a default width and length. Next, the boundary of the layout is drawn interactively inside the starting area (Figure 1). If the layout is a building, then the boundary corresponds to the outside walls. If it is a room, then the walls are the boundary. In theory, the boundary of the layout can be of any shape. Once the boundary is defined, the layout can be subdivided successively until a minimum module or cell is attained (Figure 2) . The cell size is determined by what the user wants to calculate. It can be an area corresponding to furniture, room, or buildings elements. Upon completion of the process of subdivision, the user begins to insert code in each cell of the ED, as is done in a spreadsheet (Figure 3). After that, all operations could be either to edit the graphical elements, such as the boundary, or the embedded code.

ED's vs CAD Drawings

In almost all basic commercial CAD systems, a drawing is represented internally by a sequential list of graphical elements, that is, 2D or 3D coordinate data. The CAD package, a program, processes this data to display the drawing on the screen from different points of view, to scale it, to render it, and to plot it. In viewing these outputs, the user infers spatial relationships, even though those relationships do not exist explicitly in the sequential list of elements, such as bottom left, bottom right, top right, and so on. Some of the relationships, such as one space inside another, or overlapping another, or outside of another, are computed at great expense. They are determined by programs that calculate the relative positions of each coordinate of the object with respect to other objects. Because objects are not ordered spatially, these calculations are frequently done by brute force, that is, comparing the coordinates of every object with the coordinates of every other object. Therefore, the spatial relationships between spaces in a drawing is an illusion.

As an attempt to overcome these limitations of drawings in CAD and other graphical systems, specialists in GIS (Sammet 194) have developed

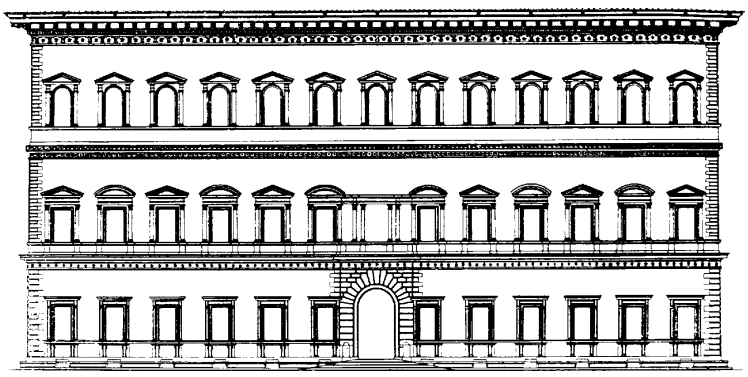

Figure 6. Elevation of the Palazzo Farnese.

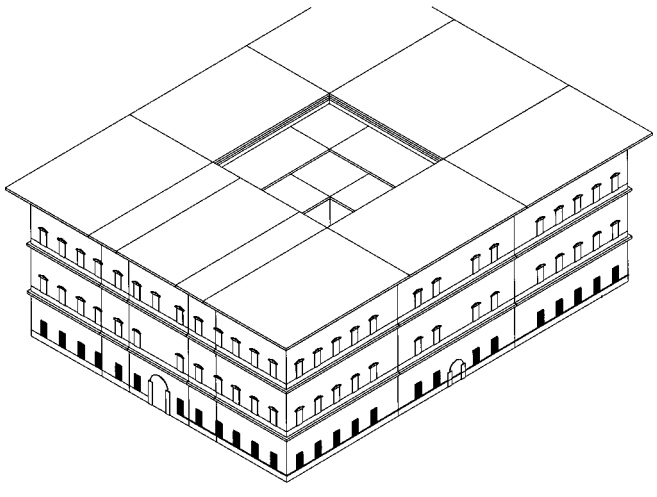

Figure 7. A simplified representation of the boundaries of the Palazzo Farnese in an ED.

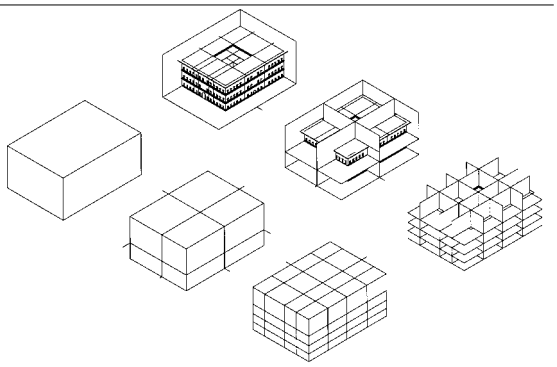

Figure 8. The process of subdividing the Palazzo Farnese into minimum cells.

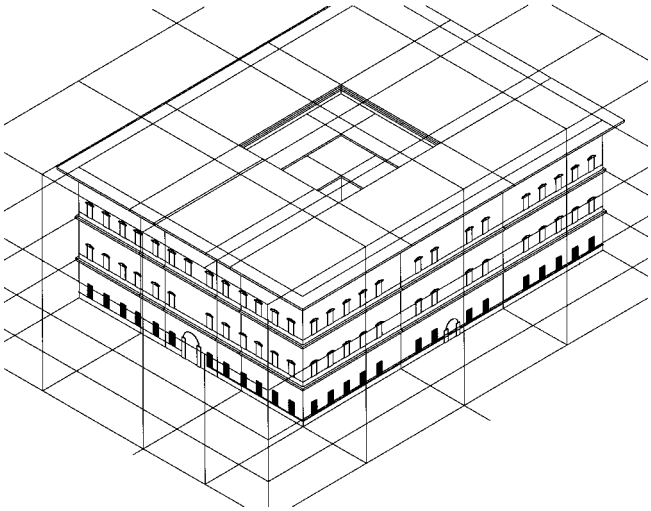

Figure 9. The Palazzo Farnese subdivided with minimum cells made transparent.

the internal representations of maps using quad and oct trees. The data, such as the boundaries of regions or spaces, is stored internally in such a way that each element has a one to one correspondence to a space in a map. The internal representation permits the easy computation of spatial relationships metioned above, as well as a determining unambiguously a sequence by which to visit each region in a map.

The idea of ED's takes as a starting point the work done in GIS quad and oct trees. Super imposing these internal representations on standard CAD drawings promises to facilitate the determination of spatial relationships. However, the fact that they provide an unambiguous sequence through a series of spaces in the drawing, also makes possible embedding code in the drawing. This makes it possible to place and execute code in two or three dimensional space.

ED's vs typical computer programs

Recipes are to programs as architectural drawings are to executable drawings. Programs written in languages such as C or AutoLisp are linear in that they are executed from top to bottom -despite "if" and "do" statements-. In this paper, they will referred to as text programs, because the code is written in ASCII text. Text programs can be understood by the analogy of the recipe used in many introductory programming courses. Recipes are instructions to process ingredients by human cooks, and programs are "recipes" to process data by electronic devices or computers. As we saw in the last section, a typical CAD drawing (ingredients) is processed by a CAD program (recipe).

The analogy that a recipe is like a program suggests another analogy: an architectual drawing is a kind of recipe for constructing buildings. Then an executable drawing is the computational counterpart of an architectural drawing, as a text program is the counterpart of a recipe.

Both typical text programs and ED's can be given to a computer for execution. However, the fact that drawings represent objects in two or three dimensional space requires defining an unambigous sequence through the drawing analogous to a linear sequence of text programs. This

suggest two problems: where to embed code in a drawing and according to what sequence to execute the instructions of the code. The solution is that the code is stored in closed nonoverlapping cells in two or three dimensional space, and the sequence of execution is controlled by the spatial ordering of the cell elements in a quad or oct tree (Figures 4-5).

advantages of ED's

As mentioned before, by changing the parameters in an ED, it is possible to systematically produce variants. Inputting the parameters can be done manually, or done by storing them in a table which is then used by the ED. The storing of parameters in tables was also implemented in the ED demo system written in object oriented MCL, but is not described here. ED's provide an alternative to parametric shapes and shape grammars. The first is a programming implementation methodology; the second is a descriptive approach to representing sets of architectural forms within a particular style. Both are ways in which to produce variations, and have become well known paradigms in influencing the direction of CAD.

Both paradigms have several advantages. First, parametric shapes are considered. In many applications, including architecture, it is necesary to insert variations of the same part in a drawings over and over again, such as furniture or construction detailing. Without the use of parametric shapes, the designer must draw a graphic entity once and later copy and edit it for each subsequent insertion. With parametric shapes, the drugery is reduced by representing a set of shapes in terms of code, such as AutoLisp or C *outside the drawing and part of the CAD system*. In the process of inserting a shape, the code that generates the shape is called. Before the shape is actually inserted, the code requests the input of parameters, which it uses to calculate the dimensions of the shape and create its corresponding graphical representation. In addition, the code may be linked to a menu in the CAD system.

The developers of the second paradigm, shape grammars and diagrammatic production rules (DPR's), are not software engineers working to augment the capabilities of existing commercial

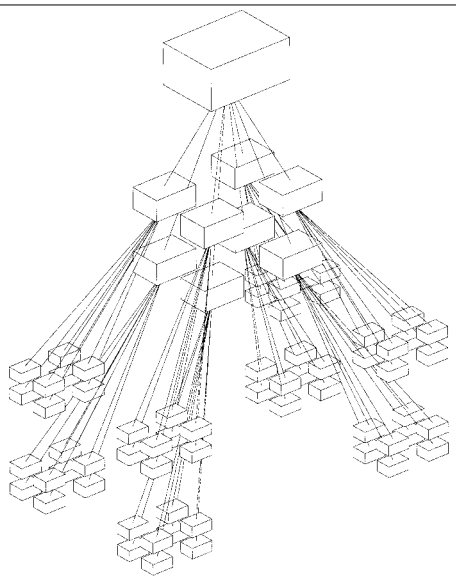

Figure 10. Exploded view of oct tree of cells of the Palazzo Farnese.

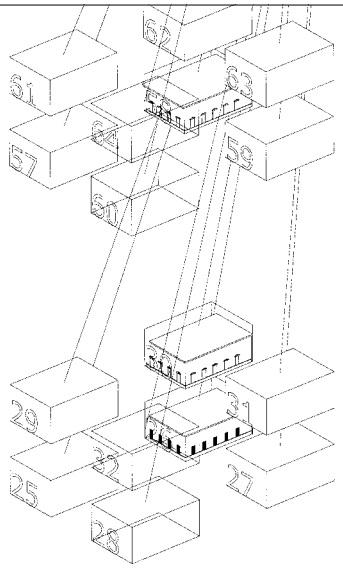

Figure 11. Detail of part of the oct tree showing execution sequence numbers.

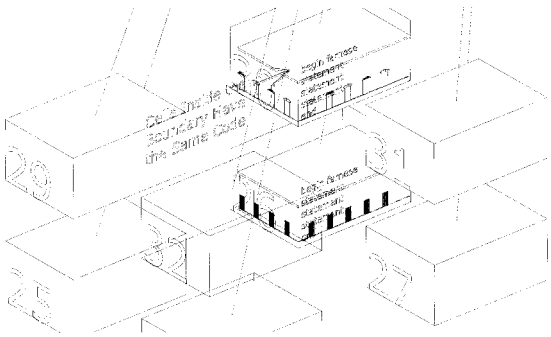

Figure 12. Detail showing cell code.

CAD systems. They are primarily concerned with conceiving approaches to representing sets of variations of shapes. Here, the shape is not coded in C or AutoLisp to be created with some parameters. Instead, each variation is member of a set of many shapes that, in principle, can be generated by so-called grammatical or production rules. The advantage of shape grammars is that many thousands or even millions of shapes can be generated using a small number of rules, say 50 or 200. They are analogous to the grammatical rules of a language, in which a small number of rules can produce (or recognize) an infinite number of sentences. In architecture, shape grammars and DPR's can be devised to generate instances of an architectural style, such as Utzon houses, San Francisco Victorian houses, Palladio villas, or Buffalo bungalows (Gerzso 1979; Mitchell 1990; Downing 1981; Stiny 1980).

Despite these advantages, both paradigms share several limitations that executable drawings do not. They are:

- the lack of an intinsic spatial representation.
- the problem of indirection.
- the necessity of a trained specialist to construct a model or a program.

the lack of spatial representation

The lack of a spatial representation in parametric shape systems is evident as a shape is inserted in the drawing. In the basic systems, which are the most common, the operator must be assure that shape does not overlap with other shapes. Only in sophisticated systems, such as ACIS, are there routines that determine the existence of inconsistencies. These systems may have an internal spatial representation similar to those proposed in this paper, and would justify a comparison between the two. But this is the topic of another paper.

Similar problems exist in shape grammars and DPR's. Unless they are perfectly understood, grammatical rules can produce nonsense variants or "sentences", such as for example, a bedroom and a bathroom occupying the same 3D space (Gerzso 1979).

The lack of a spatial representation manifests itself in other ways, and relevant to the topic at hand. It is that spatial grammars do not have an intrinsic mechanism for counting or for producing common spatial configurations, such as rectangular floor layouts. The difficulty resides in the characteristics of production rules themselves. This is a separate topic by itself and cannot be dealt with here (Gerzso 1979).

the problem of indirect representation

Indirect representation means that the set of variations of a shape is represented in some algorithmic form and the instances of the variations themselves are represented in a different form, that is, a drawing. In the case of parametric shapes, the first representation is a computer program written in C / C++, AutoLisp, or Java, and the second one is a drawing in a CAD system. In the case of shape grammars and DPR's, the first representation are the production rules, and the second is a drawing.

In implementing or defining the basic elements of parametric shapes, the implementors must deal with two different editors: a text editor for programming the programs, and a graphical editor, such as a CAD system. In developing the program, it is necesary wait to see the effect of running the program after connecting it to a CAD system. The programmer is required to think in a medium which is nonspatial, that is the code text, and then imagine its effect in a two or three dimensional space. It is not possible to edit and execute the text of parametric shapes interactively in CAD in the drawing itself.

In the case some commercial products, such as AutoCad Development System (ADS / ARX), the

problem is even more severe. Here, the C / C++ code must be edited, but the calls are made through the ADS / ARX set of functions which is analogous to two prisoners communicating through the prison plumbing in morse code. The C / C++ code must be compiled and linked and initialized in AutoCad via a special AutoLisp function. In response to this dilema, AutoDesk provided an intermediate solution, which is the codification of shapes in Visual Basic (Omura 1997).

With shape grammars, the indirection problem manifests itself also by first specifying the production rules, that is the grammar, which is different and separate from the drawing in which the resulting set will appear. As the rules are written, it is necesary to imagine the sets of solutions they can produce, because the actual computation of part or all of the set -if such a thing is possible- is delayed until most of the rules are defined. The indirection problem is further aggravated by the fact that inspecting the rules does not provide an easy way to predict the consequences of their application. It is even worse as one tries to add a new rule to the set. One must exercise the grammar, generate variants, and inspect them to see whether the grammar performs as expected. But this may take several iterations and the process of selecting the rules is done by trial and error until the rules are considered adequate.

a digression

Given the effort that has been dedicated to shape grammars in architecture, it may be beneficial to further pursue this aspect of indirect representation and gain some insights from other fields, such as software engineering.

Production rules are not unique to shape grammars or DPR's. In fact, most of the important work in this area is done outside of architecture. They are used mostly in linguistics, computer science, and software engineering. Even though the indirection problem also exists in those fields, it is dealt with in a different manner. In linguistics, the primary motivation is to model a natural language. The objective is to define a a set of rules and argue that they take into account all of the valid grammatical sentences in a language. The objective is not to generate sentences, much less prose or poetry. In computer science, a set of rules is also used

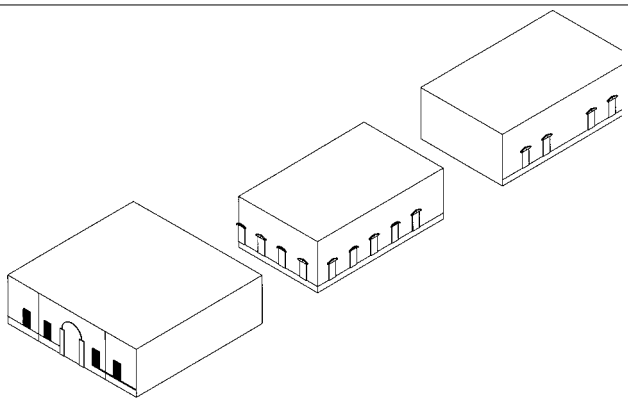

Figure 13. Some cells with windows.

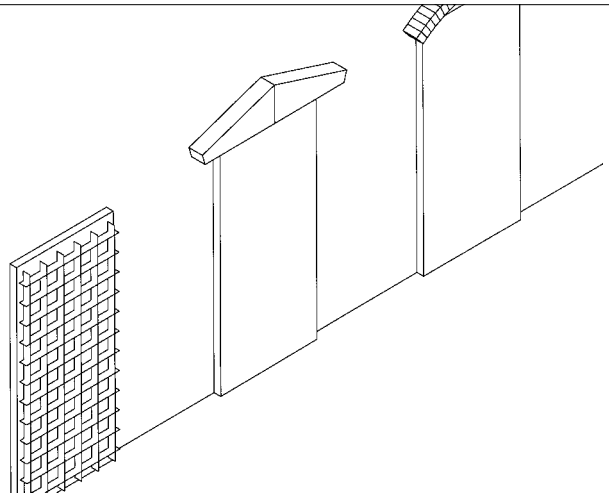

Figure 14. The three types of windows.

to take into account the valid sentences of a computer language, and in addition, to parse the sentence automatically by a compiler. Here too, generation of sentences is not the objective.

Practical experience in dealing with the indirect representation in computer languages gained by one of the authors (Buchmann 1985) in the development of an object oriented language called TM. It is not a unique experience and is well known by anybody attempting to construct a compiler (Aho 1979; Anklam 1997). First all, correctly specifying the rules of the grammar is crucial because compilers are very unforgiving. If the specification permits the recognition of an invalid sentence or is unable to recognize a valid sentence, then the rules must be modified. As in the case of shape grammars and DPR's, predicting the consequences of the modification of the grammar is not straight forward.

In the TM project, the language and compiler was redesigned four times. Each time, a major effort was concentrated on assuring consistancy of the rule system and the complete represention of the TM language -and not some other languages, such as Basic or Forth. This was done only once. After that, the writing of programs in TM became main activity. The effort in predicting the consequences of the application of the rules is justified because, once in place, the compiler will process many thousands of programs.

In the computer industry, the modification of the grammars of widely used languages, such as C / C++, Cobol or Fortran, is also a major issue. Even after a set of rules is accepted, there is a continuous stream of reports over a period of years about the unforseen consequences of the modifications. Unlike computer languages, shape grammars have additional problems. Architects are strongly motivated to explore the possibilities of generating design variants, and therefore may want to constantly modify the rules. The problem of predicting the consquences of rule modification becomes even more acute than in the case of compilers. But even if modifications are acceptable in a particular situation, the justification for doing so is weak because the rules are rarely incorporated in a system for everyday use.

In summary, the problem of indirection of grammars in software engineering exists as it does in shape grammars and DPR's. The effort in overcoming it is justified because the resulting compilers become the work horse programs to construct all other software. On the other hand, the effort in overcoming this problem in shape grammars and DPR's is not justified because there is no analogous work horse system.

the need for specialists

The third limitation of the parametric shapes and shape grammars is a direct consequence of the first two: the need for a specialist in writing code of the parametric shape or specifying production rules of a shape grammar or DPR. It is hard to image a client willing to sit down with a contractor / programmer to study variations requiring multiple cycles editing and compiling of code or production rules.

ED's overcome limitations

The idea of executable drawings aims to overcome the limitations of parametric shapes and shape grammars by adopting an intrinsic spatial representation, the quad and oct tree, as one of its fundamental characteristics. This is done by dividing up space into nonoverlapping rectangles or cells. Each cell, may have a graphic entity, computer code or nothing. Whatever graphic entity is in the cell, it remains entirely inside the cell. Whatever the computer code produces, it is confined inside the cell. The cells are stuctured so as to permit the determination of spatial relationships, such as adjacency, containment, etc. Therefore, as part of its structure, the executable drawing has a spatial ordering of cells. Anything that is computed in a cell does not invade other cells. And even though code in cell may be as arcane as the code in parametric shapes, it is confined to a specific chunk of space. In writing the code, the effect can be immediately seen in a particular place in the drawing. If the code is modified, again, the effect can be immediately seen.

In comparing executable drawing and parametric shapes, it is evident that net effect of executing a parametric shape and an executable drawing may be the same. In contrast to parametric shapes or programs, the main idea of executable

drawing is to eliminate the dual editing activities of text and graphics. Both reside in the same file, which is a drawing. Both can be edited at the same time. This means that text editing function must be added to the CAD systems. Therefore, the programming in an environment which supports executable programs promises to be more intuitive than parametric shape programming.

In addition, with executable drawings, there is the advantage that the activity of development is interactive and immediate, similar to a spread sheet. Upon finishing any modification of a drawing, the system executes it and the consequence of the modification can readily be appreciated. It is not necessary to consider them as two alternate and *incompatible* options, but *complementary*.

an example: the palazzo farnese

In this section, we will present the idea of ED's using the renaissance Palazzo Farnese built in Rome by Sangallo the younger and Michelangelo in the year 1513. Traditionally, the representation of this palace is done by drawings (Poroghesi 1972; Tzonis 1986), such as floor plans and elevations (Figure 6). However, we can represent the same building by an ED. That is, instead of only static drawings, we can use graphical elements and computer code which will produce the building features.

The Palazzo Farnese was selected for several reasons. The first one is that it has a regular floor plan and elevation which lends itself to illustrating an ED in three dimensions. Second, it can be easily related to the ideas presented in the previous sections on how an ED is created and used in two dimensions, and to the description of the ED prototype presented in subsequent sections. The third reason is that there are three types of aedicule oranamentation of the windows (Figure 14), and are placed according the their relative position on the facade. The square windows with bars are of the ground floor, the windows with rounded and triangular pediments are on the second floor, and the windows with only triangular pediments are on the third floor.

Based on the procedure described in the section "ED's: how the user interacts," it is possible to imagine how a user would interactively define the

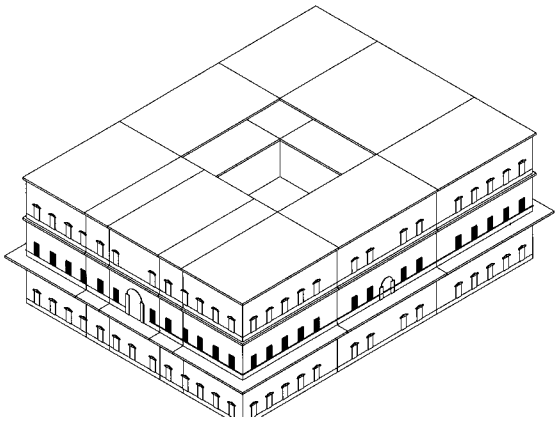

Figure 15. A variant of the Palazzo Farnese as a result of changes in the ED code.

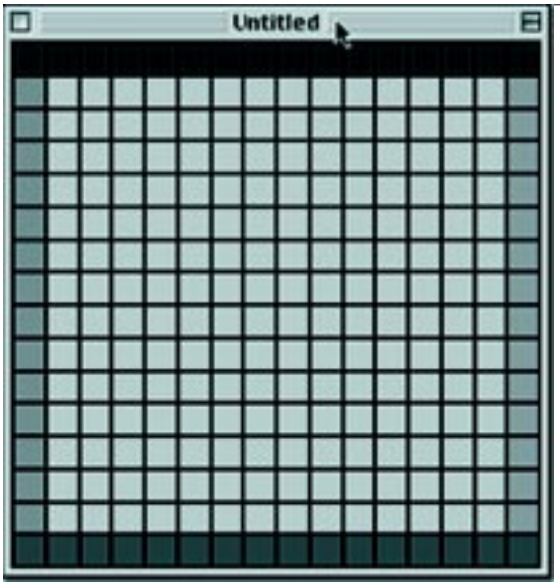

Figure 16. Area subdivided with borders colored.

Farnese ED. First, the boundaries are defined, but in three dimensions (Figure 7). Note that the figure includes the windows and doors, but this is the consequence of executing the ED. Strictly speaking, the only thing that should appear at this stage is a blank box defining the boundary. Next, the building is subdivided into minimum three dimensional cells (Figure 8-9). An exploded view of the oct tree of the palace can be seen in Figure 10. A detail of the tree (Figure 11) reveals the execution sequence numbers assigned to the cells. Three of the cells fall within the three dimensional boundary of the palace and correspond to three floors of one of the palace corners. The windows are desplayed as a consequence of the execution of the ED.

In each of the cells inside the boundary, a copy of the ED code is stored. When executed, the code determines where it is in the Farnese ED, and computes the corresponding doors and windows or both along the boundary. It does this depending upon *its spatial position in the building* (Figure 7 and 12).

An pseudo version of the code of each of the cells to compute windows may look like this:

```
BEGIN farnese facade in each cell.
    If cell on 1st floor, then
        create windows with bars.
    else if cell on second floor,
        create windows with round
            and triangular pediments
    else if cell on third floor then
        create windows with
            triangular pediments
        and create frieze.
END
```

A complete version of the cell code must include statements to make door and windows depending on whether it is on the first, second or third floor, and whether it is in the middle of the facade or on one of the corners. It also must make the facade elements along the boundary and not inside, so the cell code must be able to determine which face of the cells coincides with the boundary. An example of this type of code appears in the the furniture layout listing in (Listing 2).

Up to now, the Farnese cell code produces a representation of the palace as it exists in reality. However, if it is possible to imagine changing the code so that facade is drawn differently (Figure 15). That is, it will draw doors on the second floor, the elements of the third floor on the first floor, and elements of the second floor on the third floor.

a prototype system

The prototype of an ED system was developed to explore how a designer would embed computer code into a drawing. In this system, a drawing is created using basic graphic routines of MCL, and the code is inserted into the drawing to perform a variety of operations. The drawing is executed to display the results of the interaction between graphic elements and embedded code. In the following sections, there is a description of the ED environment and two examples that run in the system.

editing an ED

The ED system provides two viewing modes of an executable drawing: graphic editing mode and cell program editing mode. The graphic editing mode provides the user with a basic line tool for making simple drawings (Figures 1-3), while the executable content of the cell code is edited with a simple text editor.

In contrast to the definition of ED presented in previous sections, in which cell code is only stored at the bottom of the tree, in the ED prototype, code can be placed in any part of the tree. This permits placing cell code for spaces that include several of the smallest cells, such as, the definition of the boundary of the office layout (Figure 17). The smallest cells are for placing individual furniture elements. For each floor plan, there is a quad tree generated according to the structure similar to the one in Figure 5. The structure defines the sequence of execution.

The ED system has global parameters which define the size of the smallest cell, such as in the subdivision example (Figure 16) or the size of furniture elements (Figure 17). All copies of the cell code can access the global parameters, but each copy of the code cannot access the variables of its neighbors.

executing an ED

To execute a drawing, the user simply tells the system to execute the current drawing. The actual process that occurs in the background is more complex. With the possibility of code being found on mulitple levels of the drawing, a structured execution sequence is required for predictable results. In our system, we have implemented a top down clockwise execution sequence. The top down sequence is achieved by starting with the highest level quad, the quad which contains the whole drawing. Any code contained within this quad is executed before moving ahead in the execution sequence. Once the top level quad is executed, we begin to examine its subquads, if there are any. Subquads are exectuted in a clockwise rotation starting with upper left hand quad. The top down sequence takes precedent over the clockwise sequence in the execution process.

spatial relationships in an ED

This first example of an executable drawing demonstrates spatial analysis functions, basic graphic manipulation, and global parameter utilization. The final result of executing this drawing will be a drawing which has been subdivided into a series of squares which will be painted a different color depending on where they are located within the drawing (Figure 16).

The process begins with a top level quad which contains a copy of the cell code (Listing 1). This is a pseudo code version of an actual LISP function. It accesses a global parameter "global_minimum_value" which is created by the designer / programmer by using a simple dialog box. When executed, the code will determine if the width of the quad is greater than our global parameter, if it is, the quad will subdivide itself and copy its code into each one of its subquads. This subdivision will continue until the drawing has been subdivided such that the width of the lowest level quad is less than the global parameter. At this point, the code will begin to use its spatial analysis functions to determine where it is located in the drawing. In particular, it will determine whether or not the quad is on a border, and if so, which one. It then paints the quad a different color for each side of the border. The quads in the middle are not painted.

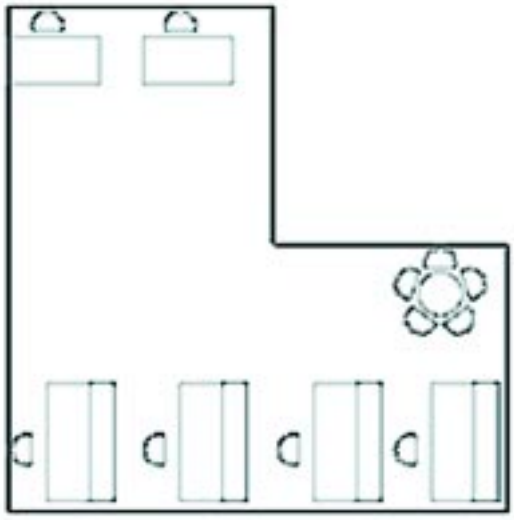

Figure 17. Furniture layout.

```
BEGIN leaf code subdivide
    width:= leaf_width;
    minimum_width:= global_minimum_value;
    IF width > minimum_width
        THEN subdivide leaf
        copy this code into subdivisions
    ELSE IF width < minimum_width
        AND leaves to east = 0
        THEN paint leaf green
    ELSE IF width < minimum_width
        AND leaves to west = 0
        THEN paint leaf orange
    ELSE IF width < minimum_width
        AND leaves to north = 0
        THEN paint leaf blue
    ELSE IF width < minimum_width
        AND leaves to south = 0
        THEN paint leaf red
    ELSE paint leaf grey
    END_IF
END leaf code subdivide
```

Listing 1. Leaf Code.

codifying rules in an ED

The second example demonstrates the ability to codify placement rules for furniture elements in an executable drawing The layout is done as described before (Figure 1). The top level quad is

subdivided to minimum cells which are to correspond to furniture elements. For those subquads that have edges close to the edges of the floorplan, the systems aligns those edges (Figure 2). The execution sequence is then determined (Figure 3). At this point, the user places cell code to draw the boundary walls of the floorplan. (not included here) These quads include three groups of subquads. The first group contains quads Q11-Q12-Q13-Q14 (Figure 3). Next, in each the smallest quads or cells, the user places the furniture code (Listing 2).

Upon execution of this ED, the wall of the floor plan is drawn and the furniture elementes are placed according to the size of the quad, and its location. If the cell is in a corner to the north and of a particular size, then a desk is placed. If it is in a corner and to the south and a particular size, then a conference table is placed--and so on (Listing 2).

```
BEGIN leaf code furniture
    width:= leaf_width;
    length:= leaf_length;
    IF leaf is in a corner
        AND width < maximum desk width
        AND length < maximum desk length
        THEN place desk in leaf
    ELSE IF leaf is in a corner
        AND width > maximum desk width
        AND length > maximum desk length
        THEN place conference table in leaf
    ELSE IF leaf is on a southern wall
        THEN place workstation in leaf
    END_IF
END leaf code
```

Listing 2. Leaf Code.

conclusion

The purpose of the paper was to present the idea of executable drawings (ED) which required showing how to merge code with CAD drawings. A fundamental characteristic of an ED is that the code is stored in an internal representation called an oct tree or quad tree. By adopting these data structures, it is possible to take advantage of the extensive work done in spatial representations in GIS (Sammet 1994). The multiple execution of an ED with different parameters makes possible digital architecture: a design is represented in terms of a program.

The idea was illustrated by means of two simple examples: the Palazzo Farnese and a simple furniture layout. In the first example, it was shown that the window pattern of the facade can be represented by one cell program stored in each of the palace cells inside the building volume. In the second example, the same idea was applied to furniture layout. In both cases, the cell program determines what to do depending upon its position in the building or the layout. The position is represented by the oct tree or quad tree. Upon execution, the cell program only affects the space within the boundaries of its cell. It has no information about the contents of the adjacent cells. However, in the tradition of object oriented programming, it can request the information.

There was no attempt to demonstrate how ED's can be applied in architectural practice or for representing more complex cases than those described here. A first step in this direction was the development of a prototype in object oriented MCL in 1997 with the purpose of validating ED's computationally. A next step would be to implement an ED module as part of a commercial CAD system in order to determine the feasibility of making them an everyday tool.

acknowledgements

We would like to thank Renato Barrera and Hanan Sammet for making us aware of quad trees and their relevance to GIS. We would like to acknowledge the collaboration of those who worked at Pronon in Mexico City, Mexico. Alejandro Villasenor who wrote the very first version of an ED, and Susana Herrera and Hector Jimenez wrote the second one. Dr. Enrique Calderon encouraged me to present my ideas in seminars at the Fundación Rosenblueth, and at the Graficom91 conference, also in Mexico City. In 1997, we received many constructive observations by Mark Gross and Ray McCall of the Sundance Lab, School of Architecture, University of Colorado (SA-UC); Aaron Fleisher , J. P. Protzen, and Anne Vernez-Moudon. And finally, Joan Draper for her help concerning aspects of renaissance architecture at the SA-UC.

references

Aho, Alfred V. and Jeffrey D. Ullman, 1979, *Principles of Compiler Design* (3rd Edition). Reading, MA: Addison Wesley.

Anklam, Patricia, David Cutler, Roger Heinen, Jr., and M. Donald MacLaren, 1977. *Engineering a Compiler*. MA: Digital Press.

Buchmann, Alejandro and Gerzso, J. Michael, 1985. "TM: An Object-Oriented Language For CAD and Required Data Base Capabilities," *in Languages for Automation*, S.K.Chan (ed), Plenum Press.

Downing, F. and U. Flemming, 1981. "The Bungalows of Buffalo," *Environment and Planning B*, Vol. 8.

Foley, J. D. (ed), Hughs, van Dam and Feiner, 1996. *Computer Graphics: Principles and Practice (Systems Programming.* Reading, MA: Addison-Wesley.

Gerzso, J.Michael, 1991. "Bases de Datos Espaciales Usando Quadtrees y Sistemas Orientado A Objectos," *GRAFICON 91*, Mexico City,. Mexico. (Paper on quadtree and executable plans presented at the conference).

Gerzso, J. Michael, 1979. *A Descriptive Theory of Architectural Built Form and its Applications*, Ph.D. Dissertation, University of California, Berkeley, CA.

Mitchell, William J., 1990. *The Logic of Architecture*. Cambridge, MA: MIT Press.

Masson, Georgina, 1966. *Italian Villas and Palaces*. New York, NY: Thames and Hudson.

Murray, Peter, 1963. *The Architecture of the Italian Renaissance*. London: B.T. Batsford Ltd.

Omura, G., 1997. *Mastering AutoCAD 14*, Sybex.

Portoghesi, Paolo, 1972. *Rome of the Renaissance*. London: Phaidon Press.

Sammet, Hanan, 1994. *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison Welsey.

Stiny, George, 1980. "Introduction to Shape and Shape Grammars," *Environment and Planning B*, 7.

Stiny, George and Gips, J., "An Evaluation of Palladian Plans," *Environment and Planning B*, 5.

Tzonis, Alexander and Lefaivre, Liane, 1986. *Classical Architecture. The Poetics of Order.* Cambridge, MA: MIT Press.