CAD GRAMMARS: EXTENDING SHAPE AND GRAPH GRAMMARS FOR SPATIAL DESIGN MODELLING

PETER DEAK, CHRIS REED, GLENN ROWE
School of Computing, University of Dundee, UK

Shape grammars are types of non-linear formal grammars that have been used in a range of design domains such as architecture, industrial product design and PCB design. Graph grammars contain production rules with similar generational properties, but operating on graphs. This paper introduces CAD grammars, which combine qualities from shape and graph grammars, and presents new extensions to the theories that enhance their application in design, modelling and manufacturing. Details about the integration of CAD grammars into automated spatial design systems and standard CAD software are described. The benefits of this approach over traditional shape grammar systems are also demonstrated.

1. Introduction

The aim of the Spadesys project is to investigate how spatial design and modelling can be automated in a generalised way, by connecting similar concepts across the various design domains and decoupling them from the intelligent design process. The core functionality of the system is based on a generative approach to design generation using CAD grammars. The initial part of this paper provides a brief description of shape and graph grammars – two of the base concepts behind CAD grammars. Afterwards, the extensions proposed by CAD grammars are introduced, and the benefits of their use in process plant layout design explained.

2. Shape Grammars

Shape grammars have proved to be applicable in a range of different design domains from camera to building design (Stiny, 1978), which sets them as an appropriate technique to further the goals of generalised design. They employ a generative approach to creating a design using match and replace operations described by a grammar rule set for a domain. There are, however, a number of limitations of shape grammars:

- Engineering domains will have a large set of inherent domain requirements, and each specific design to be generated will have a large set of problem

specific requirements and constraints related to that instance. Creating a grammar rule set that contains the maximal amount of domain knowledge, while remaining flexible and adaptable enough to fulfil the greatest number of designs can result in a large or complex grammar rule set.

- Communicating grammar effectively is difficult; justification for individual grammar rules can be difficult to provide, as they may not have a direct significance on a design, instead playing a linking role where they prepare parts of the design for further grammar rules to work on. This can make maintenance, and understanding of the grammar by anyone who was not involved with its creation difficult.
- In order to use shape grammars in an automatic design generation scenario in most engineering domains, the grammar has to be very detailed and complete, and prohibit the introduction of flaws into the design.
- It is difficult to verify a grammar. A recursive rule set can define an infinite space of possible solutions, and can therefore contain designs that may be flawed in ways that were not anticipated by the grammar designer.
- Current shape grammar implementations do not make it possible to express connectivity; if two line segments in a design share a common endpoint, it is not possible to show whether they are segments of a logically continuous line, or two unrelated lines which happen to be coincident.
- It is Difficult to create a 'designerly' grammar, where the order and application of rules proceeds and a way that makes sense to the user.

3. Graph Grammars

Graph grammars (Plump, 1999) consist of production rules to create valid configurations of graphs for a specific domain. They have been successfully employed in designing functional languages (Barendsen, 1999) and generating picturesque designs (Drewes, 2000). Graph grammar rules contain the match and replace operations for nodes and edges in a network.

There is generally no spatial layout information associated with the nodes and edges; the only relevant data is the types of nodes and edges, and the information about the connections between them. It is therefore difficult to model spatial and graphical designs with graph grammars alone. A desirable feature with graph grammars is that the application of grammar rules keep the design connected as the network is increased.

4. Shapes and Graphs

In typical CAD applications, some of the primitives used to model designs are vertices (points in 3D space), edges (lines connecting points), and faces (enclosed polygons made by edges). This has proven to be an effective way of

representing many types of spatial data, as it allows for a range of editing and analytical operations to be applied to a model. Vertices represent a sense of connectivity between lines. This makes it helpful to display and edit designs and express relationships between lines. Traditional shape grammar systems are not able to deal with CAD primitives directly, as the only components that can be present are shapes or volumes. A CAD design would first have to be represented as shapes or volumes only. Clearly, it would be desirable if the representation does not have to be altered from the one used in CAD software.

There is a clear correlation between these CAD elements and graphs. A design represented using CAD elements can be seen as a graph, with the vertices being the nodes of the graph and lines being the arcs or edges. A CAD design is more complex however, and contains more information, as not only the presence of nodes and arcs, but also their positions and lengths are relevant. Graph grammars have been used in a similar way to shape grammars to design graphs, and an advantage of graph grammars is that there is a sense of connectivity between the elements.

In the Spadesys system, one of the core ideas is to combine shape grammars with graph grammars, inheriting the beneficial features of both concepts. Additionally, in Spadesys there are a number of extensions and new possibilities which are not found in any other shape or graph grammar system. “CAD grammars” are thus an amalgam of the two systems, and inherit benefits from both. In order to address remaining limitations, a number of extensions are proposed, and their implementation in Spadesys is discussed.

5. CAD grammar fundamentals

Rules in CAD grammars are comprised of two parts, the match shape, which is a specification of the shape to be matched, and the replace shape, which is the shape to replace the specified match shape. The design shape is the specification of the current design that is being generated. The matching algorithm looks to find occurrences of the match shape within the design shape, and replace those configurations with the replace shape.

The basic elements for shapes in a CAD grammar system are *points* and *lines*. *Points* are objects which have the numerical parameters *x*, *y* (and *z* in a 3D implementation). Lines are represented by references to two points; *p0* and *p1*. It is important to consider points and lines as objects; as there may be multiple points with the same parameters, but are distinct entities.

Connectivity among two lines can be represented by the two lines sharing a common point instance. In CAD grammars it is important to be able to make this distinction in the design shape and the match/replace shape. The usefulness of this feature can be seen in instances where two lines happen to appear to

share an endpoint, but they are not intended to be logically continuous with regards to the grammar matching algorithm.
Figure 1. shows an example of the connectivity features of CAD grammars. Continuous, connected lines are shown with LineA(Point1, Point2) and LineB(Point2, Point3). Non-connected lines: LineC: Point4, Point5 and LineD: Point6, Point7

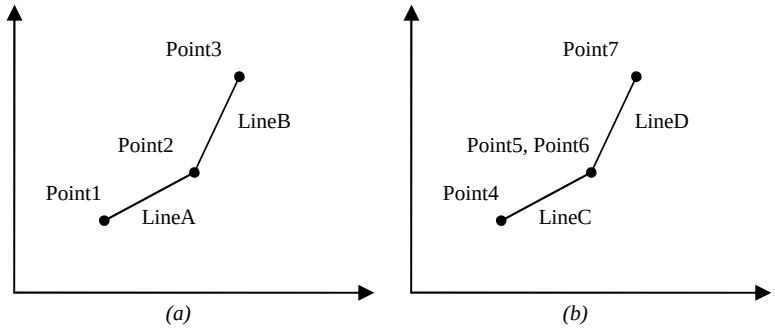


Figure 1. Line Connectedness

In Figure 1(a), the two line segments are connected, which can be seen by the use of only three point instances, with Point2 being shared by both line segments. Figure 1(b) shows spatially identical, non-connected lines, with each line having unrelated point instances.

Similarly, intersecting lines do not logically subdivide into four line segments, as is often the case in traditional shape grammar systems. That intention can be defined implicitly by the presence of a point at the intersection, with the four line segments connected to it. The reason for this is that there are many cases when the result of applying certain grammars results in lines intersecting, but it is not the intention of the grammars to have the intersection produce corners which are matched by other grammar rules. This can prevent accidental, unintended matches in further operations on a shape. For example, the match shape in Figure 2(a) would successfully match the design shape in Figure 2(b), but not that in Figure 2(c).

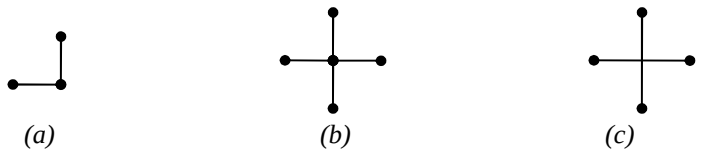


Figure 2. Matching connected lines

6. Extension 1: Length and angle constraints

Parametric shape grammars (Stiny, 1980) allow specification of variable parameters in shape grammars. In Spadesys's CAD grammar, this idea is taken a step further to allow much more customization in the expression of match shapes. Every line in a match shape can have a length constraint. This length constraint is evaluated with the line that is to be matched in the current design when running the matching algorithm.

With regards to many engineering design domains, there may be a need to specify exact line sizes in the match shape, which will result in lines only of that exact length being matched. In CAD grammars, if the length constraint for a line is an exact value, then that line will only match lines of that value. This allows match shapes to be drawn inexactly when actual values are known for the line lengths. Similarly, the length constraint may be specified as a range such as 4-10, in which case all lines of length between 4 and 10 will be matched. Logical operators can be used within the length constraint to allow further control on matching; for example we want to match lines of length 7 or 15, we can set its length in the match shape to 7 | 15. Similar constraints can also be applied to angles between lines, to provide similar flexibility with regards to appropriate angles too.

When the length constraint is set to *proportional*, the behaviour is similar to traditional shape grammars, where any line length will match, provided that all the lines which were matched have the same proportions as the lines in the match shape, making the scale of the match shape irrelevant. When the length constraint is set to *length*, then the exact length of the line is used, as it is shown graphically in the match shape. This is different from exactly specified lengths, as they may be a completely different size from the physical length of the line in the shape.

Due to the complete scripting system embedded within Spadesys, complex mathematical operations can be also used in the length constraint.

7. Extension 2: Modification

Shape grammars (as well as all formal grammars) operate using match and replace operations only. When the aim of a grammar rule is to modify a feature, it is achieved by having a similar match and replace shape, which vary in terms of the intended modification. In standard shape grammars, this approach is fine, since there is no difference between actually modifying the matched shape's elements in the current design, or simply removing it and inserting a new shape as desired. However in CAD grammars there can be meta-information associated with the lines and points in a design, which in many cases would need to be retained.

The most important part of the meta-information of a line is its connectedness; i.e. which other lines it is connected to. It is necessary to be able to state in a grammar rule whether the elements in the current shape should be replaced by new instances of the elements in the replace shape, or whether they should be modified as stated by the elements in the replace shape. The effect of this idea in practice is that grammar rules can not only match and replace, but they can also *modify*. This means that there can be two grammar rules that look identical with regards to the lines and points, but create a completely different result when applied. This is unlike the effect of modification that can be achieved using only match and replace, as seen in the following examples.

The grammar rule in Figure 3 is designed to stretch the match shape regardless of context.

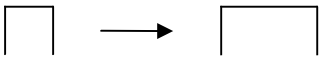
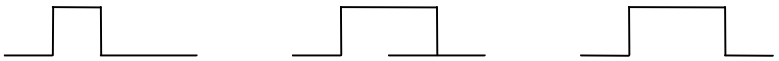


Figure 3. 'Stretch' shape grammar rule

When applied traditionally to the following example, unintended results are produced, so that the design shape in Figure 4(a) changes to the shape in Figure 4(b), rather than what was intended: Figure 4(c).



(a) *(b)* *(c)*

Figure 4. Tradition application of rule

To get the intended result with a traditional shape grammar approach, there would need to be a larger, more complex grammar that takes into account all possible contexts of the original match shape, and modifies the effected portions of the design shape separately. In Spadesys, the above grammar rule from Figure 3 would be represented as the rule in Figure 5.

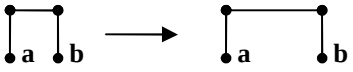


Figure 5. Connectedness in matching

This modification ability is currently implemented using a tagging system. The points in the match and replace shape can be tagged with labels (strings) to signify their correspondence. In figure 5, the 'a' and 'b' labels associated with the points represent their tags. If a point in the replace shape has the same tag as a point in the match shape, then the matched point in the design shape will be modified to spatially match the replace point, as opposed to removing it and

replacing it with a new point. This ensures that the connectivity of the point in the design shape is maintained after the replace operation, and gives the effect of modifying the shape, as opposed to deleting and inserting portions.

8. Extension 3: Line types

In current shape grammar systems, non-terminals are generally represented by a combination of terminals that is unlikely to be found elsewhere in the shape (only in the parts where it is intended to be a non-terminal). This requires complicating the design, and is not safe or efficient. Colour grammars (Knight, 1989) can be used to improve this method, but Spadesys takes the idea a step further by introducing line types. The lines in a match shape have a type rather than a colour. Types are hierarchically structured entities in the same sense as classes and subclasses are in programming languages. The base type is *Line*, and all other line types derive from it. Due to the polymorphic nature of types, if a line in a match shape is of type *Line*, then it will match any type of line in the current design (provided the length constraint is also met).

Generative design often takes place in phases (Stiny, 1978), by gradually lowering the level of the solution from a high-level/abstract design to a low-level/complete design, until it satisfactorily represents the requirements. For example in architecture, the solution can initially start off as a grid of squares covering the approximate layout of the intended design. Applying an initial grammar set in the first phase will add some temporary walls to outline a basic conceptual layout. The next phase can add more detail on the shape of the walls, and position them adequately. Further phases may add additional details such as doors or windows, and so on. By annotating the lines in grammars with their types, we can show clearly which grammars should be applied at the first phase (by setting the match shapes lines to type *grid*) and what phase it will prepare its results for (by setting the replacement shapes lines to type *basicwall*). This opens up more flexible approaches with regards to the progression of the shape generation. One half of the building can be generated right up to the windows and doors phase, and once satisfactory, the other half may be worked on without interference. This region based workflow may be more appropriate in some cases than a phase based one.

Grammar interference is also removed, and the grammars from different phases do not have to be handled separately. A grammar rule will only be applied where it is intended to be applied, on the types of lines it is intended to be applied.

A grammar rule becomes self documenting to an extent, as the line types describe when and where it is applied, and more accurately shows what the designer is trying to achieve with the grammar rule.

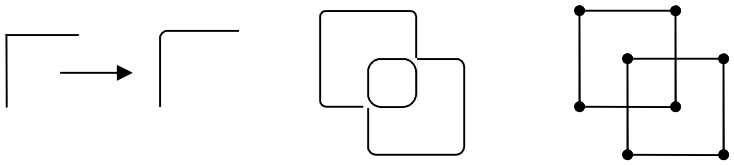
9. Partial Grammars

Spadesys attempts to drive the use of partial grammars as a way to tweak and modify designs in a clear and simple way. Formal grammars theory states that all valid designs must derive from the grammar rule set in play. However, when the aim is to modify existing designs with new features, it may be inefficient to determine their grammar rule set and modify it in a suitable way so that when the design is re-generated it contains the intended changes. It may be simpler having a partial grammar containing only the rules for the new features, and applying that to modify the design. A partial grammar is a reduced set of grammar rules with the intent to modify existing designs, rather than generate a complete design from nothing. For example, given the existing design in Figure 6(a), the aim is to round off the edges to produce Figure 6(b).

(a) *(b)*

Figure 6. An example of modification

The complete grammar would either have to contain all the rules to produce the source shape with the addition of rules to perform the modification, or the rules would have to be modified so that the intended design is produced directly. Either way requires the original grammar, which may not exist and can be difficult to derive. In Spadesys, the grammar rule similar to the one in Figure 7 can be directly applied to any design shape.



(a) *(b)* *(c)*

Figure 7. Rounding

The application of the rule in Figure 7(a) on the design shape in Figure 6(a) demonstrates another useful feature of CAD grammars that derives from the extended connectivity features. Without the connectivity information in the design shape, automatic application would be difficult, as unexpected results can be produced, such as the one in Figure 7(b). With the CAD grammar elements, the initial design shape would be represented as shown in Figure 7(c), with the intended corners specified. This way there can be control over how intersections

are treated, and this control is given to the user, rather than determined by the implementation.
Partial grammars may also be used as the basis for generating the design. In an architectural example, a grammar for a house may be designed in a way, that it is incomplete, and cannot generate a design on its own. However, when provided with a building outline as the current design, it can generate the remainder of the house.
The modification features of CAD grammars as described above in extension 2 are very assistive to the idea of using partial grammars. The modifications can be represented in a compact, context free way by being able to preserve connections between lines and therefore modify the surrounding context suitably. The length constraints feature as described in extension 1 is also a valuable feature for such situations, because a single grammar rule becomes more flexible and can apply to more varying configurations.

10. Application Domain – Process Plant Layout Design

One of the intended domains for evaluation of the Spadesys system is process plant layout design (Patsiatzis, 2004). Process plant design is a complex field with several conflicting factors, and a large number of constraints controlling the design generation. The layout design is the phase that is to be performed by Spadesys, with results of the earlier phases being used as inputs to the design generation. These earlier phases will have determined the equipment and their relationships among them, as well as the overall layout and size constrains imposed by the plant. These requirements have to be represented in the problem code, so the design layout generation can use analyze the constraints and commence layout generation.
Using a CAD grammar based automated design generation system for a domain such as plant design has several benefits:

- One of the main factors in design generation is hazard prevention (Goring, 1993). The grammar rules to generate the layout can be designed in a way to prevent many sources of hazard relating to spatial configurations. Since the grammar rules are the building blocks the generate the design, these *blocks* can be made so that they only fit together safely.
- Cost and space reduction is supported by the iterative approach provided by a system such as Spadesys. After selecting the most appropriate candidates from an initial set of designs proposed by the system, each can be further branched out or refined further to increase the suitability of the results. This allows a controlled level of user interaction to ensure the users are content with the way the designing is being performed.

11. Conclusion

CAD grammars provide a more flexible approach to the applications of shape grammars. The enhanced matching features allow the construction of smaller

grammar rule sets. Grammar rules can adapt and match a larger number of relevant configurations due to the length and angle constraint features. The modification features, which allow grammars to directly modify designs introduces a new dimension to design generation.
The emergent features of shape grammars, where large complex designs can be generated from a few simple rules are still present; however the ability to define clear and predictable grammars is also enhanced. Using the *line-types* extension has not only functional benefits; in addition grammar rules can become self-documenting, with their features and intentions made clear by the visible type name of every line in the match and replace shapes.
The polymorphic nature of line-types provides more power to the grammar designer, by being able to further specify the intentions of grammars. This allows for the creation of more *'designerly'* grammars, where the design process can flow consistently with the design intention of the grammar designer. The use of CAD grammars allows a homogeneous model to be used for the design representation and the tools used to generate it.

12. References

1. G. Stiny: "Introduction to Shape and Shape Grammars" Environment and Planning B, 7, 1980.
2. T. W. Knight, "Color grammars: designing with lines and colors" *Environment and Planning B: Planning and Design* 16: 417-449, 1989.
3. T. W. Knight, "Shape grammars: six types" *Environment and Planning B: Planning and Design* 26: 15-31, 1999.
4. G. Stiny, W. J. Mitchell, "The palladian grammar" *Environment and Planning B, 5*, 5-18, 1978.
5.
6. E. Barendsen and S. Smetsers, Graph rewriting aspects of functional programming, 17, Chap. 2, pp. 62–102, 1999.
7. D. Plump, ''Handbook of Graph Grammars and Computing by Graph Transformation'' Vol. 2, Chap. 1, pp. 3–61, World Scientific, Singapore, 1999.
8. F. Drewes, "Tree-based Picture Generation" Theoretical Computer Science, 246(1), pp1-51, 2000.
9. R. Reffat, "Utilisation of artificial intelligence concepts and techniques for enriching the quality of architectural design artefacts. *Proceedings of the 1st International Conference in Information Systems, Department of Computer Science, Cairo University, Egypt*, CS5:1-13, 2002.
10. Patsiatzis,D.I., Knight,G., Papageorgiou,L.G. (2004). An MILP Approach to Safe Process Plant Layout. *Chemical Engineering Research and Design, Transactions of the Institution of Chemical Engineers, Part A* 82, 579-586
11. M. Goring, H. G. Schecker. *HAZEXPERT: An integrated expert system to support hazard analysis in process plant design*. Computers Chemical Engineering, 17:429--434, 1993.