# CAD GRAMMARS

*Combining CAD and Automated Spatial Design*

PETER DEAK, GLENN ROWE, CHRIS REED
*University of Dundee Applied Computing Department, UK*

**Abstract.** 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 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 with regards to traditional shape grammar systems are explored.

## 1. Introduction

The aim of the Spadesys project is to investigate how spatial design can be automated in a generalized way, by connecting similar concepts across the various design domains and decoupling them from the intelligent design process. The primary focus is on engineering design domains, where there is a large number of domain specific constraints and requirements, as well as problem specific constraints and requirements for each design being produced.

Shape grammars have proved to be applicable in a range of different design domains from camera to building design, which sets them as an appropriate technique to further the goals of generalized 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 issues or limitations associated with 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.

## 2. 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.

## 3. 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. Using a design from a CAD application in a shape grammar system would require conversion of the designs representation to be compatible with the components of the specific system. 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.

## 4. 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.

The following is an example of the connectivity features of CAD grammars:

Point1: X=2, Y=3
Point2: X=4, Y=4
Point3: X=5, Y=6
Point4: X=2, Y=3
Point5: X=4, Y=4
Point6: X=4, Y=4
Point7: X=5, Y=6

<u>Continuous, connected lines:</u>
LineA: Point1, Point2
LineB: Point2, Point3

<u>Non-connected lines:</u>
LineC: Point4, Point5
LineD: Point6, Point7

Visually, both LineA→LineB and LineC→LineD appear similar:

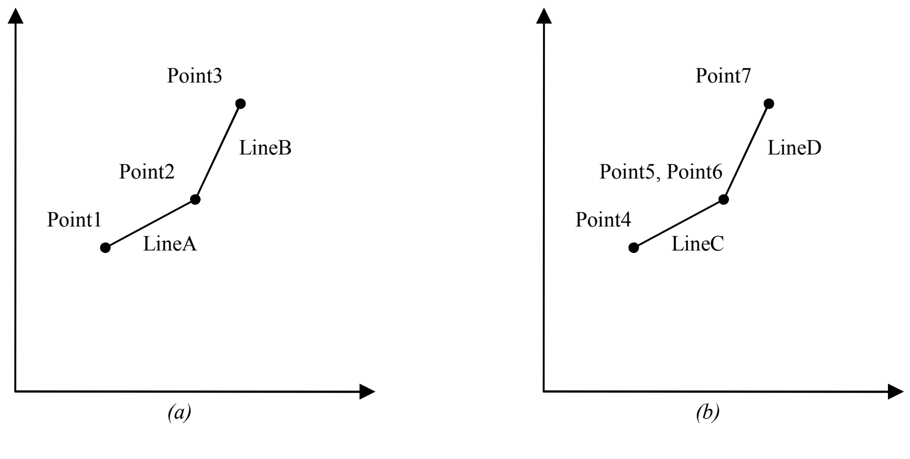


*Figure 1.* Line Connectedness

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, unless this is specified as the intended operation of the grammar rule, by setting properties of lines in the replace shape to join-and-divide at intersections. 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).

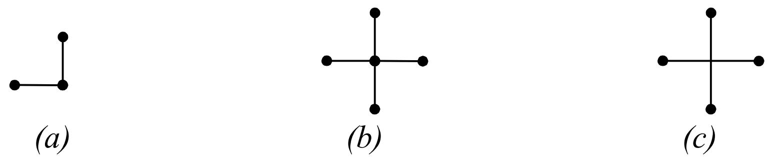


*Figure 2.* Matching connected lines

## 5. Extension 1: Length and angle constraints

Parametric shape grammars (Stiny 1980) allow specification of variable parameters in shape grammars. In Spadesys's CAD grammar, the definition of parameters and their usage is enhanced. Similar work has been done in Liew (2004).

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 most traditional shape grammar systems, 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.

## 6. 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.

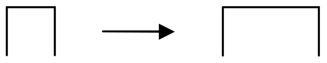


*Figure 3.* 'Stretch' shape grammar rule

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).

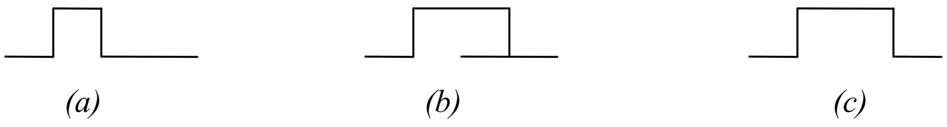


*Figure 4.* Tradition application of rule

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.

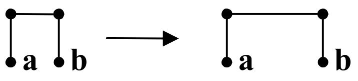


*Figure 5.* Connectedness in matching

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.

## 7. Extension 3: Line types

Initially, non-terminals in shape grammar systems have been 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. Colours (Knight 1989) or Weights (Stiny 1992) can be added to grammar rules to improve this method, but Spadesys introduces polymorphic and hierarchical ‘Line types’ as a parameter for lines in shapes. 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.

The polymorphic nature of types allows control over the generality of grammar rules: from being applicable to any relevant part of a design (when the line type is set to the base type 'Line') to domain or problem specific locations. Grammar designers can incorporate this into the abstraction of the operations; create domain independent rules such as a rule that extends the dimensions of a rectangle (Which can apply to lines of all types), to domain specific rules such as a rule that adds an alcove to lines of type 'WoodenWall' and its derivatives.

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.

## 8. 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. 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).

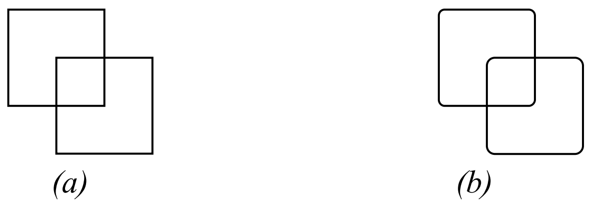


*Figure 6.* An example of modification

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.

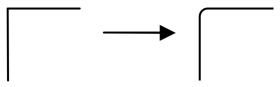


*Figure 7.* A rounding rule

The application of the rule in Figure 7 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 require the original shape to contain additional shape labels regarding which corners should be treated as ones to be rounded; otherwise unexpected results can be produced, such as the one in Figure 8.

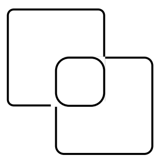


*Figure 8.* Incorrect rounding

But with the CAD grammar elements, the initial design shape would be represented as shown in Figure 9. Using lines and points as components in the design means that additional labeling is not required, as the intention of corners and intersections is implicit. Liew (2004) presents an alternative method for controlling the application of grammar rules.

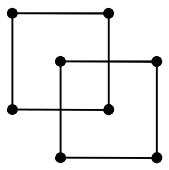


*Figure 9.* CAD grammar representation

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.

## 9. Implementation

### 9.1 CAD GRAMMAR SYSTEM: MATCHING ALGORITHM IMPLEMENTATION

The matching algorithm is used to determine if, and where the match shape in a grammar rule can be found in the current design. It is a recursive algorithm that is carried out on each line of the current design. This is the pseudo code of the matchLine algorithm:

```
matchLine(Line source, Line match)
    1. If we have already passed this match line, return true
    2. If the number and angle of every branch of match is not the same as
       the number and angle of every branch of source, return false
    3. If match's length constraint does not prove valid with source, return
       false
    4. If source's type is not of match's type or a subtype of that, return
       false
    5. For each corresponding branch of match and source
        a. matchLine(sourceBranch, matchBranch)
```

This algorithm attempts to traverse the match shape and the current segment of the design shape in a mirrored way by passing and attempting to match each line and their branches recursively. Visited lines are ignored, which breaks up the traversal and prevents infinite loops in circular designs. This also results in the shapes (which have the connectivity structure of a graph) to be parsed as a tree.

## 9.2 CAD GRAMMAR SYSTEM: REPLACEMENT ALGORITHM IMPLEMENTATION

The replacement phase occurs when the match shape has been found at a location in the design shape. An important factor to consider is that the replacement shape must have the same scale and alignment as the match shape has in the design shape as it was located. Therefore, the transformation matrix between the match shape and its corresponding shape in the current design has to be determined. The transformation matrix encapsulates the scale, rotation and translation that is required to convert the replace shapes alignment to the correct configuration. The first line from both the match and the design shape is used as the point of reference (the first line can be specified, but by default it is the line that was placed first), and the transformation matrix is determined from these lines only. The scale and rotation can be obtained directly from the two arbitrary lines. The translation can then be obtained by applying the scale and rotation to the line from the match shape, and finding the offset between the corresponding endpoints of the match and source line.

This matrix is applied to the replace shape before its insertion into the design shape. During insertion, if any points have a matching tag in the match shape, then the point in the source is modified to correspond to the point in the replace shape, rather than replaced.

## 10. CAD grammar applications

### 10.1. AUTOMATED DESIGN GENERATION – SPADESYS

At the core of the Spadesys project is the CAD Grammar, which provides an implementation of the theories described above. There is no tie to any of the phases of the design process (Reffat, 2002); a base design can be generated automatically and exported for further manual refinement; an existing base design may be imported for detailing; or the entire process may take place within the software. The application and use is all down to the grammar sets in play, which may be domain independent (simple CAD-like modelling operations) or domain dependent (grammar to generate buildings/phones/circuit boards etc.).

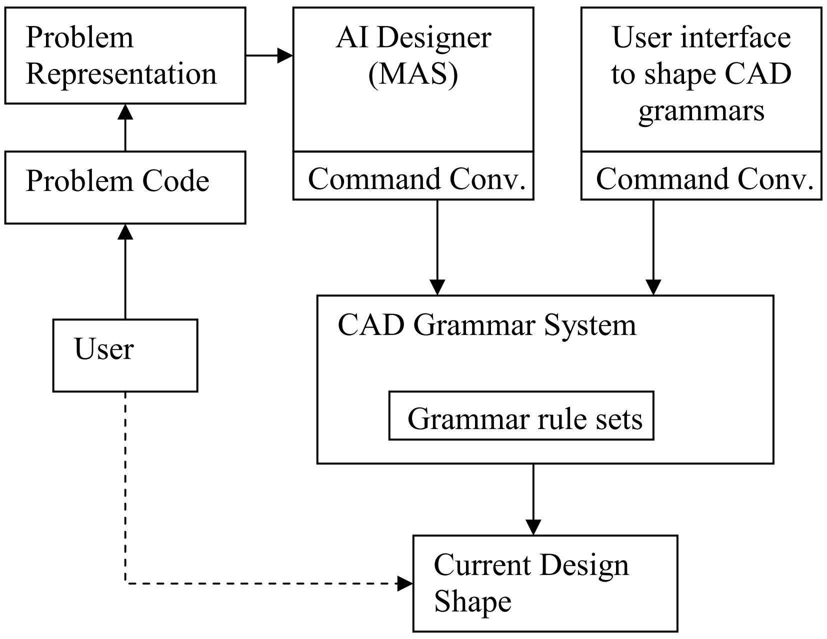


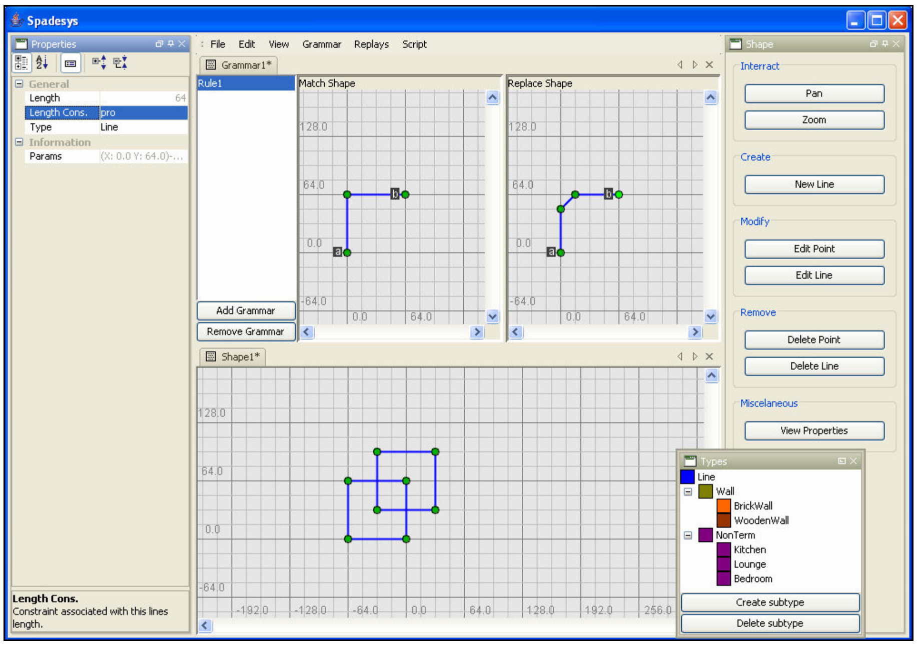
*Figure 10.* Architecture of Spadesys

As demonstrated in Figure 10, the first step is defining the problem that is to be solved. This involves producing problem code, which details the constraints and requirements of the design that is to be generated. The code is converted into a native representation that can be loaded into the intelligent designer. An existing design may also be imported or created by the user based on the grammar set to be used. Partial grammars may require an existing design of some form to operate on. The grammar rule sets that are to be used must also be specified. These can be made up of domain independent basic construction grammars, but will most likely contain domain dependant sets that contain grammar rules specifically for the generation of the intended artefact.

The intelligent designer then uses the data from the problem code to make decisions regarding the design's generation. It is currently implemented as a multi-agent system, but can be replaced in the future with a different kind of *reasoner*.

The command conversion layer for the intelligent designer translates decisions made by the intelligent designer into commands for the CAD grammar system, such as 'apply grammar rule *x* at location *y*, with these parameters'. The command conversion layer for the user interface does a similar job by taking the actions of the user and converting that into the intended commands. At its core, it is the same shape grammar system performing the same process, without any knowledge as to whether it is a human user or the intelligent designer making the decisions.

The Spadesys application provides a feasible motive for the design of CAD grammar rules.



General

| | |
|---|---|
| Length | 64 |
| Length Cons. | pro |
| Type | Line |

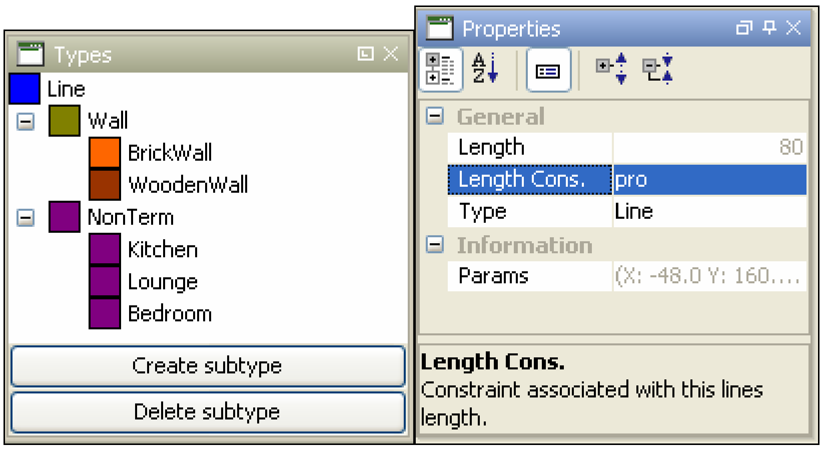
Information

| | |
|---|---|
| Params | (X: 0.0 Y: 64.0)-... |

**Length Cons.**
Constraint associated with this lines length.

*Figure 11*. Main Spadesys interface

Figure 11 shows the entire interface with the rounding example from section 8. The main user interface to the application is user configurable, with all windows being dockable to any portion of the screen, in the same way as many Integrated Development Environments (IDEs). The top half of the central viewport has a grammar rule set opened with the rule to round off edges. The bottom half contains the design shape which is to be modified. All shapes can be edited by the user using the provided CAD-like toolset.



| | |
|---|---|
| Length | 80 |
| Length Cons. | pro |
| Type | Line |
| Params | (X: -48.0 Y: 160.... |

**Length Cons.**
Constraint associated with this lines length.

*Figure 12*. Types and Properties window

The ‘Types’ window shown in Figure 12 allows the definition of line types as described in extension 3, section 7. The tree structure models the hierarchical relationships between the types defined. Lines in the match and replace shape can have their type set to any one of the values in this tree. Each type can have an associated colour, which can be used as the draw colour for every line of that type.

The properties window from Figure 12 dynamically displays the properties of the currently selected object. The editable parameters of lines, points and types etc. can be changed here.

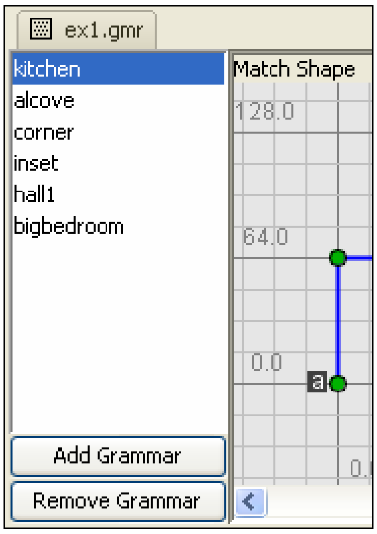


*Figure 13.* Grammar rule list

The image from Figure 13 demonstrates that the currently shown grammar rule is the one selected from the list on the left, which contains all the rules in the current grammar set. Each individual rule can be given a name to be able to clearly determine their role.

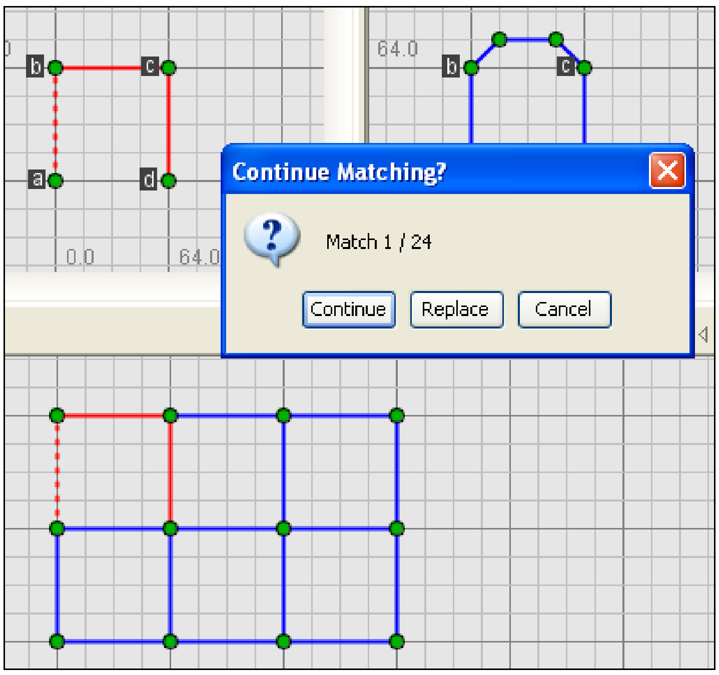


*Figure 14.* Manual matching and replacement

Figure 14 demonstrates the manual application of a grammar rule to the design shape. The interface presents to the user the location and relative orientation of the match shape as it is found in the design shape. One of the lines from the match and design shape are dashed, to show their correspondence. The user may cycle through all possible matches, and apply the replace shape at the intended location.

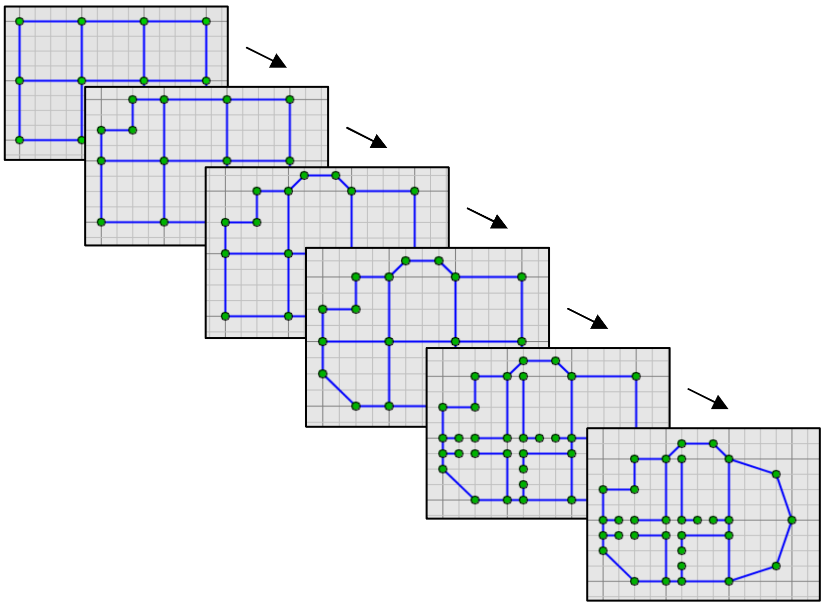


*Figure 15.* Design progression

Using the embedded scripting system, the grammar application process can be automated. Various strategies can be used for applying the grammar rules. Simple, random algorithms as well as advanced logical reasoners can be written using the BeanShell scripting language, and executed on a design. The scripting language is similar to the Java language, and allows for rapid development of scripts. The *intelligent designer* itself is implemented as a script. Figure 15 shows the automatic progression of a design, as grammar rules are selected and applied.

## 10.2. CAD SOFTWARE

Most common modelling operations available in CAD software, such as extrude, bevel, chamfer, slice etc. can be represented using CAD grammars.

As en example, the *extrude* operation found in the majority of cad software can be represented using this parametric cad grammar:

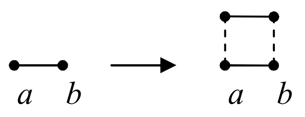


*Figure 16.* Extrude operation

The letters '*a'* and '*b*' in Figure 16 represent tags as described in the modification extension description of this paper. This is to ensure that the connectivity states of the associated points are maintained after the rule is applied. The dashed lines represent the parameterized values; their length should be alterable in some way through the interface. This operation would generally be performed on a polygon face rather than a single edge, in which case the same rule is applied to all edges on a face, rather than a single one.

Similarly, a 2D bevel operation could be represented as:

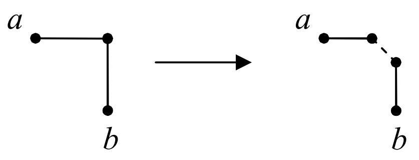


*Figure 17.* Bevel operation

## 11. Conclusion

CAD grammars provide a 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; since traditional grammars can be created that do not take advantage of CAD specific features. The ability to define clear and predictable grammars is also enhanced, as the extended features can be applied where and when desired. Based on a design's requirements, the tradeoff between predictability and emergence can be made by the grammar designer. Similarly, predictable and emergent grammars can be mixed and used together within the same problem.

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.

## References

Barendsen, E and Smetsers, S: 1999, Graph rewriting aspects of functional programming, in H Ehrig, G Engles, HJ Kreowski and G Rozenberg (eds), *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific, pp. 62–102.

Knight, TW: 1989, Color grammars: designing with lines and colors, *Environment and Planning B: Planning and Design* **16**: 417-449.

Knight, TW: 1999, Shape grammars: six types, *Environment and Planning B: Planning and Design* **26**: 15-31.

Liew, H: 2004, SGML: A Meta-Language for Shape Grammars, *PhD dissertation*, Massachusetts Institute of Technology, Cambridge, Mass.

Plump, D: 1999, Term Graph Rewriting, in H Ehrig, G Engles, HJ Kreowski and G Rozenberg (eds), *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific, pp. 3–61.

Reffat R: 2002, Utilisation of artificial intelligence concepts and techniques for enriching the quality of architectural design artefacts. *Proceedings of the 1st International Conference in Information Systems* **5**, 1-13.

Stiny, G: 1980, Introduction to Shape and Shape Grammars, *Environment and Planning B* **7**(3): 343-351.

Stiny, G and Mitchell, WJ: 1978, The Palladian grammar, *Environment and Planning B* **5**: 5-18.

Stiny, G: 1992, Weights, *Environment and Planning B: Planning and Design* **19**: 413-430.