A graph-theoretic implementation of the Rabo-de-Bacalhau transformation grammar

Tiemen Strobbe [a,*], Sara Eloy [b], Pieter Pauwels [a], Ruben Verstraeten [a], Ronald De Meyer [a], Jan Van Campenhout [c]

[a] Department of Architecture and Urban Planning, Ghent University, J. Plateaustraat 22, 9000 Ghent, Belgium.

[b] Department of Architecture and Urbanism, Instituto Universitário de Lisboa (ISCTE-IUL, ISTAR-IUL), Av. das Foras Armadas 1649-026, Lisbon, Portugal.

[c] Department of Electronics and Information Systems, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium.

[*] Corresponding author, J. Plateaustraat 22, 9000 Ghent, Tel +32 9 264 3880, tiemen.strobbe@ugent.be

Short title: Graph-theoretic shape grammar implementation

Number of pages: 49

Number of tables: 2

Number of figures: 16

A graph-theoretic implementation of the Rabo-de-Bacalhau transformation grammar

Abstract:

Shape grammars are rule-based formalisms for the specification of shape languages. Most of the existing shape grammars are developed on paper and have not been implemented computationally, so far. Nevertheless, the computer implementation of shape grammar is an important research question, not only to automate design analysis and generation, but also to extend the impact of shape grammars towards design practice and computer-aided design tools. In this paper, we investigate the implementation of shape grammars on a computer system, using a graph-theoretic representation. In particular, we describe and evaluate the implementation of the existing Rabo-de-Bacalhau transformation grammar. A practical step-by-step approach is presented, together with a discussion of important findings noticed during the implementation and evaluation. The proposed approach is shown to be both feasible and valuable in several aspects; We show how the attempt to implement a grammar on a computer system leads to a deeper understanding of that grammar, and might result in the further development of the grammar; We show how the proposed approach is embedded within a commercial CAD environment to make the shape grammar formalism more accessible to students and practitioners, thereby increasing the impact of grammars on design practice; and the proposed step-by-step implementation approach has shown to be feasible for the implementation of the Rabo-de-Bacalhau transformation grammar, but can also be generalized using different ontologies for the implementation.

Keywords:

shape grammar, implementation, graph grammar, architectural design

1. Introduction

Spatial grammars are rule-based, generative and visual formalisms for the specification of spatial languages. Spatial grammars include set grammars, graph grammars, shape grammars, and other kinds of grammars for describing spatial languages. This ‘uniform treatment’ of grammars is introduced in the work of Krishnamurti & Stouffs (1993), and also used in later work of Hoisl & Shea (2011) and McKay et al. (2012). The potential of shape grammars (a specific kind of spatial grammar) as a theoretical framework for analyzing and generating (architectural) designs has been demonstrated through a broad range of formal studies (Koning & Eizenberg, 1981; Duarte, 2005; Flemming, 1987; Stiny, 1977). However, many existing grammars in architectural design (and other design disciplines) are developed on paper, and relatively few grammars have been implemented computationally, so far. Some exceptions can be found, including the work of Aksamija et al. (2010), Granadeiro et al. (2013), and Grasl (2012). Nevertheless, the computer implementation of shape grammars remains an open research question, because there seems to be no definite answer in the literature on how shape grammars can be implemented to a computer system. For example, a recent overview of McKay et al. (2012) summarizes the key limitations, benefits, and open challenges of the main representative shape grammar implementations to date. The question how to implement shape grammars is also an important research question, because computer implementations are beneficial in many cases, including: to automate several aspects of design analysis and generation (especially for grammars that are too extensive to explore manually), to learn from the computer implementation about the design of the shape grammar itself, and to extend the impact of shape grammars towards design practice and computer-aided-design tools by providing tools to apply shape grammars in practice.

In this paper, we describe a method for the implementation of a shape grammar, originally developed on paper, on a computer system using a graph-theoretic representation of this grammar. We start from a literature review of previous research efforts in which spatial grammars are implemented to a computer system (*Section 2*). In particular, the definition and characteristics of different kinds of spatial grammars are compared, thereby focusing on shape grammars since they are often used for analyzing and synthesizing architectural and creative designs. Also, an overview of previous shape grammar implementation approaches is given. Next, we describe the research method, in which we point out a practical step-by-step approach for the computer implementation of shape grammars (S*ection 3*). In the following section (*Section 4*), the proposed approach is evaluated through the implementation of the Rabo-de-Bacalhau (RdB) transformation grammar, originally developed by Eloy (2012). In particular, three relevant types of rules that are used in the RdB transformation grammar are implemented: (1) assignment rules, (2) rules to connect spaces by eliminating walls, and (3) rules to divide spaces by adding walls. A discussion of several issues encountered during the implementation and an evaluation of the proposed approach is given in *Section 5*. Finally, conclusions and future research are described in *Section 6*.

The work presented in this paper contributes to the current state of the art in shape grammar implementations in several ways. First and foremost, a practical step-by-step approach is presented for the computer implementation of a shape grammar. While the proposed approach builds further on existing research on the graph-theoretic representation of shapes, such as recent work of Grasl (2013) and Wortmann (2013), the proposed approach is also different because it can be applied in different contexts, ranging from simple shape grammars to more complex grammars. Second, the implementation of the RdB transformation grammar (Eloy, 2012) is in itself a useful contribution, because it demonstrates how shape grammar implementations can be applied in architectural design practice. In most papers on shape grammar implementations, the approach is validated through a rather straightforward "showcase" grammar, while the RdB transformation grammar is more complex due to the parallel representation of designs, rule conditions, etc., but it is also an example of a grammar that is being used in practice. One of the most common criticisms to shape grammars is that there is little evidence of their use in practice, so the implemented RdB transformation grammar serves as an example application in architectural design practice, in particular, the development of a (semi)automated methodology for supporting mass housing refurbishment. Third, the case study of implementing the existing RdB transformation grammar reveals several findings on how grammar designers can learn from the implementation about the design of the original grammar, and about the implications on the original grammar itself. Finally, the proposed approach is embedded within a commercial computer-aided design (CAD) environment to make the shape grammar formalism more accessible to students and practitioners (architects, product designers), and therefore, it might increase the impact of shape grammars on design practice.

2. Related work

In this section, we compare the definition and characteristics of different kinds of spatial grammars. In particular, we focus on shape grammars since they are often used for analyzing and synthesizing architectural and creative designs. Also, we provide a literature review of previous research efforts in which a graph formalism is used for shape grammar implementations.

2.1. Spatial grammars

Grammars, in general, are formal mathematical structures for specifying languages. All different kinds of grammars (string grammars, shape grammars, graph grammars, and set grammars) share certain definitions and characteristics (Krishnamurti & Stouffs, 1993). In particular, a grammar is defined as a 4-tuple (N, T, R, I) where:

- *N* is a finite set of non-terminal entities;
- *T* is a finite set of terminal entities;

- *R* is a finite set of rewriting rules or productions;
- *I* is an initial entity, a subset of N ∪ T.

A rewriting rule (or production) has the form *lhs* → *rhs* and can be considered as an "IF-THEN" statement. The left hand side (*lhs*) contains entities from *T* and *N*, but cannot be empty. The right hand side (*rhs*) also contains entities from *T* and *N*, but can be empty. A rule can be applied if the left hand side of the rule matches a part of the given object, under a certain transformation (*f*). If this is the case, the matching part of the given object is replaced by the right hand side of the rule, under the same transformation *f*. As a result, a grammar defines a language that contains all objects generated by this grammar.

Spatial grammars, in particular, are specific kinds of grammars that operate on objects in an Euclidean space E^2 (for two-dimensional objects) or E^3 (for three-dimensional objects). Krishnamurti and Stouffs (1993) describe four kinds of grammars that can serve as spatial grammars: string grammars, set grammars, graph grammars and shape grammars. String grammars deal with a single string of symbols, in which each symbol corresponds to a geometrical entity represented as a graphical icon. Set grammars deal with spatial objects that are described as sets of geometrical entities. For example, three-dimensional solids can be represented as collections of faces, edges and vertices. An example that combines string and set grammars can be found in the work of Woodbury et al. (1992). Graph grammars deal with a set of entities (nodes) where some pairs of entities are connected by links (edges). A common characteristic of string grammars, set grammars and graph grammars is that spatial objects are represented using symbolic entities: strings, sets and graphs, respectively.

Unlike other spatial grammars, shape grammars operate directly on spatial objects (shapes), rather than through symbolic entities (Stiny, 2006). A powerful feature of shape grammars is that shapes and their properties can be reinterpreted continuously during the process of rule application, allowing *emergence* of shape features or properties that are not apparent in the initial definition of the shapes (Knight, 2003). In shape grammar theory, algebras are used to represent shapes. An algebra *Uij* consists of a set of geometrical shapes defined in dimension i = 0, 1, 2 or 3, which are points, lines, planes and solids, respectively. These shapes are combined in a dimension j ≥ i. Also, labels and weights are introduced to define new algebras *Vij* and *Wij* (Stiny, 1991).

In the architectural design domain, shape grammars are often used for analyzing and generating creative designs (Koning & Eizenberg, 1981; Duarte, 2005; Flemming, 1987; Stiny, 1977). Relatively few of such shape grammars have been implemented to a computer system, with some exceptions available (Aksamija et al., 2010; Grasl, 2012; Granadeiro et al., 2013). Typically, the user of such unimplemented shape grammars is meant to interpret the grammar and manually apply the rules in order to generate designs (see the work of Chase (2002) for an overview of interaction strategies with shape grammars). For computer implementations of shape grammars, on the other hand, the computer system should automatically determine where and how rules are to be applied. While human designers are extremely good at recognizing possible rule applications and readily make meaning from visual fragments, there is general agreement that the ability of computer systems for shape recognition and interpretation is below human capacities. For example, Jowers (2010) has successfully applied automatic object recognition techniques (in particular, a method called Hausdorff distance) to interpret shapes without an underlying representation. However, this implementation is also limited; for example, it may not be able to identify spaces in a floor plan, while a (trained) human person can almost immediately detect different spaces by just looking at the floor plan. While computer systems often rely on predefined representations in order to interpret given information, shape grammars rely on emergence and continuously changing representations, thus making them not particularly amenable for computer implementation. Even for shape grammars that do not support emergence, the detection of applicable rules is a complex task to solve for computer, such as finding subshapes for rule application (Krishnamurti, 1981).

A large spectrum of shape grammar types can be identified (Knight, 1999; Yue & Krishnamurti, 2014), including subshape-driven versus label-driven shape grammars, nonparametric versus parametric shape grammars, rectilinear versus curvilinear shape grammars, and shape grammars with or without emergence enabled. As Yue & Krishnamurti (2014) point out, the complexity and choice of the implementation approach depends on the type of shape grammar to be implemented. In this paper, we propose an implementation approach that consists of translating a shape grammar to a graph-theoretic equivalent grammar. Graphs provide an elegant way to describe topological compositions and incidence relations of spatial objects, but can also account for geometrical properties by associating attributes to the graph objects. Moreover, practical solutions and algorithms for (sub)graph matching and automatic rule application exist in the literature (Geiβ et al., 2006; Taentzer, 2004). This graph-based approach is applicable for shape grammars that are either subshape-driven or label-driven and support parametric shapes. Another important benefit of using such a graph-based implementation approach is that shape grammars can also be implemented with emergence enabled. A short discussion of this can be found in Section 3.1 and other research work of Grasl (2013) and Wortmann (2013), however, enabling emergence is not the main topic of this paper. An aspect that is not considered in this paper is how to support curvilinear shapes, but a good discussion of this can be found in the work of Jowers & Earl (2011). An overview of existing shape grammar implementations, including graph-based approaches and other approaches, is presented in the following section.

2.2. Previous implementation approaches

The implementation of shape grammars has been the subject of many research efforts since the original conception of shape grammars in the 1970s (Stiny & Gips, 1971). The dilemma, addressed by Gips (1999), is the tension between the visual nature of shape grammars and the inherently symbolic nature of computer representations and processing. An overview of more recent research efforts is given in the work of Gips (1999 and Chase (2010). Chase (2010) summarizes the main representative shape grammar implementation systems (Li et al., 2009; Trescak et al., 2012; Hoisl & Shea, 2011; Jowers & Earl, 2011; Ertelt & Shea, 2010; Correia et al., 2010). These systems are analyzed and compared in terms of form, semantics, definition interface and generative capabilities. McKay et al. (2012) analyze these systems according to four characteristics: representation and algorithms, user interaction and interface, support for particular problems, support for specific stages of the development process. Currently available implementations, although still prototypes and each focusing on just a few particular aspects, have made valuable contributions on enabling subshape recognition, emergence, parametric rules, curvilinear shapes, and present more user friendly interfaces and flexible representation processes.

A fundamentally different approach towards shape grammar implementation is the use of graphs as an underlying framework for representing shapes. Graphs are data structures that represent a set of entities (nodes) where some pairs of entities are connected by links (edges). Graphs offer a natural framework to model spatial entities (solids, faces, edges, and vertices) and the relations between these entities. The use of graphs to represent spatial shapes or designs is not uncommon in the architectural design domain. For example, Fitzhorn (1990) uses graphs to represent three-dimensional solids and Steadman (1976) describes a graph-theoretic representation of architectural arrangements. If graphs are used to represent shapes or designs, graph rewriting systems can be used to create new graphs out of an original graph, similarly to how it occurs for shape grammars. Graph grammars are used for divergent purposes, but they can also be used to develop formal languages of spatial objects. Among the first attempts to describe such formal languages is the approach of Fitzhorn (1990). In this approach, a graph grammar is defined to generate boundary representations of three-dimensional solids. These solids are defined as sets of geometrical entities (solid, face, edge and vertex) and their corresponding topological relations. The production rules of the grammar are defined as Euler operators in order to generate solids that are syntactically correct. This approach was adopted and extended in the work of Heisserman (1994). In this approach, three-dimensional solids are represented as labeled boundary graphs, and boundary solid grammars are used to develop spatial languages. Other examples include the work of Shea & Cagan (1999), in which graph-like shapes are applied to produce structural forms, and the work of Helms & Shea (2012) on representing designs as graphs. In other recent work, it has been demonstrated how graph grammars can also represent parametric shape grammars, and how emergence, a foundational feature of shape grammars, can be supported (Grasl, 2013; Wortmann, 2013). In recent work of Grasl & Economou (2013), a graph-based shape grammar library called GRAPE is proposed that provides a general framework for graph-based shape grammar implementations.

In conclusion, several shape grammar implementation systems are available, each having a specific focus and purpose. The shared focus of these systems is to allow designers or shape grammar users to implement their shape grammar on a computer system. Still, computer implementations of complex shape grammars are seldom. An interesting counterexample can be found in the work of Grasl (2012), in which a graph-theoretic equivalent of the Palladian shape grammar is described that can generate the same language of Palladian villas as the original shape grammar introduced by Stiny & Mitchell(1978). In the next section, an approach for the implementation of shape grammars is proposed that, on the one hand, builds further on existing research on the graph-theoretic representation of shapes and shape grammars (Grasl, 2013; Wortmann, 2013), but it is also more general than previous approaches and it can be applied in different contexts.

3. Method: Implementing a shape grammar

In order to translate a shape grammar, specified on paper, to a computer-amenable and graph-theoretic grammar, several steps are needed. In this section, a step-by-step approach is given to define a graph-theoretic representation of the shape grammar to be implemented.

3.1. Step 1: Defining the ontology

The first step in the proposed translation of a shape grammar to a graph-theoretic grammar is to construct an ontology that defines the node types, node properties and relations between the nodes. A graph is a mathematical structure that represents relations between nodes. Therefore, the node types that are used and the possible relations between different node types of the graph-theoretic grammar must be defined. In other words, this corresponds to defining an ontology beforehand, which allows computers to more easily interpret given visual information in terms of this predefined ontology. The predefined ontology should describe the different entities considered and how they relate to other entities (spaces, walls, edges, vertices, and other kinds of geometric, semantical, or spatial entities). This ontology determines what information can be expressed in the computerized grammar and how this information will be interpreted by the computer system. The definition of an ontology depends on the given shape grammar and on the envisioned functionality of the grammar implementation; for example, if the given shape grammar concerns only two-dimensional shapes, the ontology needed is more limited than for shape grammars that operate with more complex semantic entities (such as walls, spaces, and other architectural concepts).

As an example, an ontology with six different node types is considered: *vertex*, *edge*, *space*, *wall*, *door*, and *window*. With such an ontology, the computer system is able to recognize both geometrical entities (vertex and edge) and non-geometrical entities (space, wall, door and window. Also, seven different relations between the predefined node types are defined: *edge-vertex*, *space-edge*, *space-vertex*, *edge-*

wall, *wall-door*, *wall-window*, and *access-to*. In the domain of graph grammars, a type graph provides a useful way to represent which node types are allowed and which edge types can be used to define relations between the nodes (which is exactly the ontology). Figure 1 shows the type graph for the six node types and seven edge types. Both geometrical and non-geometrical node types are shown as circles, using different colors to indicate the different types. The multiplicity of a node type specifies the number of other nodes (using a lower and upper bound) that can be connected to this node, using a given edge type. Depending on whether the multiplicity is defined at the end or source of the edge type, this defines the number of incoming or outgoing edges, respectively. If an indefinite number of connections is allowed, this is indicated using an asterisk (*).

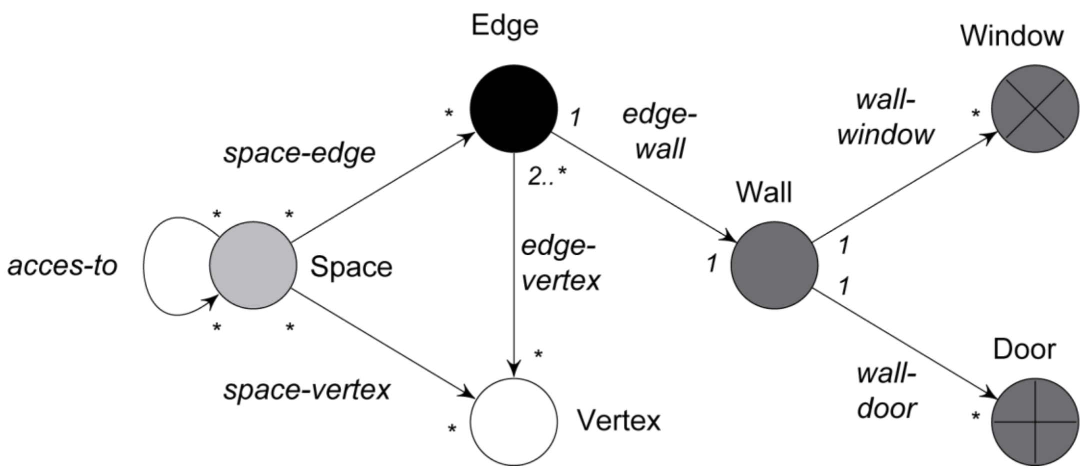


***Figure 1**: The type graph defines the node and edge types used for the grammar implementation. If an indefinite number of nodes or connections is allowed between node and edge types, this is indicated using an asterisk (*).*

For unimplemented shape grammars (developed on paper), architectural designs and objects (spaces, walls, doors, and windows) are all represented as shapes. The power of these shape grammars lies in the fact that shapes and their properties can be reinterpreted continuously (Stiny, 2006), allowing the emergence of features which are not apparent in the initial definition of a shape. For a good theoretical overview of emergence in shape grammars, we refer to the work of Knight (2003). Using a graph-based ontology to implement a grammar, shapes are now considered in terms of finite sets of entities, relations between these entities, and entity properties. One of the main advantages is that computer systems are now able to 'interpret' the visual information using the underlying graph representation. For many simple shape grammars, an ontology that contains only geometrical node types (vertex, edge) would be sufficient. Moreover, in the work of Grasl (2013) and Wortmann (2013) it is shown that when shapes are represented as graphs with geometrical nodes, the characteristic features of shapes (emergence and reinterpretation of shapes) can be maintained. In particular, GRAPE (Grasl, 2013) is a shape grammar implementation system in which shape emergence is supported by continuously translating graphs to shapes, and Wortmann (2013) describes several algorithms to translate simple two-dimensional shapes in the algebra *U12* to graphs. In other words, none of the essential features of shapes are lost when translating shapes to this kind of graphs. While (architectural) designs can be represented as (collections of) shapes, and thus be translated to graphs with only geometrical nodes, this would result in very large graphs, especially when a lot of semantic elements or details are involved.

In order to avoid overly large graphs, architectural design elements (space, wall, door, window) are treated as non-geometrical (symbolical) entities in the ontology shown in Figure 1. This separation of geometrical and non-geometrical data is well-established in Building Information Modelling (BIM) (Eastman et al., 2008). In this case, the "meaning" of designs or shapes becomes disambiguated, thereby omitting the freedom of interpretation that is typical for shape grammars. As Grasl (2012) correctly points out, for many shape grammars that focus on modeling an extensive, finite corpus of designs, emergence is not needed or could prove to be counterproductive. Both approaches (using only geometrical node types, and using also non-geometrical node types) may have merit in different design situations, which indicates the importance of letting the designer choose her own ontology.

3.2. Step 2: Constructing attributed part-relation graphs

The second step is to construct a graph representation of the shape grammar, based on the predefined type graph or ontology. These graphs can be constructed in several ways, some of which are summarized in the work of Wortmann (2013): maximal graphs, direct graphs, complete graphs, inverted graphs, and elaborate graphs. In the context of our proposed implementation approach, the use of elaborate graphs is the most appropriate, because all geometrical and non-geometrical entities can then be represented as the nodes of the graph, and their relations as the edges of the graph. In this paper, we will consistently use the term *part-relation graph* to refer to elaborate graphs, which is also the case in the work of Grasl & Economou (2013). Moreover, a part-relation graph can be attributed, which means that attributes are assigned to the nodes and edges of the graph, resulting in a so-called *attributed part-relation graph.* If such attributed part-relation graphs are used with only geometrical node types, they support "*the embedding and part relations and multiple intersections*" (Wortmann, 2013), however, they can also easily be extended with other kinds of node types (such as architectural or semantical entities). Depending on the ontology that is chosen beforehand, part-relation graphs can represent designs in a compact way (compared to for example, direct or maximal graphs). In order to construct attributed part-relation graphs, the following steps are needed.

First, the geometrical topology of the shape is to be determined. The main issue here is that shapes need to be represented in such a way that the pattern shape of rules should be detected as a (sub)shape in the given shapes. In order to do this, *maximal lines* are created, after which the intersections and endpoints of these maximal lines are calculated in order to obtain a complete representation. Maximal lines (Stiny, 1980) are lines created by combining all collinear line segments that touch or overlap. The use of maximal lines results in an unambiguous interpretation of the shape, in which lines do not consist of smaller line segments. These maximal line entities are represented as *edge* nodes in the graph. Also, the intersections and endpoints of the maximal lines are detected in the floor plan, and added as *vertex* nodes in the graph. The *edge-vertex* relations between the *edge* and *vertex* nodes are added to complete the geometrical topology of the graph. At this moment, the resulting graph represents the topology of the shape, but the shape is not limited to a specific geometrical realization. In this sense, the graph accounts for several parametric variations of the shape, which can be constrained by adding (geometrical) attributes to the nodes of the graph. The *vertex* nodes are attributed with coordinate geometry to constrain the graph to specific geometrical shapes. In particular, *vertex* nodes have "*x*" and "*y*" attributes, though this could be generalized for the three-dimensional case, for example in the work of Heisserman (1994) or Grasl (2013). To some extent, this approach allows the implementation of parametric shape grammars, because the graph can also be constrained to a set of geometrical variations instead of a single value. Figure 2 (a) shows an example of a simple two-dimensional shape and the part-relation graph constructed so far. This graph representation highly facilitates the computation of possible rule matches, because the two squares in the shape can be found using an identical graph search pattern (see Figure 2.b and 2.c). This example illustrates how shapes represented as part-relation graphs behave in a similar way as plain shapes, which is a result of the maximal line representation. By continuously translating shapes to graphs, and vice versa, the shape grammar implementation also supports the emergence of new shapes that arise, or are formed from the shapes generated by rule applications (Grasl & Economou, 2013).

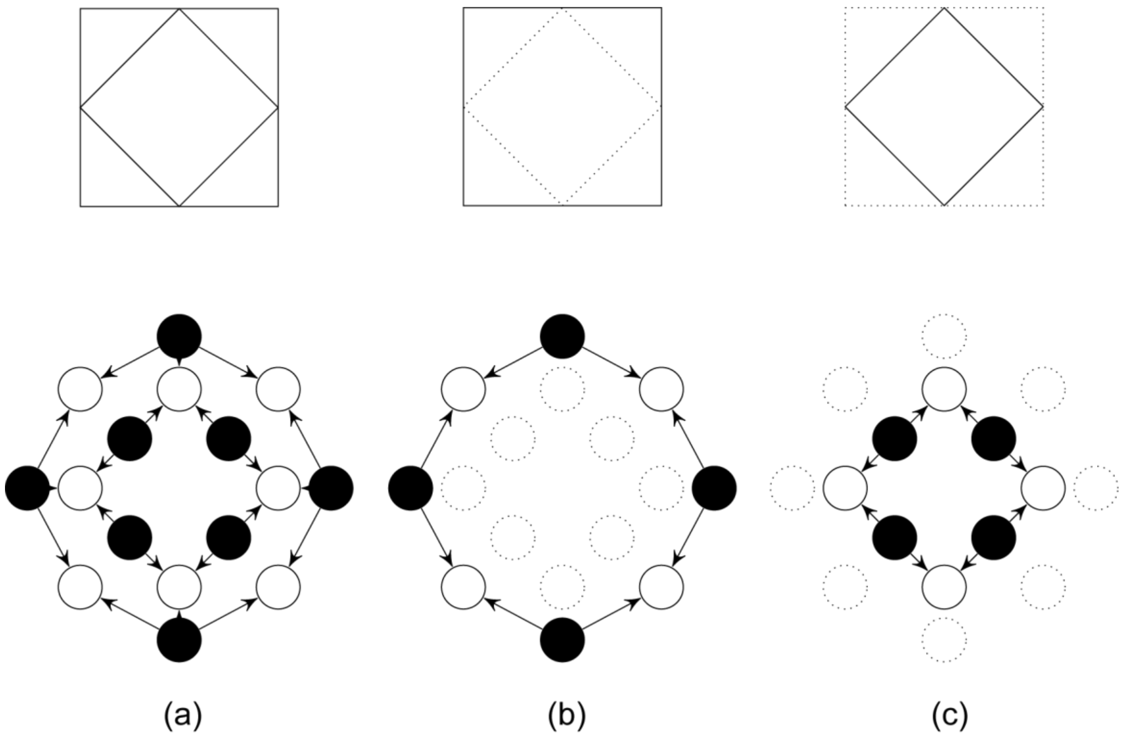


Figure 2*: (a) Example of a shape of one square embedded in another, and its corresponding part-relation graph with vertex nodes (white), edge nodes (black), and edge-vertex relations (arrow). Because the shape is represented using maximal lines, the two squares can be found using identical graph search patterns (b and c). The node attributes are not shown in this figure.*

Second, non-geometrical objects in the shape, if available, are to be determined, including walls, spaces, windows and doors. These objects are typically represented as shapes in hand-made drawings, and can easily be recognized by the human eye. This is not the case for computer implementations, and representing these entities using vertex and edge nodes in the graph would make the graph overly large and complex. Also, since the calculation time of rule matching and application in graph rewriting systems heavily depends on the number of graph objects (Strobbe et al., 2015), compact graph representations are preferable. Following the ontology described in Figure 1, non-geometrical objects are represented as *wall*, *space*, *window*, and *door* nodes in the graph representation. Also, the relations between the different nodes are identified and added to the graph representation, following the ontology. Finally, attributes are associated with the nodes for different purposes: to characterize material properties of wall objects, to include additional information about the function of spaces, or to describe geometrical properties of doors and windows. An example of a drawing of a floor plan and the corresponding attributed part-relation graph is shown in Figure 3. In the visual representation (Figure 3 left), a wall is drawn as a filled rectangular shape, which is a common way to draw walls in architectural floor plans. In the graph representation (Figure 3 right), wall entities are defined symbolically using wall nodes and their corresponding center edges (axis lines). Attributes are used to specify the function of the space "*f*" , to characterize element properties (thickness "*t*" and width "*w*"), and to constrain the graph to a specific geometrical realization ("*x*" and "*y*").

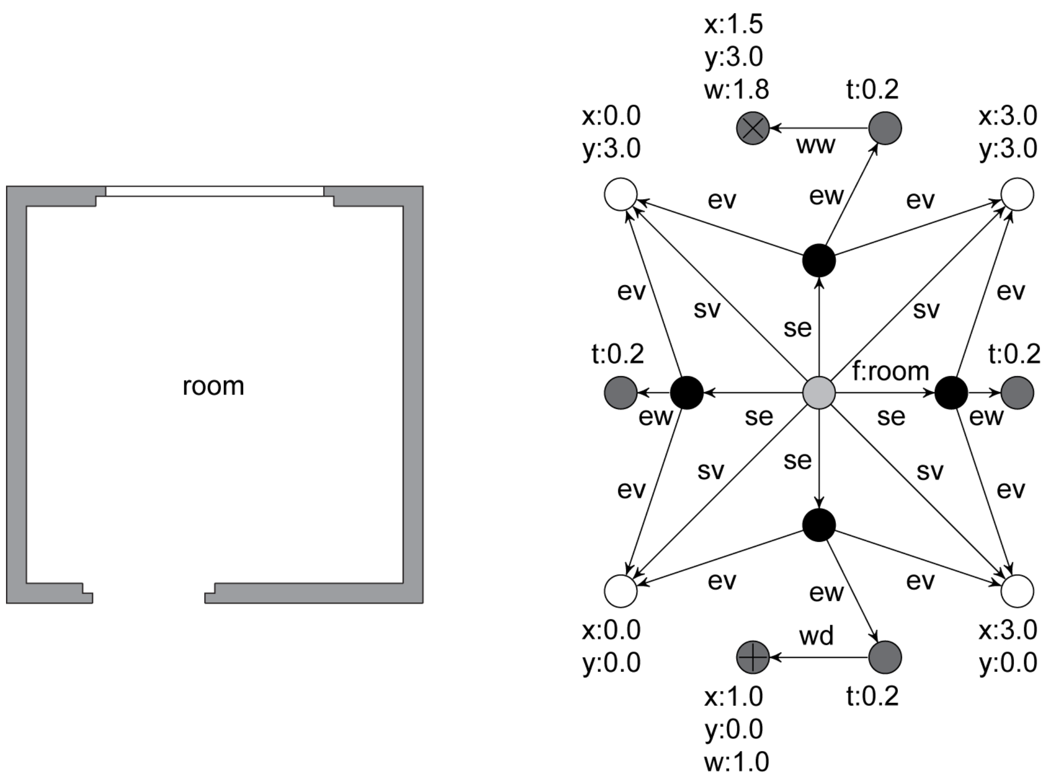


***Figure 3**: (left) Example of a floor plan. (right) Attributed part-relation graph with geometrical and non-geometrical nodes. The relations between the nodes are indicated by edges: edge-vertex (ev), space-edge (se), space-vertex (sv), edge-wall (ew), wall-door (wd), and wall-window (ww).*

3.3. Step 3: Adding conditional statements

In order to implement the grammar rules, both the left-hand side and the right-hand side of the shape part of the rules need to be described using the graph representation described in the previous section. The left-hand side of a rule describes the *pattern graph* that needs to be matched to a given graph representation of a dwelling. The right-hand side describes the *replacement graph* that will replace the matched part of the given graph. A grammar rule can include deleting or manipulating existing graph nodes, creating new graph nodes, and performing computations on the graph node attributes.

For graph grammar rules, additional rule application conditions are needed, for example to constrain the pattern graph to specific geometrical realizations, or to specify other conditional statements that are associated with shape grammar rules. In the domain of graph grammar theory, such application conditions can be defined using either *Attribute Conditions* (AC) or *Negative Application Conditions* (NAC) (Ehrig et al., 2006). Attribute conditions define restrictions on the attributes of graph objects. These ACs are defined as logical expressions using logical operators (including the equality operator, and the relational operator). Therefore, ACs can be used to describe conditional descriptive requirements, such as geometrical requirements (for example, area and proportion) and functional requirements. For example, considering a rule that detects a non-habitable space in a floor plan drawing (Figure 3), the pattern graph of this rule is associated with an AC "*f==nhs*" to constrain the matches found to spaces that have an attribute "*f*" equal to the value "*nhs*" (non-habitable space).

Negative application conditions specify requirements for non-existence of graph objects. While an AC is defined over attribute variables, NACs define conditions about the non-existence of graph nodes, edges, or even a specific subgraph. NACs do not have a direct equivalent in the shape grammar formalism, however, they are useful to guide and control rule application. NACs can be used to ensure that rules are applied only if specific graph objects are non-existent. For example, considering a rule that assigns a specific function to a space in the floor plan, only if this function has not yet been assigned to another space, a NAC can be used to ensure that no other spaces with this function exist.

3.4. Methodology

In order to evaluate the feasibility of the implementation approach described in the previous section, we have implemented part of the RdB transformation grammar, originally developed by Eloy (2012). The implementation is based on a JAVA development environment for graph rewriting, called AGG (http://user.cs.tu-berlin.de/~gragra/agg/). The existing editor in AGG is used to develop the grammar, and the available algorithms are used for automatic rule matching and rule application (Taentzer, 2004). We have built an interface on top of the underlying graph framework that shows a visual representation of the shape grammar derivation process. In other words, the graphs are used for the computer representation and computation of shapes, rules, and grammars, while a visual representation is shown to the designer. This corresponds to Tapia's characterization of a shape grammar interpreter: "the computer handles the bookkeeping tasks (…) and the designer specifies, explores, develops design languages, and selects alternatives." (Tapia, 1999). The focus of this paper is on the implementation of the RdB transformation grammar to a graph-theoretic grammar, and not so much on the interface of the presented tool. Nevertheless, several approaches exist for providing designers with visual and interactive functionality to develop and explore grammars (McKay et al., 2012; Strobbe et al., 2015). Automated shape grammar tools have several levels of automation, ranging from a stand-alone tool, in which the generation of a solution is totally controlled by the computer, to a lower level of automation where derivation and exploration is guided by the designer (Chase, 2010).

The proposed implementation approach is also embedded within a commercial CAD environment to make the shape grammar formalism more accessible to students and practitioners. In particular, shapes drawn in a common CAD format can be converted to attributed part-relation graphs. At the moment, it is possible to convert Industry Foundation Classes (IFC) files to graphs, which are described in an Extensible Markup Language (XML) format. IFC is an object-based data model that is intended to describe building and construction industry data. Following the approach described in section 3, geometrical (vertex, edge) and non-geometrical entities (space, wall, door and window) found in the IFC model are first added as nodes to the graph. More specifically, these entities correspond to *IfcCartesianPoint*, *IfcPolyline*, *IfcSpace*, *IfcWallStandardCase*, *IfcDoor* and *IfcWindow* in the IFC model, respectively. In the next step, the relations between the nodes are determined and connected by links. Subsequently, when the IFC model is imported to the shape grammar implementation tool, the properties of the IFC entities are read (for example, wall material properties, the width and height of doors and windows, and other properties), and they are added to the corresponding graph nodes. Figure 4 shows how the grammar implementation system is integrated within a wider CAD environment. A more elaborated user interface that supports enhanced exploration abilities is described in previous work (Strobbe et al., 2015). The output results of the graph transformation process are shown (visually) in the interface, allowing designers to automatically generate designs in the language of the grammar.

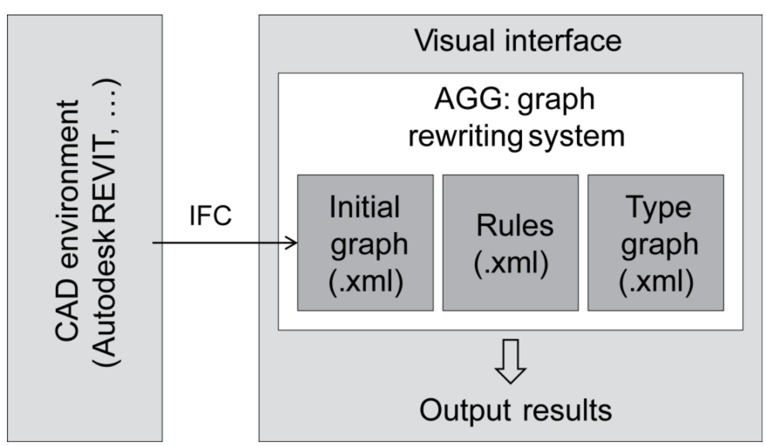


***Figure 4**: Integration of the shape grammar implementation system within a wider CAD environment. The output results of the graph transformation process are shown in the visual interface.*

4. Case-study: Rabo-de-Bacalhau transformation grammar

In this section, we describe the implementation of the RdB transformation grammar, developed on paper by Eloy (2012). The RdB transformation grammar provides an answer to the need for mass refurbishment of the existing housing stock in Lisbon (Portugal). In particular, a large part of the existing housing stock in Lisbon shows several constructional and functional problems, resulting in unsuitable housing in terms of contemporary comfort and accessibility standards. The RdB transformation grammar constitutes a formal methodology to generate alternative housing solutions that meet the current standards, depending on specific client needs and cost requirements. Moreover, the grammar includes various customized transformation strategies to adapt existing RdB houses to the current standards, depending on specific client needs. These transformation strategies describe how an existing dwelling is transformed to meet the standards and requirements in the form of transformation rules. In recent work, Eloy & Duarte (2014) describe the process undertaken to develop the RdB transformation grammar, and discuss how both the knowledge of the designer and knowledge acquired from other experiences of refurbishment are incorporated in the grammar. The implementation of this grammar to a computerized grammar can be seen as the next step in the development of a (semi)automated methodology to support mass housing refurbishment.

4.1. The original RdB transformation grammar

The original RdB transformation grammar uses a compound representation of the designs and the rules. For example, Figure 5 shows the compound representation of an existing RdB dwelling using five different representations, corresponding to five algebras *U12*, *U02.U12*, *U22*, *V02*, and *W02*. In particular, the algebra *U12* combines lines in a two-dimensional plane to represent floor plans of dwellings (Figure 5.a), the algebras *U02* and *U12* are used to represent topological relations between spaces of dwellings (Figure 5.b), and the algebra *U22* is used to represent spatial voids in floor plans of dwellings (Figure 5.c). Further, an algebra *V02* consists of labels and is used to control rule application or to associate non-geometrical information with shapes. In this case, labels are attributed to each space in a RdB dwelling (Figure 5.d), for example: habitable space (*hs*), non-habitable space (*nhs*), existing kitchen (*Xki*), and existing bathroom (*Xba*). An algebra *W02* consists of weights and is used to incorporate shape properties, for example to characterize construction systems for walls (Figure 5.e), including brick walls (dark gray), structural elements, side walls (black) and partition walls (light gray). As a result, a dwelling is described using five different representations in the RdB transformation grammar.

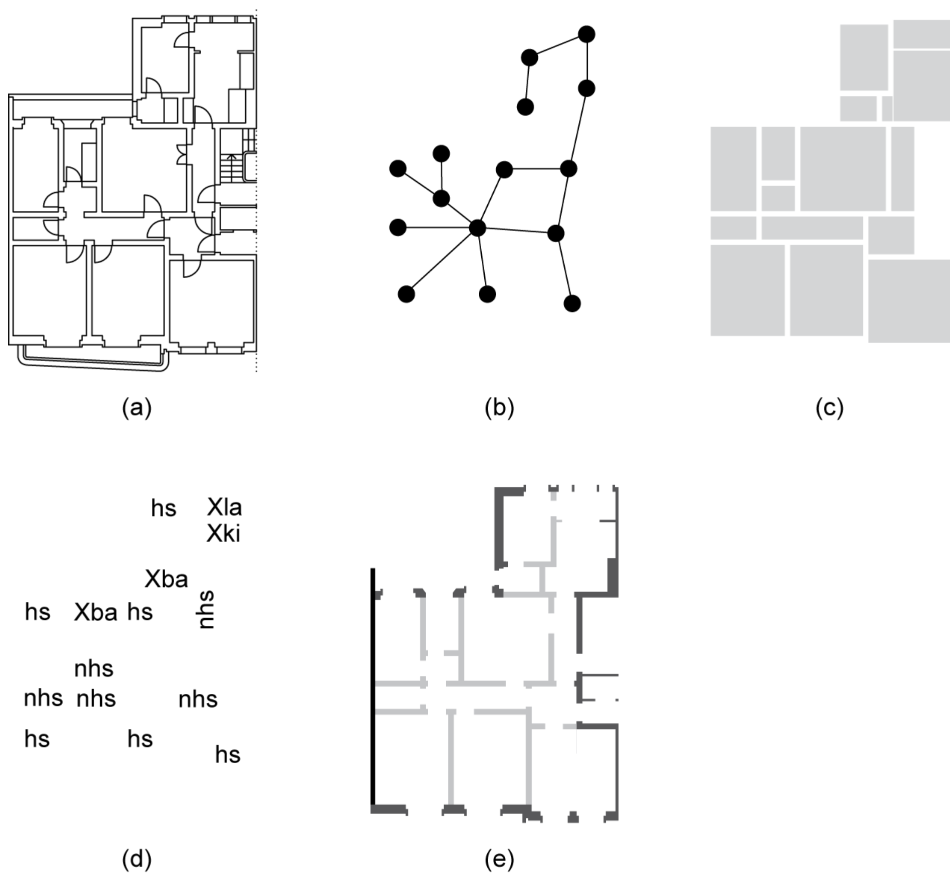


***Figure 5**: Compound representation of an existing RdB dwelling: (a) floor plan representation of the dwelling, (b) topological configuration of spaces in the dwelling (continuous lines for door connections and hidden lines for adjacency between rooms), (c) representation of spatial voids in the floor plan, (d) labels, and (e) weights. This image is reproduced from (Eloy, 2012).*

The rules of the RDB transformation grammar define the different transformation strategies that can be applied in order to meet the current standards and requirements. These rules are also defined using a compound representation. First, the rules consist of a shape part using two or more of the representations discussed in the beginning of this section. At least two representations are needed: for example, the graph representation and the labels are sufficient for rules that consider topological aspects only. However, in other cases, a combination of multiple or even all representations is needed to incorporate the desired design knowledge in the rules. Second, the rules consist of a conditional part to express additional rule application conditions considering dimensional or functional aspects of the shape. These application conditions provide a mechanism to control rule application towards specific limited cases. Third, a descriptive part is added to keep track of spaces required by the transformation strategy, spaces already assigned to the given dwelling, and spaces still available for assignment. In the original RdB transformation grammar, three sets are used to control the assignment of spaces: a set of existing spaces (*E*), a set of required spaces not yet assigned (*Z*), and a set of spaces already assigned to the proposed dwelling (*Z'*). In general, the descriptive part is defined as a transformation on a tuple of elements. Several example rules are shown further in this paper (Figure 8, Figure 11, and Figure 13), indicating the different rule parts. An extensive overview of the RdB transformation grammar rules is given in the work of Eloy (2012).

The RdB transformation grammar provides an interesting case study for implementation, because the grammar is extensive (142 shapes rules) to explore manually, and the implementation serves as the next step in the development of a (semi)automated approach for supporting mass housing refurbishment. Also, it provides an interesting case study to investigate the proposed implementation approach, because the grammar uses multiple representations (in different algebras), subshape detection, labels, and parametric rules. Another difficulty in implementing this grammar is how to implement the large number of conditional statements that are associated with the rules.

4.2. Translation of the original grammar to a computerized grammar

Following the approach described in section 3.1–3.3, the first step is the definition of the ontology. The RdB transformation grammar involves the transformation of existing dwellings to dwellings that meet the current standards and client needs. These dwellings are represented in multiple ways: a two-dimensional floor plan, a topology graph, a spatial void representation, and the representation of labels and weights. The goal is to find a type graph (ontology) for an attributed part-relation graph that can account for all these representations in one. The type graph shown in Figure 1 proves to be sufficient for this purpose. Indeed, the geometrical node types *vertex* and *edge* are used to represent the geometry of the floor plan, the node type *space* and the relation *access-to* are used to represent the topology graph and spatial voids, and the node types *wall*, *door*, and *window* are used to represent non-geometrical entities in the floor plan.

In order to construct the attributed topology graph, attributes are associated with the nodes to characterize construction systems for walls (brick walls, structural elements, side walls and partition walls), to include information about the functionality of spaces, and to describe geometrical properties of doors and windows. In particular, information about the construction system is added as an attribute "*s*" to the *wall* nodes in the graph. Also, labels "*cx*", "*cy*", "*wi*" and "*he*" describe the position, width and height, respectively, of doors and windows. In some cases, labels are used to add information not provided by shapes (such as the function of spaces and information about technical appliances (smoke detector, temperature detector). The label "*f*" describes the function of a space (e.g. habitable space "*hs*", non-habitable space "*nhs*"). In other cases, labels are used to control rule application or, in other words, to specify which rules can be applied at a specific moment in the transformation process. As a result, the floor plan is represented as an attributed part-relation graph that is at the same time compact and maintains sufficient semantic meaning.

Figure 6 shows the visual and graph representation of an existing RdB dwelling, described in the work of Eloy (2012). For illustrative purposes, the edge types are not shown, and only one node attribute is shown (function “*f*”). The resulting graph contains 113 nodes, 294 edges, and 126 attributes. The graph representation is used for the computation of shapes, rules, and grammars, while the visual representation is shown to the designer. This dwelling is one possible starting point for the transformation process using the RdB transformation grammar.

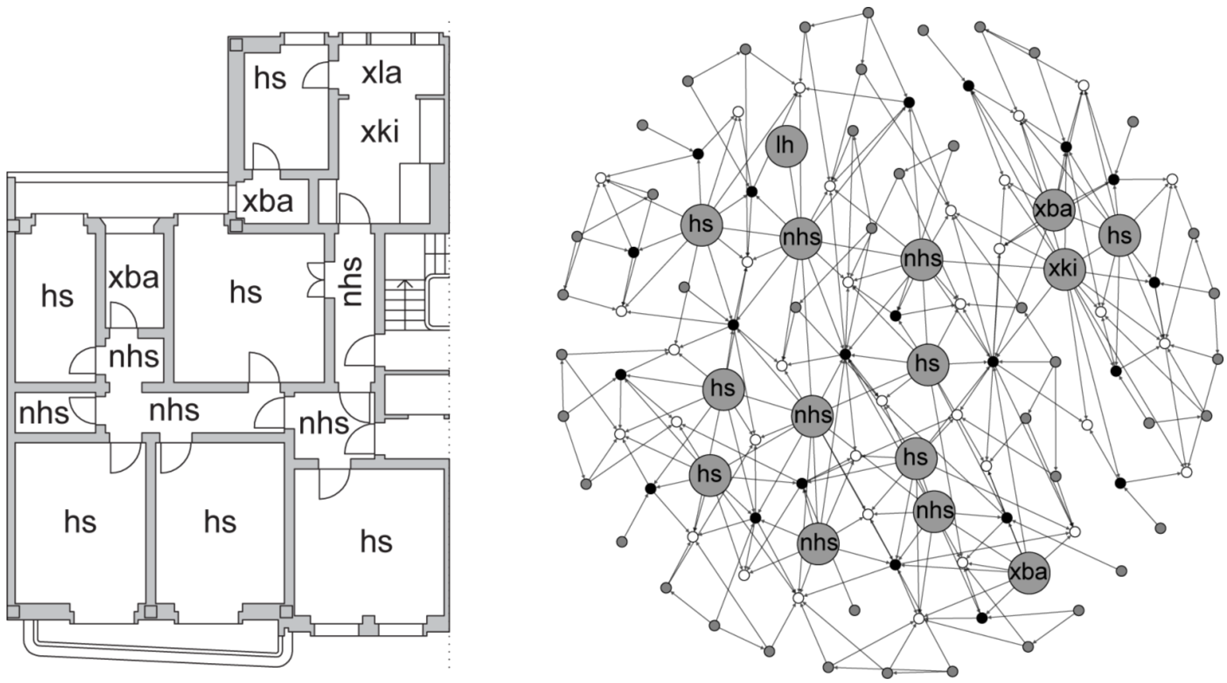


***Figure 6**: (left) Original floor plan from a RdB dwelling described in the work of Eloy (2012). (right) Attributed part-relation graph of the floorplan. For illustrative purposes, the edge types are not shown, and only one node attribute is shown (function “f”).*

For the RdB transformation grammar, there is no predefined initial shape. Instead, there are countless possibilities, because the initial shape can be the floor plan of any existing RdB dwelling. The development of these initial floor plan shapes (using a graph-based representation) is not part of the RdB transformation grammar, but they are usually drawn in traditional CAD environments. Figure 7 demonstrates the conversion from an initial RdB dwelling (modelled in Autodesk REVIT 2014) to the graph rewriting environment, using the IFC file format. Some details of the floor plan (e.g. balcony and constructional elements) are deliberately left out because they are less relevant in the scope of this experiment.

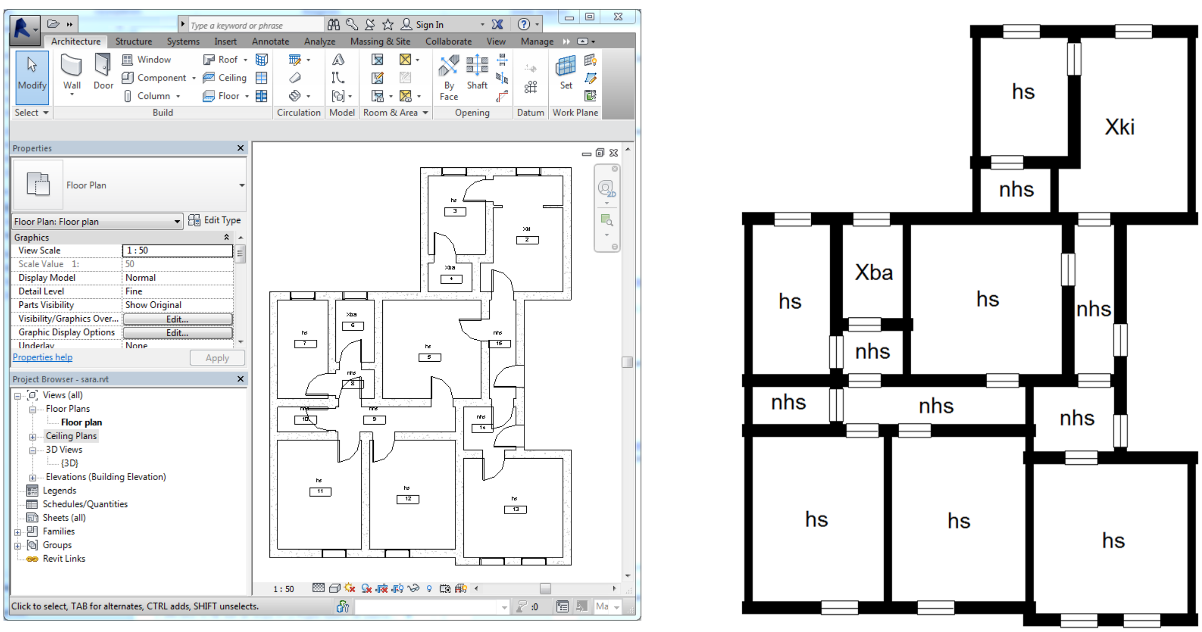


***Figure 7**: Model of a RdB dwelling in Autodesk REVIT 2014 (left) and visual representation of the dwelling in the graph rewriting environment (right).*

4.3. Three example rule types

In order to demonstrate the feasibility of the proposed approach, we discuss three relevant types of rules from the RdB transformation grammar: (1) assignment rules, (2) rules to connect spaces by eliminating walls, and (3) rules to divide spaces by adding walls (Eloy, 2012). For each rule type, an example rule from the original grammar is shown, together with the corresponding implemented rule.

Assignment rules

Assignment rules allow the required functions to be assigned to the existing spaces. An example assignment rule is shown in Figure 8. This rule transforms a non-habitable space (label "*nhs*") to a new hall space (label "*hl*") by modifying the label from the matched space, both in the floorplan representation and the topology representation. As mentioned in Section 4.1, each rule contains three parts: a shape part, a conditional part, and a descriptive part. The shape part of the rule consists of a parametric shape to create correspondence between the geometries of the different spaces within the dwellings studied (parameters *w*, *l*, *w1*, *w2*, *l1*, and *l2*). The conditional part of the rule defines dimensional conditions (size and area) on the one hand, and functional conditions on the other. In particular, a space can only be assigned as a hall space, if this space is connected to a lift hall (label "*lh*") and another non-habitable space. The descriptive part of the rule is described as an operation on a four-tuple with the following format: *<Dn: Fb, Ff; F; Z'; E>* → *<Dn: Fb, Ff; F1; Z' + {F1}; E – {F}, E + {F1}>*, where *Dn* denotes the stage in the derivation, *Fb* and *Ff* denote the back and front space, *F* denotes the function of the space involved, *Z'* denotes the set of spaces assigned to the proposed dwelling, and *E* denotes the set of existing spaces. The rule in Figure 8 removes only the non-habitable space that is under consideration in the rule (using a unique identifier) from the set of current spaces *E*, and adds the hall space to both the list of current spaces *E* and the list of already assigned spaces *Z'*. Please refer to (Eloy, 2012) for an elaborated discussion on the assignment rules of the RdB transformation grammar.

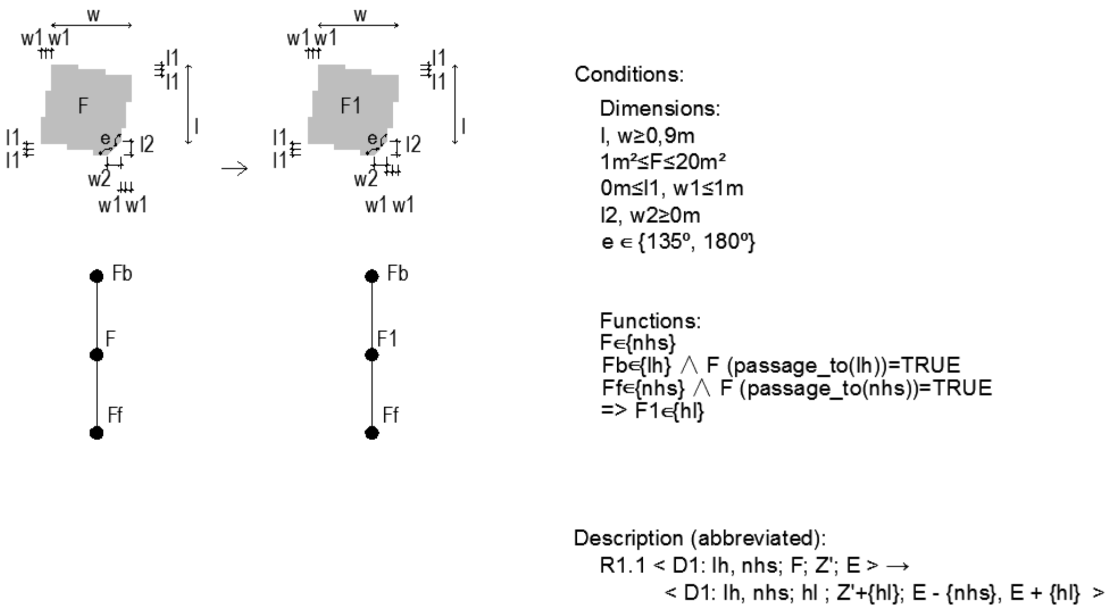


Figure 8*: Example rule from the RdB transformation grammar: assignment of hall. This image is adapted from Eloy (2012).*

The graph representation of this rule is given in Figure 9. The pattern graph of the rule consists of three *space* nodes with the functions *Fb*, *F* and *Ff*, together with the geometrical *vertex* and *edge* nodes of the middle space. In the original rule, the spatial void is drawn as a parametric shape in order to apply to all different geometries that can be found. In the implemented graph rule, the topology of all quadrilateral shapes (square, rectangle, parallelogram) with different dimensions is represented. As a result, the implemented graph rule is a parametric rule in the sense that all geometrical realizations of a quadrilateral shape can be matched. Since most spaces in the RdB dwellings are quadrilateral, such representation is sufficient in most cases. Nevertheless, the original rule also includes irregular spaces (*w1*, *w2*, *l1*, and *l2*), and therefore, a pattern graph should be implemented for each topology that can be found (pentagon, hexagon, and other polygons). In other words, the pattern shape of the original rule has multiple pattern graph equivalents. Several ACs are used to specify the conditional requirements of the original rule: the length *(Y4-Y1)>0.9m*, the width *(X4-X1)>0.9m*, the area *(Y4-Y1)*(X4-X1)<20m²*, and some functional conditions concerning the spaces *F*, *Fb*, and *Ff*. Also, a NAC is added to the replacement graph to ensure that a hall space has not already been assigned to the dwelling. In other words, NACs provide the functionality to keep track of the spaces already assigned, and spaces still available for assignment. The rule morphism specifies which graph objects of the pattern graph are preserved in the replacement graph, which is indicated in Figure 9 by showing identical numbers for each graph object in both rule sides. In this case, the replacement graph of the rule is nearly identical to the pattern graph (the rule does not change the topology), but the *f* attribute is changed to "*hl*" in order to assign the hall space in the dwelling.

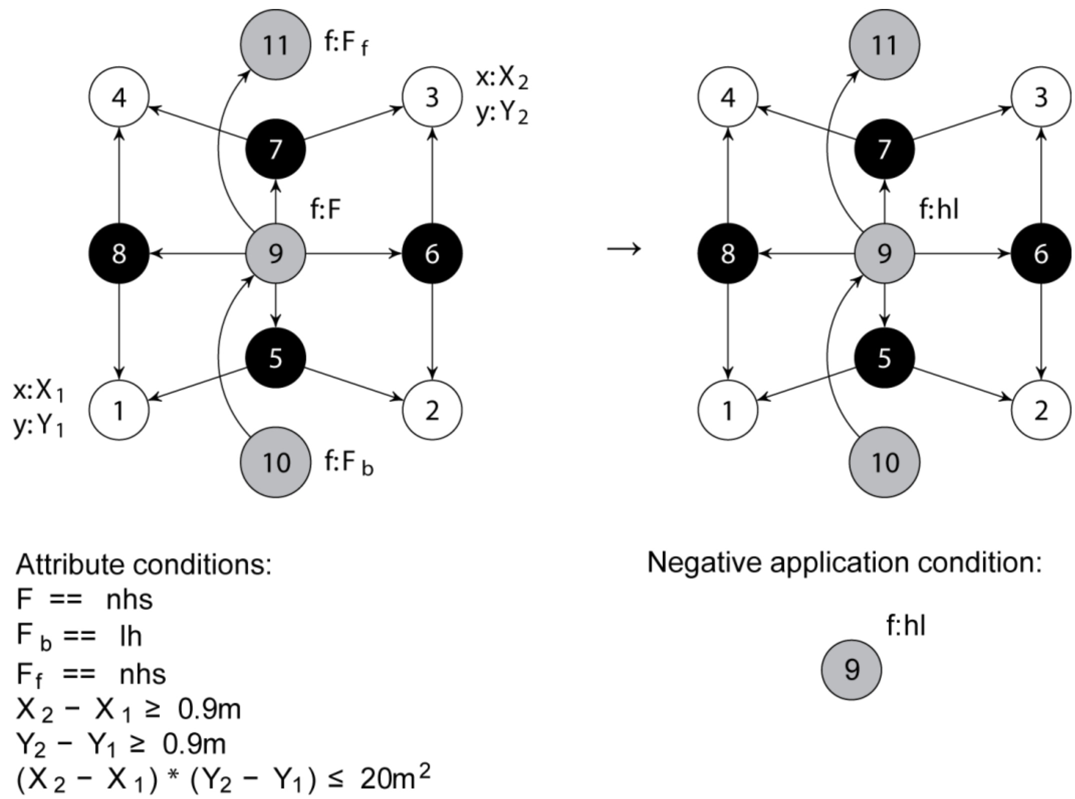


Figure 9*: Graph representation of the hall assignment rule, consisting of a pattern graph (left), a replacement graph (right), attribute conditions and negative application conditions (bottom). The numbers indicate the rule morphism between the pattern graph and the replacement graph.*

The implementation of the rule on a computer system demonstrates that the original rule is in fact under-constrained. In particular, the pattern graph of the original and implemented rule can be matched to two different sets in the dwelling, returning two identical results. Indeed, the *space* nodes with labels *Fb* and *F* are matched to the lift hall and the entrance adjacent to the lift hall, respectively, but the third *space* node with label *Ff* can be matched to two different non-habitable spaces in the dwelling (Figure 10). Therefore, the rule results in two distinct, but identical, rule application results. This behavior of the rule is difficult to foresee, because such forms of ambiguity in the rules often remain unnoticed. However, this is not the case for computer implementations of grammars, in which such form of ambiguity becomes directly noticeable. This is an example of how designers can learn from the computer implementation about the grammar itself.

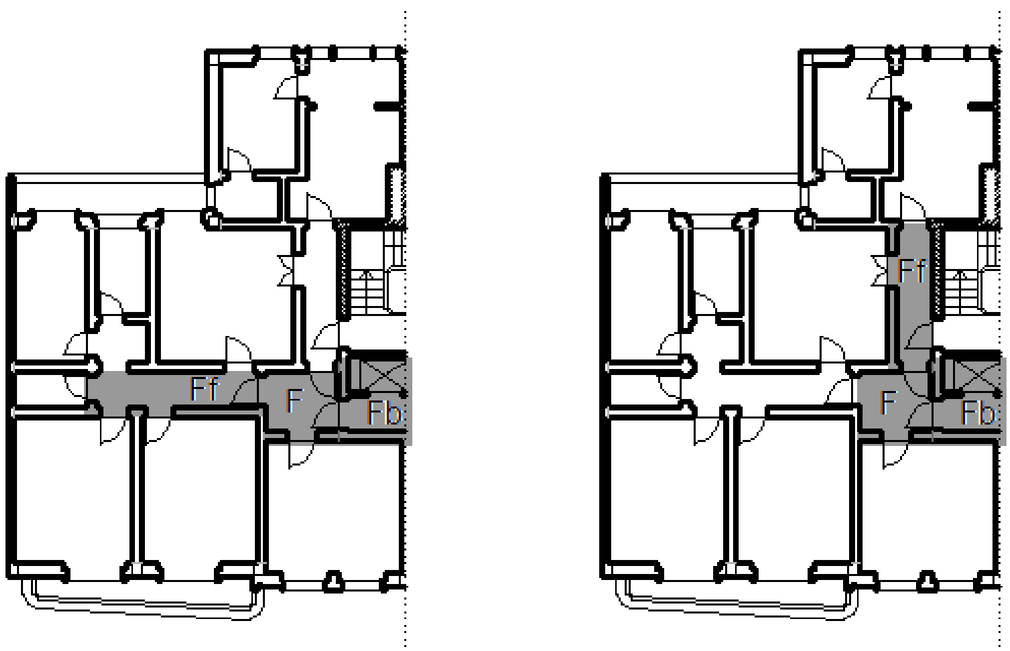
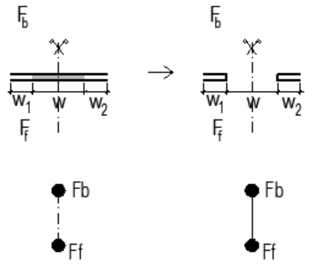


Figure 10*: Two possible rule applications of the assignment rule. The space nodes with labels Fb and F are matched to the lift hall and the entrance adjacent to the lift hall, respectively. The space node with label Ff can be matched to two different non-habitable spaces in the RdB dwelling.*

Connection rules

Connection rules connect spaces by eliminating parts of a straight wall , thereby connecting (or enlarging) spaces. An example rule to connect two adjacent spaces, if several conditions are satisfied, is shown in Figure 11. In this case, the representation of the rule is simplified for illustrative purposes: only the conditional requirements for private spaces are shown (bottom Figure 11), while requirements for other spaces are omitted. The conditional part of the rule describes that only specific adjacent spaces can be connected, for example single, double, or triple bedrooms (label values “*be.s*”, “*be.d*”, and “*be.t*”, respectively) with non-habitable spaces and corridors (label values “*nhs*” and “*co*”, respectively). The descriptive part of the rule is described as an operation on a tuple, having the following format: *<Dn: F1, F2; w * wcs(F1, F2)> → <Dn: F1, F2; w’ * wcs(F1,F2)>*, where *w* denotes the width of the wall, and w_{cs} denotes the wall construction system. In particular, a part (*w*) of the existing brick wall (*wub*) is demolished to allow for a door opening in the wall between two spaces (*w*Ø*). Please refer to (Eloy, 2012) for an elaborated discussion on the connection rules of the RdB transformation grammar.



Conditions:

Dimensions:
w1+w+w2≥1m
w∈{0.8m, 0.9m, 1m, 1.2m, 1.6m}
w1,w2≥0m

Function:

Private areas
Fb∈{be.d, be.t, be.s} ∧ Ff∈{nhs, co.p, co} ∧ Fb (passage_to(x)=FALSE, ∀ x∈{nhs, co.p, co}) ⇒ w∈{0.8m, 0.9m, 1m}
Fb∈{be.d, be.t, be.s} ∧ Ff∈{cl, ba.p} ∧ Ff (passage_to(x)=FALSE, ∀ x∈{Z, hs, nhs}) ⇒ w∈{0.8m, 0.9m, 1m}
Fb∈{ba.p, cl} ∧ Ff∈{nhs, co.p, co} ∧ Fb (passage_to(x)=FALSE, ∀ x∈{Z, hs, nhs}) ⇒ w∈{0.8m, 0.9m, 1m}

Description (abbreviated):
R7.1f < D7: Fb, Ff; w*w*ub*(Fb, Ff) > →
< D7: Fb, Ff ; w*Ø >

Figure 11*: Example rule from the RdB transformation grammar: connecting two adjacent spaces by eliminating part of a straight wall. Only the conditions for private spaces are shown here. This image is adapted from Eloy (2012).*

The graph representation of this rule is given in Figure 12. The pattern graph consists of two *space* nodes, together with nodes representing their shared edge, wall and vertex entities. The geometrical and functional requirements from the original rule are implemented as ACs. These attribute conditions are described in a similar format as the original rule, using logical conjunctions and disjunctions to express the different cases when the rule can be applied. Again, only the ACs for the connection of private spaces are shown in Figure 12, for illustrative purposes only. Also, a NAC is added to the replacement graph to ensure that the two spaces are not yet connected. The replacement graph adds an additional *door* node to the matched graph (width 0.8m, 0.9m, 1m, 1.2m, or 1.6m) and an *access-to* relationship between the two *space* nodes.

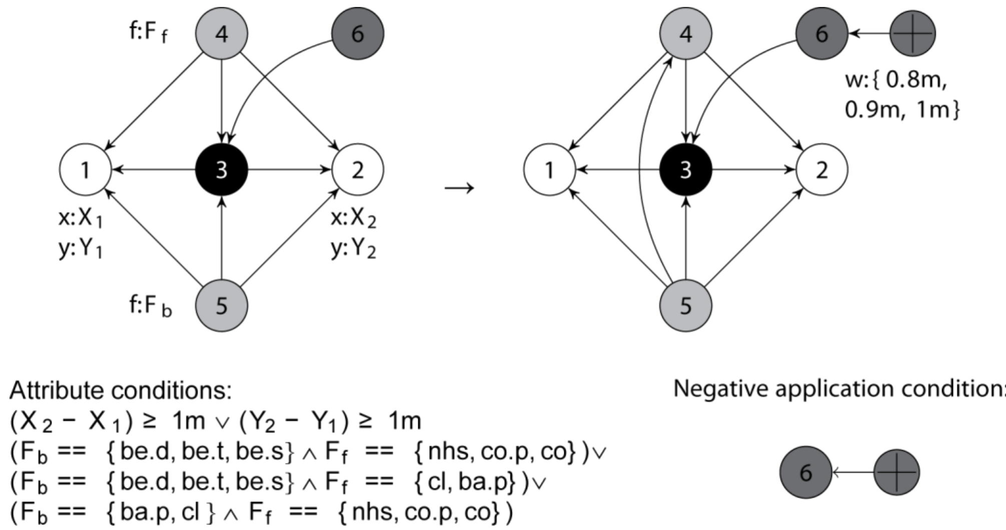


Figure 12*: Graph representation of the adjacent space connecting rule, consisting of a pattern graph (left), a replacement graph (right), attribute conditions and negative application conditions (bottom).*

Depending on the given conditional requirements, this rule can be applied to a given dwelling in many ways. Using the original grammar rule, developed on paper, it is difficult to detect all the possibilities where the rule can or cannot be applied, because of the large number of conditions that need to be considered. According to previous research by Woodbury & Burrow (2006), herein lies one of the main benefits of computer aided design tools: their ability to support designers in exploring large design spaces. A computer system can enumerate all possible rule applications automatically. In this way, designers benefit from computer implementations, because they can focus on selecting and exploring alternatives, leaving the underlying rule application and calculation tasks for the computer.

Division rules

Division rules divide a space by adding a wall between two parts of the space. An example rule to divide a bathroom by adding a wall, if several conditions are satisfied, is shown in Figure 13. The shape part of the rule describes how a new wall is added perpendicular to an existing wall of a space that has been assigned as a private bathroom (label *Fb* has the value "*ba.p*"). Several conditional requirements are defined, for example to ensure that the area of the spaces meet a predefined comfort level: a bathroom larger than 3.5m² for a minimum level of comfort, and a bathroom larger than 5m² for a recommended level of comfort. The descriptive part of the rule is described as an operation on a tuple with the following format: *<Dn: F1; E> → <Dn: F1, F2; E + {F2}, w * wcs(F1,F2)>*. In particular, a light partition wall (*wul*) is added with a specific width (*w*) to divide the space. Please refer to (Eloy, 2012) for an elaborated discussion on the division rules of the RdB transformation grammar.

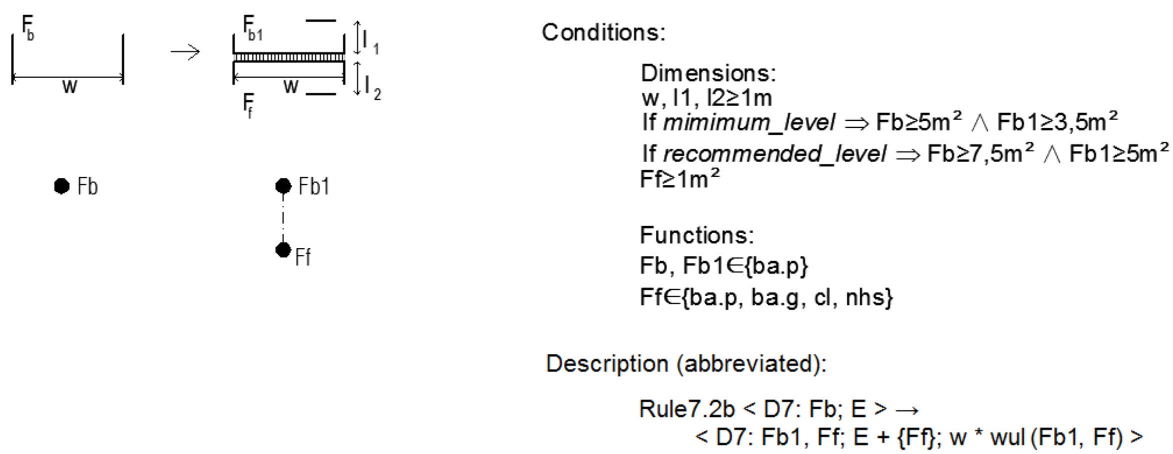
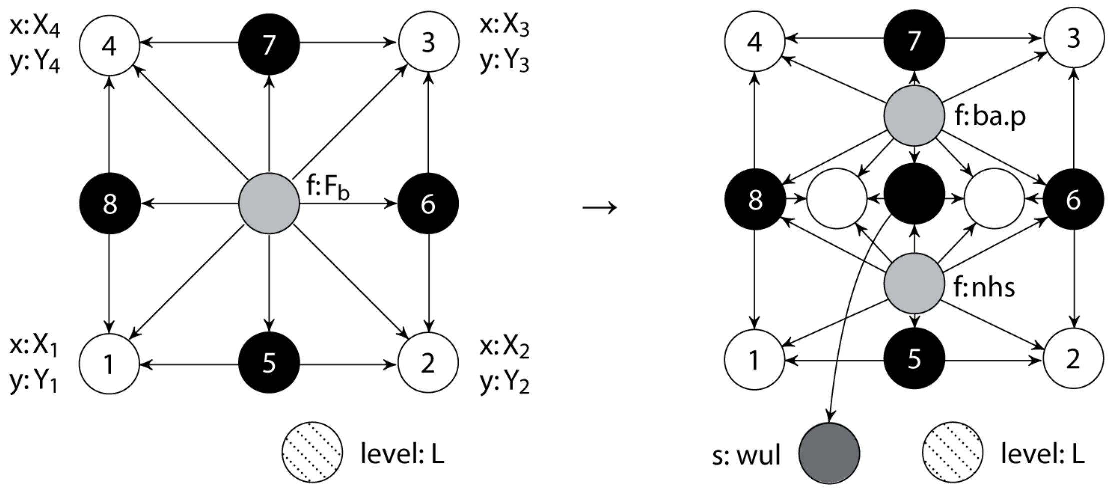


Figure 13*: Example rule from the RdB transformation grammar: dividing a private bathroom space by adding a wall. This image is adapted from Eloy (2012).*

The graph representation of this rule is given in Figure 14. The pattern graph of the rule consists of one *space* node with a label *Fb*, and eight other nodes that represent four boundary edges and four vertices of the space. The information of the edges and vertices is needed to calculate the area and length of the space that are dependent on the shape. Similarly to the assignment rule in Figure 9, this rule is parametric in the sense that all geometrical realizations of a quadrilateral shape can be matched. For irregular shapes, different pattern graphs should be implemented, and therefore, the original rule has multiple graph rule equivalents. In order to create a perpendicular wall, the two opposing edges need to be parallel, which is achieved by using an AC to ensure that the slope of the two edges is identical: $(X3 - X1)/(Y3 - Y1) == (X4 - X2)/(Y4 - Y2)$. The other geometrical requirements (*w*, *l1*, *l2*) are specified to ensure that spaces have sufficiently large edge dimensions: $(Y3 - Y1) > 2m$, $(Y4 - Y2) > 2m$, $(X2 - X1) > 1m$. In order to implement the conditional requirement of the predefined comfort levels (minimum or recommended), it is necessary to extend the type graph in Figure 1 with an additional node type "c*omfort level*" that has an attribute *level*. Depending on the value of this *level* attribute *L*, a specific AC is applicable: $[L == \text{“minimum”} \wedge (X2 - X1) * (Y3 - Y1) > 5m^2] \vee [L == \text{“recommended”} \wedge (X2 - X1) * (Y3 - Y1) > 7.5m^2]$. The replacement graph adds a light partition wall (label *s* has value "*wul*") that divides the space in two new spaces (private bathroom and non-habitable space). Two new *vertex* nodes are created that are incident to the two existing parallel edges. Next, a new *edge* node is created between the two vertices, together with a new *wall* node. The values of the attributes of the new *vertex* nodes depend on the area of the two new spaces and the specified comfort level (minimum or recommended).



Attribute conditions:
$(F_b == ba.p)$
$(Y_3 - Y_1) > 2m$
$(Y_4 - Y_2) > 2m$
$(X_2 - X_1) > 1m$
$(X_3 - X_1) / (Y_3 - Y_1) == (X_4 - X_2) / (Y_4 - Y_2)$
$(L ==$ “minimum” $\wedge (X_2 - X_1) * (Y_3 - Y_1) > 5m^2) \vee$
$(L ==$ “recommended” $\wedge (X_2 - X_1) * (Y_3 - Y_1) > 7.5m^2)$

Figure 14: *Graph representation of the division rule, consisting of a pattern graph (left), a replacement graph (right), and attribute conditions (bottom).*

Finally, the RdB transformation grammar includes several other types of rules, which are not discussed in this paper. For example, one rule type in the grammar is used to permute functions between spaces. This rule type is very similar to the assignment rule type, and can thus be implemented using the same approach. Another example is a rule for changing the derivation stage in the transformation process. In this case, labels are used to include information about the derivation stage and to control the derivation process. Lastly, the RdB transformation grammar consists of rules for integrating information, communications and automation technologies (ICAT) in the dwellings. These rules use labels to incorporate information about the ICAT in the dwellings. Therefore, they can be implemented as graph rules that modify the attributes of the nodes that correspond with the locations of the ICAT.

4.4. Results of the implementation

A part of the RdB transformation rules has been implemented using the proposed methodology described in section 3.4. In particular, we have implemented rules of the rule types described in Section 4.3: (1) assignment rules, (2) rules to connect spaces by eliminating walls, and (3) rules to divide spaces by adding walls. An example of a derivation for a possible RdB dwelling is shown in Figure 15. The shape rules used at each step of the derivation are shown between the intermediate derivations. For illustrative purposes, several derivation steps are merged (as indicated in Figure 15), because the changes to the floor plan are subtle. The labels that have been used for the derivation process are described in Table 1. The rules that have been used are shown in Table 2. For each rule, a short description is given, together with a reference to the original rule in the RdB transformation grammar (Eloy, 2012).

Label	Description	Label	Description
nhs	Non-habitable space	be	Bedroom
hs	Habitable space	be.s	Single bedroom
xba	Existing bathroom	be.d	Double bedroom
xki	Existing kitchen	ki	Kitchen
xla	Existing laundry	li	Living room
co	Corridor	ba.p	Private bathroom
co.p	Private corridor	ba.g	Guest bathroom
la	Laundry	lh	Lift hall
hl	Hall	di	Dining room
ba	Bathroom	ho	Home office

***Table 1**: The labels used for the representation of housing designs.*

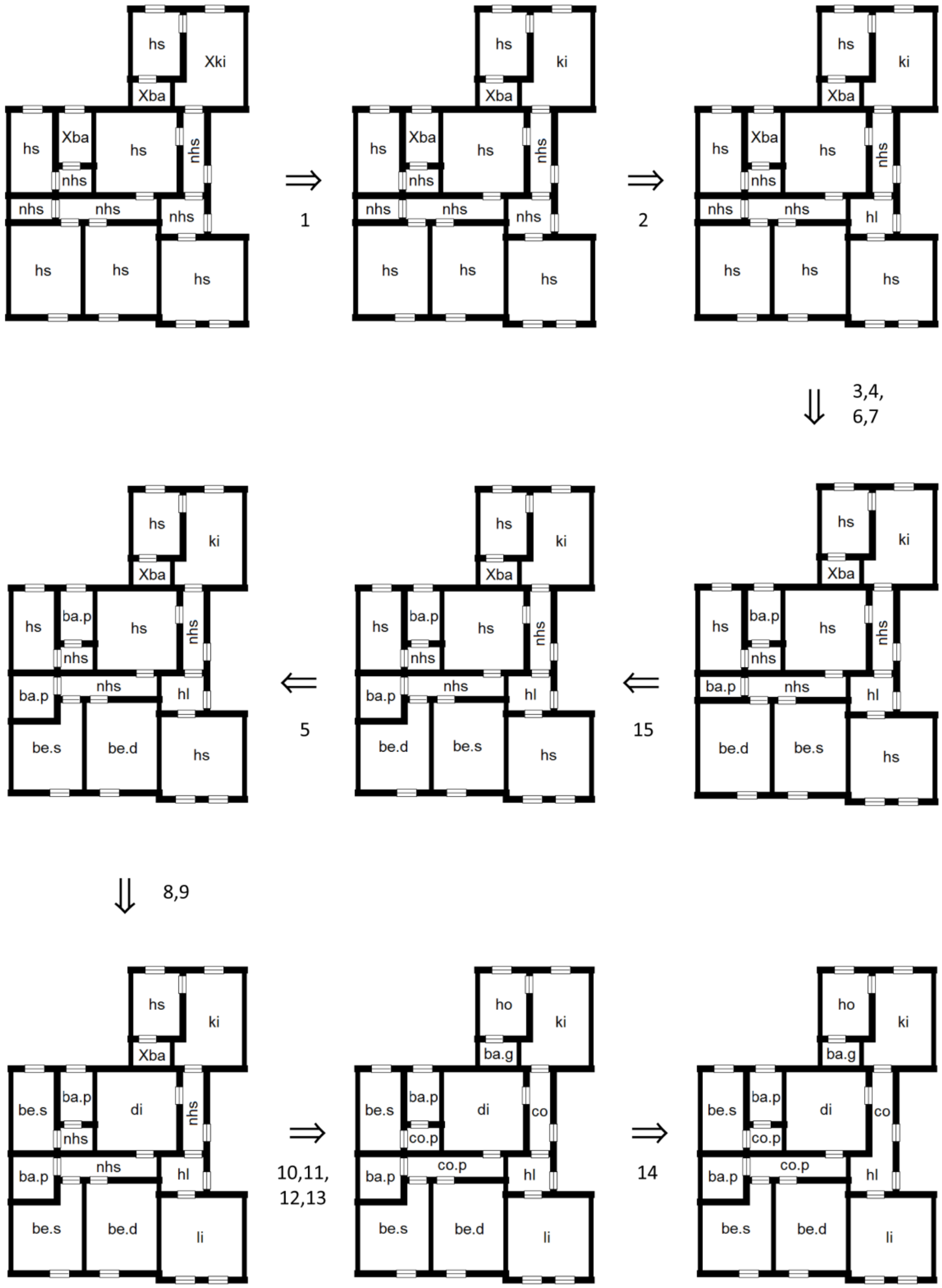


Figure 15*: Example of a derivation of a possible RdB dwelling. The shape rule used at each step of the derivation is shown between the intermediary steps. For illustrative purposes, several derivation steps are merged.*

Rule	Description	Reference to original rule
Rule 1	Assignment of isolated kitchen	(Rule 0.1)
Rule 2	Assignment of hall	(Rule 1.1)
Rule 3	Assignment of double bedroom	(Rule 2.1b)
Rule 4	Assignment of single bedroom	(Rule 2.3b)
Rule 5	Permuting bedroom assignment due to area criteria	(Rule 2.5)
Rule 6	Assignment of main private bathroom	(Rule 2.6)
Rule 7	Assignment of second private bathroom	(Rule 2.8b)
Rule 8	Assignment of living room	(Rule 3.1a)
Rule 9	Assignment of dining room	(Rule 3.2b)
Rule 10	Assignment of isolated home office	(Rule 3.4)
Rule 11	Assignment of guest bathroom	(Rule 3.11)
Rule 12	Assignment of private corridors	(Rule 4.1)
Rule 13	Assignment of corridors	(Rule 4.2)
Rule 14	Widening the connection between two rooms (by eliminating walls on both sides of a door opening)	(Rule 7.1.i)
Rule 15	Changing room dimension by moving a wall	(Rule 7.4b)

***Table 2**: Transformation rules used during the derivation process. For each rule, a short description is given, together with a reference to the original rule in Eloy (2012).*

5. Discussion

In this section, we describe several findings and issues encountered during the implementation and evaluation of the proposed approach and the RdB transformation grammar case study. First, while human designers are able to reinterpret shapes and shape rules during a shape grammar derivation process, computer systems are able to “interpret” the visual information in terms of the ontology used, or in this case, using the underlying graph representation. Therefore, emergence of shape features or properties that are not apparent in the initial definition of the shapes is not readily supported, but we describe how computer implemented grammars can support emergence for shapes in the algebra in *U12* in Section 3.1. The main benefits of the approach set out in this paper are gained for extensive and complex grammars that are difficult to explore manually, because computer systems enable easier rule application. The proposed approach in this paper has been evaluated for the RdB transformation grammar, which is a complex shape grammar with multiple representations (in different algebras), subshape detection, labels, and parametric rules. The computer implementation of this grammar is a good way to quickly generate several outcomes (in the context of the mass housing program).

Second, while several objects, such as spaces, walls, doors and windows, can be considered as geometrical entities from an architectural point of view, they are treated as non-geometrically in the implemented grammar. The representation of such objects using only geometrical graph nodes (such as vertex and edge) would result in complex graph representations. Among the main benefits of this approach is that the initial shape and the pattern and replacement shape of the rules can be represented quite intuitively, and with as little graph objects as possible. As rule matching is the most runtime intensive step in the graph rewriting process, it is important to keep the number of graph objects low. Moreover, the calculation time of rule matching depends on the size of the initial graph and the pattern graph. In this paper, we have used an ontology that mixes geometrical with non-geometrical node types.

In order to validate this approach, we have generated four more derivation examples (Figure 16), starting from the same initial design of Figure 15. These results show some variation in the placement of the dining room (either north or south, connected to the living room), the placement of the private bathroom, and the placement of the corridors. The entire derivation shown in Figure 15 and the four resulting designs in Figure 16 can be generated in approximately 1 – 2 seconds using a common personal computer (in this experiment, an Intel Core 2 Quad @3.00GHz processor with 4GB RAM and Windows 7 (64-bit) is used). This experiment demonstrates how the computer implemented grammar might support a designer in exploring design alternatives. Even more alternatives could be generated by taking into account the parametric variables in the rules. While an extensive benchmarking of computer implemented shape grammars would be very interesting, it is out of scope in this paper, and therefore it is part of our current ongoing and future research. More details on performance measurements of grammar implementations can also be found in earlier research work (Grasl & Economou, 2013; Strobbe et al., 2015).

Third, the implementation of the RdB transformation grammar has shown some unexpected rule application results, largely due to the original rules being under-constrained or ambiguous. Human designers can easily make meaning from visual patterns in the rules and, as a result, such forms of ambiguity in the rules often remain unnoticed. This is not the case for computer implementations of grammars, in which such form of ambiguity becomes directly noticeable. As a result, the attempt to implement a grammar on a computer system leads to a deeper understanding of that grammar, and might result in the further development of the grammar. In this way, designers might learn from the computer implementation of their shape grammar about the grammar itself.

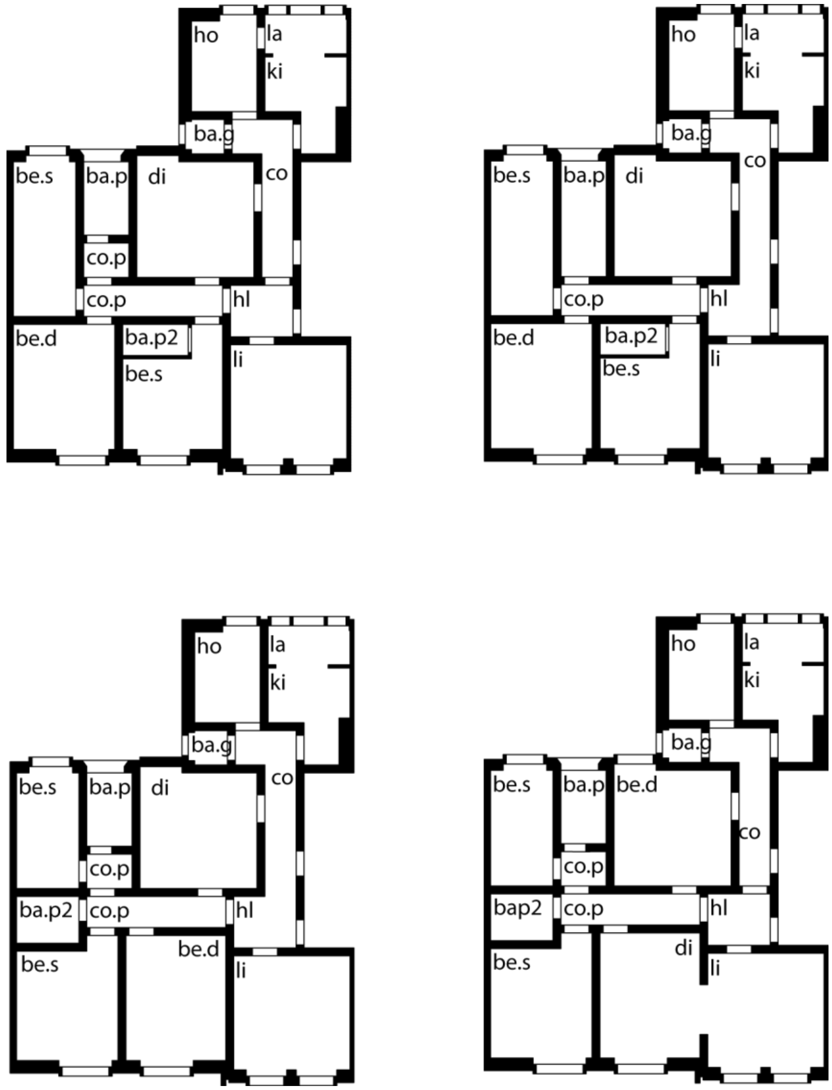


Figure 16*: Four alternative automated productions of other RdB dwellings, based on the same initial design of Figure 15. The resulting designs show some variation in the placement of the dining room, the private bathroom, and the corridors.*

Fourth, as a result of the structured (graph-based) representation of computerized grammars, rewriting systems might be used to enumerate all possible rule applications automatically. Using the original grammar, developed on paper, it is often difficult to detect all the possibilities where rules can be applied, because of the large number of conditions that need to be taken into account. Therefore, a computer system might be used to handle automatic rule application and to handle the management of possible design alternatives. In this case, the designer can focus on selecting and exploring these alternatives. This results in a mixed human-computer interaction, in which the computer supports the designer in exploring the language of a grammar, or a design space in the more general sense. This is achieved by using a visual interface (that has been built on top of the underlying graph rewriting framework), which shows all the possible design alternatives that can be selected at a certain point in the derivation process. As a result, the designer can navigate in the design space of the implemented grammar, which is an important amplification strategy for computers to support human design space exploration. In previous research work (Strobbe et al., 2015), a general framework for design space exploration using shape grammars is presented. For the RdB transformation grammar, such a human-machine interaction is beneficial, because of the large number of conditional statements in the rules, which makes it difficult to detect manually where the rules can be applied.

Finally, the proposed implementation approach is evaluated through a case study of the RdB transformation grammar, but the proposed approach and findings should be generalizable to other specific shape grammars. For example, the graph-theoretic representation of the Palladian grammar, described in the work of Grasl (2012), can be defined using an ontology that contains node types for spaces, rooms, porticos, center rooms, exterior, and orientation (structured in a hierarchical manner), and edge types for describing east-west and north-south relations. In general, the translation of an existing shape grammar to an equivalent graph-theoretic grammar is an interesting exercise, because designers need to think about several aspects of their grammar in a different way: for example, the use of multiple graph equivalents of a parametric shape, the definition of the ontology, or the definition of conditional requirements to control rule application. As a result, the graph-theoretic equivalent of a shape grammar might operate using different underlying principles, leading to an alternative understanding of the grammar at hand, which is complementary to designing shape grammars.

6. Conclusion

The work presented in this paper demonstrates an approach for a graph-theoretic implementation of a shape grammar, originally developed on paper, on a computer system. The issue of the computer implementation of shape grammars is shown to be important, because the computer implementation of shape grammars concerned with modeling an existing corpus work enables (semi)automatic rule application, but also influences the design of the shape grammar itself. A practical step-by-step approach is given for the translation of a shape grammar to an equivalent graph-theoretic grammar. The RdB transformation grammar, originally developed by Eloy (2012), is used to demonstrate the details of this approach and to evaluate the feasibility. In particular, three relevant types of rules used in the RdB transformation grammar are discussed: assignment rules, rules to connect spaces by eliminating walls, and rules to divide spaces by adding walls. In order to evaluate the feasibility of the implementation approach, a part of the RdB transformation grammar is implemented, using a JAVA development environment for graph rewriting. This implementation is shown to be both feasible and valuable in several aspects. First, the proposed approach contributes to the existing state of the art on the graph-theoretic representation of shape grammars. It is shown how the implementation of a shape grammar to a computerized grammar might influence the design of the original shape grammar. Second, the work presented in this paper can be considered as an example of how shape grammars are implemented to a computer system, which might in the turn increase the impact of grammars on design practice. In particular, the development of a (semi)automated methodology to support mass housing refurbishment is described. Finally, the proposed approach is embedded within a commercial CAD environment to make the shape grammar formalism more accessible to students and practitioners. Some future lines of research include the further integration of the proposed approach within a CAD environment in order to allow the results to be returned to the CAD environment, and an extensive benchmarking of the proposed implementation approach and other shape grammar implementations in general.

7. Acknowledgments

The graph grammars are implemented using AGG, a JAVA development environment for attributed graph transformation. The research is funded by the Agency for Innovation by Science and Technology in Flanders (IWT).

8. References

- Aksamija, A., K. Yue, H. Kim, F. Grobler & R. Krishnamurti (2010). Integration of knowledge-based and generative systems for building characterization and prediction, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 24, 3–16.
- Chase, S. (2002). A model for user interaction in grammar-based design systems, Automation in Construction 11(2), 161–172.
- Chase, S. (2010). Shape grammar implementations: the last 35 years, in: 4th International Conference on Design Computing and Cognition.
- Correia, R., J. P. Duarte, A. Leitao (2010). MALAG: a discursive grammar interpreter for the online generation of mass customized housing, in: 4th International Conference on Design Computing and Cognition.
- Duarte, J. (2005). A discursive grammar for customizing mass housing: the case of Siza's houses at Malagueira, Automation in Construction 14, 265–275.
- Eastman, C., Teicholz, P., Sacks, R., Liston, K. (2008). BIM Handbook: a guide to building information modelling for owners, managers, architects, engineers, contractors, and fabricators. New York: Wiley.
- Ehrig H., K. Ehrig, U. Prange, & G. Taentzer (2006). Fundamentals of Algebraic Graph Transformation. Springer.

- Eloy, S. & J. Duarte (2014). Inferring a shape grammar: translating designers knowledge, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 28, 153–168.
- Eloy, S. (2012). A transformation grammar-based methodology for housing rehabilitation: meeting contemporary functional and ICT requirements, Phd, TU Lisbon.
- Ertelt, C. & K. Shea (2010). Shape grammar implementation for machine planning, in: 4th International Conference on Design Computing and Cognition.
- Fitzhorn, P. (1990). Formal graph languages of shape, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 4 (3), 151–163.
- Flemming, U. (1987). More than the sum of parts: the grammar of Queen Anne houses, Environment and Planning B: Planning and Design 14, 323–350.
- Geiβ, R., G. V. Batz, D. Grund, S. Hack & A. Szalkowski (2006). GrGen: A fast SPO-based graph rewriting tool, vol. 4178, Springer Berlin Heidelberg, 383–397.
- Gips, J. (1999). Computer implementation of shape grammars, in: NSF/MIT Workshop on Shape Computation, vol. 55, 1–11.
- Granadeiro, V., J. Duarte, J. Correia & V. Leal (2013). Building envelope shape design in early stages of the design process: Integrating architectural design systems and energy simulation, Automation in Construction 32, 196–209.
- Grasl, T. & A. Economou (2013). From topologies to shapes: parametric shape grammars implemented by graphs, Environment and Planning B: Planning and Design 40 (5), 905–922.
- Grasl, T. (2012). Transformational Palladians, Environment and Planning B: Planning and Design 39 (1), 83–95.
- Grasl, T. (2013). On shapes and topologies: graph theoretic representations of shapes and shape computations, Phd, TU Vienna.

- Heisserman, J. (1994). Generative geometric design, IEEE Computer Graphics and Applications, 37–45.
- Helms, B. & K. Shea (2012). Computational synthesis of product architectures based on object-oriented graph grammars, Journal of Mechanical Design 134(2), 1–14.
- Hoisl, F. & K. Shea (2011). An interactive, visual approach to developing and applying parametric 3-D spatial grammars, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 333–356.
- Jowers, J. & C. Earl (2011). Implementation of curved shape grammars, Environment and Planning B: Planning and Design 38, 616–635.
- Jowers, I., D.C. Hogg, A. McKay, A. de Pennington (2010). Shape detection with vision: implementing shape grammars in conceptual design, Research in Engineering Design 21(4), 235–247.
- Knight, T. (1999). Shape grammars: six types, Environment and Planning B: Planning and Design 26 (1), 15–31.
- Knight, T. (2003). Computing with emergence, Environment and Planning B: Planning and Design 30 (1), 125–155.
- Koning, H. & J. Eizenberg (1981). Frank Lloyd Wright's prairie houses, Environment and Planning B: Planning and Design 8, 295–323.
- Krishnamurti, R. (1981). The construction of shapes, Environment and Planning B: Planning and Design 8, 5–40.
- Krishnamurti, R. & R. Stouffs (1993). Spatial Grammars: Motivation, Comparison, and New Results, in: Fifth International Conference on Computer-Aided Architectural Design Futures (CAADFutures), North Holland, 57–74.

- Li, A., H. H. Chau, L. Chen,W. Y. (2009). A prototype system for developing two- and three-dimensional shape grammars, in: 14th Int. Conf. Computer-Aided Architectural Design Research in Asia.
- McKay, A., S. Chase, K. Shea, H. H. Chau (2012). Spatial grammar implementation: From theory to useable software, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 26, 143–159.
- Shea, K. & J. Cagan (1999). Languages and semantics of grammatical discrete structures, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing, 13(4), 241–251.
- Steadman, P. (1976). Graph-theoretic representation of architectural arrangement, in: The architecture of form, 94–115.
- Stiny, G. & J. Gips (1971). Shape grammars and the generative specification of painting and sculpture, in: IFIP Congress (2), 1460–1465.
- Stiny, G. (1977). Ice-ray: a note on Chinese lattice designs, Environment and Planning B: Planning and Design 4, 89–98.
- Stiny, G. & Mitchell W.J. (1978). The Palladian grammar, Environment and Planning B: Planning and Design 5(1), 5–18.
- Stiny, G. (1980). Introduction to shape and shape grammars, Environment and Planning B: Planning and Design 7, 343–351.
- Stiny, G. (1991). The algebras of design, Research in Engineering Design 2, 171–181.
- Stiny, G. (2006). Shape: talking about seeing and doing. MIT Press.
- Strobbe, T., P. Pauwels, R. Verstraeten, R. De Meyer & J. Van Campenhout (2015). Towards a visual approach in the exploration of shape grammars, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (In Press).

- Taentzer, G. (2004). AGG: A graph transformation environment for modeling and validation of software, Springer, 446–453.
- Tapia, M. (1999). A visual implementation of a shape grammar system. Environment and Planning B: Planning and Design 26, 59–73.
- Trescak, T., M. Esteva & I. Rodriguez (2012). A shape grammar interpreter for rectilinear forms, Computer-Aided Design 44, 657–670.
- Woodbury, R. & A. Burrow (2006). Whither design space?, Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 20, 63–82.
- Woodbury, R., A.D. Radford, P.N. Taplin & S.A. Coppins (1992). Tartan worlds: a generative symbol grammar system, in: ACADIA ’92.
- Wortmann, T. (2013). Representing shapes as graphs: A feasible approach for the computer implementation of parametric visual calculating, Massachusetts Institute of Technology.
- Yue, K. & R. Krishnamurti (2014). A paradigm for interpreting tractable shape grammars, Environment and Planning B: Planning and Design 41 (1), 110–137.