Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2015), *29*, 503–521.
© Cambridge University Press 2015 0890-0604/15
doi:10.1017/S0890060415000475

Toward a visual approach in the exploration of shape grammars

TIEMEN STROBBE,[1] PIETER PAUWELS,[1] RUBEN VERSTRAETEN,[1] RONALD DE MEYER,[1] AND JAN VAN CAMPENHOUT[2]

[1]Department of Architecture and Urban Planning, Ghent University, Ghent, Belgium
[2]Department of Electronics and Information Systems, Ghent University, Ghent, Belgium

(RECEIVED September 13, 2014; ACCEPTED March 15, 2015)

Abstract

The concept of shape grammars has often been proposed to improve or support creative design processes. Shape grammar implementations have the potential to both automate parts of the design process and allow exploration of design alternatives. In many of the existing implementations, the main focus is either on capturing the rationale of a particular existing grammar or on allowing designers to develop a new grammar. However, little attention is typically given to the actual representation of the design space that can be explored in the interface of the implementation. With such representation, a shape grammar implementation could properly support designers who are still in the process of designing and may not yet have a clear shape grammar in mind. In this article, an approach and a proof-of-concept software system is proposed for a shape grammar implementation that provides a visual and interactive way to support design space exploration in a creative design process. We describe the method by which this software system can be used and focus on how designers can interact with the exploration process. In particular, we point out how the proposed approach realizes several important amplification strategies to support design space exploration.

Keywords: Architectural Design; Design Space Exploration; Shape Grammar; User Interface

1. INTRODUCTION

One of the most fundamental questions underlying current computer-aided design (CAD) research is how information systems can effectively support a creative design process (Lawson, 2005). In the current article, creative design is characterized as a specific kind of problem solving in which design problems and design solutions are not only ill defined (Cross, 1982) but also coevolve throughout the design process (Maher & Poon, 1996). In other words, not only is it impossible to capture design problems and solutions in unambiguous, closed and complete representations, even if some feasible representation is found, but it is also continuously changing because of the impact of an evolving design context. In particular, the design problem tends to be constrained by quantifiable and nonquantifiable requirements, whose formulation is often part of the design problem itself. According to this characterization, design problems belong to the category of ill-defined, ill-structured, or even wicked problems (Rittel & Webber, 1973; Cross, 1982). Typical examples of such design problems can be found in the domains of architectural design, urban planning, product design, and so forth.

Starting from this characterization of creative design, diverse strategies can be conceived for supporting creative designers in their decision-making processes. One possible strategy is to amplify designers' capabilities to represent and search a design space (Woodbury & Burrow, 2006). This strategy assumes that designers typically address a design problem, as understood in the above interpretation, through *design space exploration*: they continuously represent and search an evolving "space" of design solutions with the aim of finding an appropriate design solution to an evolving design problem. The common interpretation of the design space exploration task involves the following three steps (Gero, 1994):

- representing many design solutions in a structured network called the *design space*;
- searching and navigating the design space by traversing paths in the network; and
- evaluating possible design solutions against evaluation criteria or design goals.

Reprint requests to: Tiemen Strobbe, Department of Architecture and Urban Planning, Ghent University, J. Plateaustraat 22, Ghent 9000, Belgium. E-mail: tiemen.strobbe@ugent.be

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

The implementation of the above three design space exploration tasks into dedicated algorithms results in an information system that can semiautonomously go through the design exploration task. In this sense, design alternatives are generated automatically by the information system, while evaluation and selection remains with the designer. Among the possible benefits of allowing an information system to do this are that certain parts of the design process can be automated and that design alternatives designers had not thought of before can emerge. According to Knight (2003), *emergence* of design alternatives is available in all information systems, and is even a foundational feature of shape grammars (Stiny, 2007). As a result, the information systems concerned can function as specialized "agents" that act like assistants in a creative design process. In an artificial intelligence context, agents are defined as systems that perceive and act upon an environment in order to achieve a specified goal (Russel & Norvig, 2010). Existing agent applications provide specific functionality in a wide variety of application domains, including text editing, web browsing, information visualization, and so forth. Information systems for design space exploration could to some extent be implemented and used as agent-based systems, leading us forward on the avenue toward a more intelligent role for computers in design, rather than their more limited role as oracles or draughtsmen (Lawson, 2005).

In recent decades, the potential of shape grammars (Stiny & Gips, 1972) to support design space exploration has been demonstrated through several research initiatives. In particular, several shape grammar implementations exist that allow designers to automatically generate a set of shapes or designs that are part of the language of a specific grammar. Many existing implementations focus on capturing the design rationale of a particular existing shape grammar as correctly as possible, or at least as it is commonly understood or appreciated. For example, Grasl and Economou (2012) have implemented a graph grammar for the generation of Palladian-style villas, and Granadeiro et al. (2013) have implemented the Frank Lloyd Wright prairie house grammar (Koning & Eizenberg, 1981). In contrast, more general implementations exist in which the main focus is to allow designers to specify and develop new shape grammars (Hoisl & Shea, 2011; Trescak et al., 2012; Grasl & Economou, 2013). However, current shape grammar implementations hardly support the representation of the design space that can be explored in the interface of the implementation.

In this article we investigate and develop an alternative shape grammar implementation approach that is able to support design space exploration in a visual and interactive way. It is particularly useful for designers who are in the process of developing and exploring new shape grammars in a creative design process. The key properties of this implementation are

- the ability to represent an evolving "space" of design solutions, including ways to visualize both previously generated solutions and new alternatives; and
- the ability to explore this design space, including several amplification strategies for supporting search and navigation.

These features are typically not of central concern to current shape grammar implementations, so they should make the alternative approach presented in this article stand out as a novel approach. In our discussion, we provide some background on the concept of the design space and define terminology that is used throughout the article. Next, shape grammars are discussed and a more detailed overview is given of design space exploration possibilities in current shape grammar implementations. Following that, we describe an approach to representing the design space and its implementation in a proof-of-concept software system. We present and evaluate the main functionalities of this software system using a case study of an implementation of the Frank Lloyd Wright prairie house grammar. In addition, we indicate how designers can interact with the exploration functionalities at hand. Finally, our findings and directions for future work are discussed at the end of the article.

2. THE DESIGN SPACE

In this section, we introduce the basic concepts and terms related to the design space, and explain how design space exploration is a compelling model from a cognitive point of view. We also indicate why design space exploration is a feasible basis for computer implementation and give an overview of several information systems that support design space exploration.

2.1. Design space exploration

The design space can be defined as a dynamic network structure of related (final or intermediate) designs that are visited during a creative design process. The designs in the network structure are represented as *design states*, which are representations of a design at a particular moment in time. The various design states are related to each other through *design paths*. At any moment in time, designers take decisions that lead them from one design state to the next, following particular design paths. Therefore, designers are able to traverse these paths to explore the design space. More specifically, a design path corresponds to a specific designer action or move, such as visiting previously visited design states or finding new unexplored design states. In this sense, exploration comprises both navigation in the design space and the generation of new design paths and states. The collection of design states that designers traverse over time is called the *explicit design space*. The importance of the explicit design space as an expanding library of potentially recoverable design states or explored design paths is shown by Woodbury and Burrow (2006). The collection of all design states that can be traversed if designers follow all possible paths is called

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

the *implicit design space*. The explicit design space is only a small subset of the implicit design space, which can be infinitely large.

In a design state, the available information is reduced to a more manageable level that is suited for a design process and for decision making within this design process. This reduction is often referred to as a "bounded rationality," a term coined by Simon (1957) to indicate that designers always take into account only a small part of the design information available in the real world. Defining the design space thus relies heavily on removing detail from a complex design situation and reducing it to a set of necessary *design features* (or variables). This process is often called abstraction or disambiguation (Simon, 1973). While this is the case, the concept of the design space can be useful for designers to reflect upon the available design states and design paths. While reflecting on and choosing particular design paths, designers receive feedback, based on which they can subsequently revise, further develop, or even entirely reject their chosen exploration strategy. In other words, designers engage in a dialogue or "conversation" with the design space during the design process. In continuously traversing design paths, designers reconsider the design problem and associated design solution, resulting in the coevolution effect that we identified in the Introduction (Maher & Poon, 1996). In this way, the concept of the design space provides a useful model to support a creative design process. The key concepts of the design space are shown in Figure 1.

The design space, as described above, is both a compelling model from a cognitive point of view and a feasible basis for computer implementation. Goldschmidt (2006), for instance, discusses the notion of design space exploration on a more cognition-oriented basis. The author argues that exploration is an important part of any inquiry or experiment that designers undertake. These inquiries or experiments are to be understood in the context of Schön's theory of the designer as a "reflective practitioner" (Schön, 1983). Schön describes design as a combination of different kinds of experiments: exploratory experiments, move-testing experiments, and hypothesis testing. Designers start an inquiry by formulating the design situation, after which they can perform exploratory experiments to either find new design solutions (move testing) or alter their exploration strategy (hypothesis testing). Under the effect of these inquiries or experiments, both the design problem and corresponding solutions coevolve within the design space. As demonstrated above, this continuous interaction between designers and the design space is an important feature to support the creative design process.

2.2. Design space exploration in an information system

The design space is also a feasible basis for computer implementation and one of the long-standing motivating ideas underlying CAD research in recent decades. The first attempt to formalize the concept of design space was made in research on design methods in the 1960s (Jones & Thornley, 1962), and further steps were taken with the introduction of artificial intelligence (for example, the use of numerical optimization in the work of Radford and Gero (Radford & Gero, 1980, 1988; Gero et al., 1983) and cognitive science (Akin, 2006). To date, many information systems and digital tools for creative designers have been conceived as systems for design space exploration. Such systems are able to support designers in exploring an implicit and explicit design space containing either previously visited design states or new, unexplored design paths. Specific examples of the former are case-based reasoning tools that allow designers to find information that could in some way be useful for their design. An overview of early research efforts and more recent work in case-based reasoning is given in the paper of Goel and Craw (2006). Specific examples of tools for design generation are parametric design tools (Tidafi et al., 2011; Turrin et al., 2011; Charbonneau & Tidafi, 2013) and grammar-based design tools (Shea & Cagan, 1999; Schaefer &

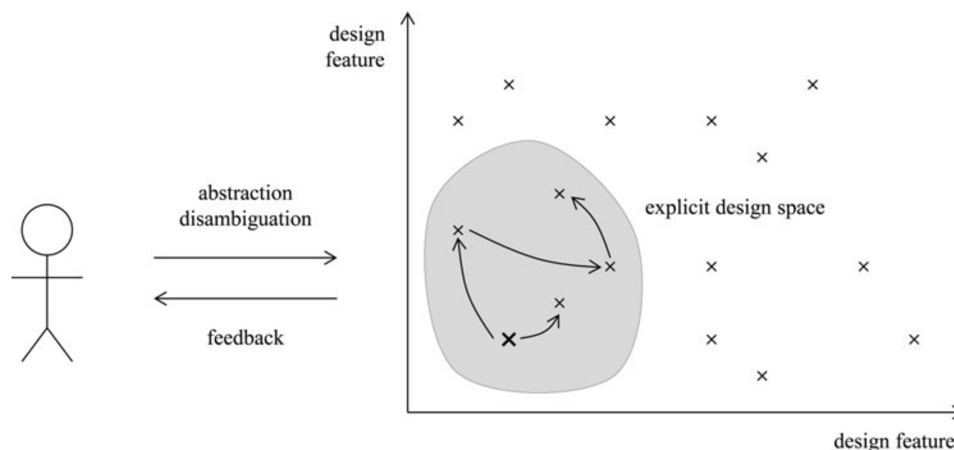

Fig. 1. The design space consists of design states and design paths. The initial design state (**x**) and subsequent design states (x) are related by design paths (black arrows). The implicit design space consists of the complete collection of design states and design paths. The explicit design space is the collection of visited design states.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

Rudolph, 2005; Geyer, 2008). Generation in parametric design tools corresponds to assigning different values to a number of (geometrical) parameters, while grammar-based design tools deal with alternative systems or rearrangements of design components.

In the context of an information system, most of the terms in Figure 1 (design space, design state, design path, etc.) are typically associated with a formal or structured representation. In particular, the formalized design space can be seen as a network structure of design states (nodes in the network) and design paths (arcs between the nodes). In the specific case of grammar-based design tools, the design space also includes an initial design, represented as the root design state. A design path corresponds to the generation of a new design state by means of a specific generative mechanism. Based on the chosen generative mechanism, the information system is able to generate an implicit design space consisting of various design states that can be reached through a number of design paths. Within the scope of the formalized design space, information systems are able to guide exploration toward a design goal that is at best near optimal, both in a quantifiable and nonquantifiable way. Because of the bounded nature of the information taken into account in the formalized design space, a fully optimal design state cannot be reached. As a result, we can talk about a "satisficing" process (Simon, 1956). Satisficing, a combination of satisfy and suffice, directs exploration for design alternatives toward criteria for adequacy within a bounded rationality, rather than a fully rational solution. As soon as a goal formulation and a set of design constraints are added, the exploration task can be formulated as an optimization problem (Gero & Kazakov, 1996).

As indicated above, the act of design space exploration is limited by the availability of a finite number of resources. In the context of information systems for design space exploration, the design space is also heavily computationally bound. However, such information systems draw their utility from allowing designers to satisfice design alternatives against goal criteria, rather than optimizing performance toward a fully optimal design state (Simon, 1956). Therefore, representing the design space facilitates a useful interaction between designers and the tool that is used for design space exploration. In the following section, we give an overview of design space exploration possibilities in current shape grammar implementations.

3. AN OVERVIEW OF SHAPE GRAMMARS AND THEIR IMPLEMENTATIONS

3.1. Definition and terminology

At any moment in time, designers have a number of design moves that they can apply in order to change the current design state into another design state. By formalizing these design moves and using the computational power of an information system, designers might be able to foresee the possible effects of taking a particular design move. An interesting and widely developed approach to support design space exploration is the "codification" of design moves (or paths in the design space) as *rules*. This codification coheres with the ability of computer tools to store and represent rules, while designers focus on specifying, exploring, and developing design alternatives (Chase, 2002). On the one hand, such rules need to comply with an exploration process that is well bound and limited in breadth. The advantages of considering a limited number of coherent design alternatives, rather than many widely different alternatives, have been indicated in several (empirical) research studies (Goldschmidt, 2005, 2006). On the other hand, such rules need to be flexible to allow the generation of design alternatives that are meaningful and diverse enough to be useful to the designers. According to Goldschmidt (2005), designers can perform exploration, within a deliberately limited yet broad enough design space, in which the goal is refinement of a coherent and well-developed design idea.

The existence of such rules has been demonstrated for Palladian villas by Stiny and Mitchell (1978), for Alvaro Siza's Malagueira houses by Duarte (2005), for Frank Lloyd Wright's prairie houses by Koning and Eizenberg (1981), and so on. In the domain of creative design, these rules are often formulated using shape grammars (Stiny, 2007). Shape grammars were originally introduced by Stiny and Gips (1972) as a way of analyzing and synthesizing paintings, and later extended to various other application domains. They are a class of production systems that generate geometric shapes or solids in Euclidean space (E^2 or E^3). More specifically, a shape grammar is a 4-tuple $\langle V_T, V_M, R, I \rangle$, consisting of a finite set of terminal shapes (V_T) and marker shapes (V_M), a finite set of shape rules (R), and a nonempty set of initial shapes (I) that is part of $V_T \cup V_M$. The shapes defined in the set $V_T \cup V_M$ form the basic elements for the definition of shape rules in the set R and the initial shape I. New shapes are generated by iteratively applying shape rules to the initial shape. A shape rule is described as an if-then statement $A \rightarrow B$, and can be applied when the pattern shape A (if-part) can be detected in a given shape C. This matching process can occur under certain Euclidean transformations (t) to find more possible matches: translation, rotation, scaling, and other linear transformations. Rule application results in "subtracting" the transformed shape A from C, and "adding" the transformed replacement shape B (then-part). This results in a new shape $C' = C - (A) + t(B)$. Further details about the shape grammar formalism are given in the work of Stiny (2007).

Using the design space terminology defined in the previous section, shape rules R correspond to design steps in the design space. Shape rules are a generative mechanism to derive new design states and shapes in particular. Each rule of the form $A \rightarrow B$ involves replacing a transformed version of A by an identically transformed version of B. Therefore, rule application corresponds to navigating from one design state in the design space to another. Shape rule application can include deleting, adding, and transforming

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

parts of the shape. A design path consists of multiple steps. An initial shape I corresponds to an initial design state, upon which applicable rules are applied iteratively. The set of shapes that can be generated using a specific initial shape and set of rules is called the *language* of the grammar $L(G)$. This language $L(G)$ defines an implicit design space, which is the part of the design space that can be explored using this specific grammar G. As soon as the shape grammar is defined, the design space is implicitly defined, and rules can be used to explore a deliberately limited, yet broad enough, design space. A similar definition of a design space in the context of shape grammars was proposed in early work of Gero and Kazakov (1996). Design states or shapes that are beyond the scope of the implicit design space or language can only be reached by either changing the current shape grammar or manually manipulating shapes with no regard to the current shape rules. The latter strategy enables designers to make shortcuts in the design space, allowing "on the spot" experimentation. The exploration process is terminated either when no applicable rules are found or when a given goal state is satisfied.

3.2. Overview of shape grammar implementations

To date, a broad range of research has been done on shape grammar implementations. An overview of research efforts up to 1999 is given in the work of Gips (1999), while more recent shape grammar implementations are discussed by McKay et al. (2012), and an overview of the interaction possibilities in several shape grammar implementations is given by Chase (2002). Based on these overviews, it is clear that current implementations of shape grammars have made valuable contributions to enabling subshape recognition and shape emergence, allowing parametric rules, supporting curvilinear shapes, and so forth. However, researchers have focused to a far lesser extent on representing the design space and supporting exploration in the interface of the implementation.

In the following overview, the focus is on the capacities of several existing shape grammar implementations to represent a design space, support the generation of new design states, and support exploration of already visited design states (the explicit design space). In particular, we discuss the *GEdit* shape grammar implementation (Tapia, 1999), because it is one of the first implementations that considers the issue of exploration. In addition, three more recent shape grammar implementations are included: *Spapper* (Hoisl & Shea, 2011), *SGI* (Trescak et al., 2012), and *GRAPE* (Grasl & Economou, 2013). We either have obtained a working copy of the implementation discussed or determined its functionality from a published paper. Our comparative overview of these four implementations is summarized in Table 1.

GEdit, developed by Tapia (1999), is an early example of a shape grammar implementation in which design space exploration is considered to a certain extent. The design space is represented through a visualization of the current shape, together with a separate visualization of shape alternatives that are the result of a single rule application. The shape alternatives are structured in a two-dimensional array in which designers can browse and examine shapes. The generation of

Table 1. *Comparison of design space exploration possibilities in several shape grammar implementations*

Name	GEdit[a]	Spapper[b]	SGI[c]	Grape[d]
(1) Design Space Visualization				
Current design state	Visual	Visual	Visual and symbolical	Visual
Application results	Two-dimensional array	Individual results	List	Individual results
Derivation history	No	No	Yes, current derivation	No
(2) Generation of Alternatives				
Rule application	(Semi)automatic	(Semi)automatic or manual	Automatic	(Semi)automatic
Automatic detection of applicable rules	No	No	Yes	No
Manual intervention	No	No	No	Yes
(3) Navigation and Storage of Shapes				
Backtracking	Yes	(?)	No	No
Save & reuse designs	No	Yes, current (.dxf, etc.)	Yes, current (.xml)	Yes, current (.dxf)
Save derivation history	No	No	No	Yes, current derivation (.dxf)
Reuse derivation history	No	No	No	No

[a]Tapia (1999).
[b]Hoisl et al. (2011).
[c]Trescak et al. (2012).
[d]Grasl et al. (2013).

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

design alternatives happens in a semiautomatic manner: all possible rule applications are calculated automatically, after which designers can select and delete alternatives manually. The author recognizes the importance of a design space navigation mechanism that is richer than pure generation: “the designer explores the language of designs, generating designs, ... backtracking to a previous design, or saving the current state” (Tapia, 1999). However, these features are only partially implemented in the current version of GEdit.

Spapper is a more recent shape grammar implementation, developed by Hoisl and Shea (2011), which provides a visual way to edit or develop shape grammars. Similar to GEdit, the interface visualizes a single shape that designers can alter over time. Therefore, only the current design state in the design space is visualized. Designers can choose to explore the language of a grammar using manual rule application (through a predefined sequence or manual selection), semiautomatic rule application, or automatic (random) rule application. However, it is not possible to automatically detect all rules that can be applied to the current shape. It is possible to store a current shape using the standard functionality of the underlying CAD system. However, the history of the rule applications used to generate this shape is not stored. Therefore, the stored shape is added to the set of initial shapes as part of a new exploration process.

SGI, developed by Trescak et al. (2012), includes several features to explore the language of a shape grammar in an interactive way. First, a render line view shows the current derivation line of the exploration process. This provides designers with the possibility of tracing the execution of the shape grammar from the initial shape to the current shape. Second, a list view in which the resulting shapes of one rule application are stored, allows designers to select and delete shape alternatives manually. Third, the current shape is displayed in a visual manner and a symbolic manner by showing a list of all the design properties (for example, a name, position, etc.). This allows designers to compare shapes with regard to both geometric and nongeometric design properties.

GRAPE is a graph-based shape grammar library, developed by Grasl and Economou (2013). Several interfaces to the GRAPE library have been developed, either based on commercial CAD packages or as web applications. The design space is represented through a single design state view, visualizing the current shape. The generation of alternatives happens in a semiautomatic manner, because designers explore and select shapes one at a time. Based on the functionality of the underlying CAD package, it is possible to manually intervene in the exploration process by adding, deleting, and modifying shapes without the use of shape rules. This feature is reflected in the structure of the user interface, which distinguishes between a common CAD mode and a grammar mode. The CAD functionality makes it possible to store the current shape (without history) as a .dxf file. In addition, it is possible to export a snapshot of the current derivation line (from the initial shape to the current shape). This derivation cannot be reused in a new exploration process. In recent work (Grasl & Economou, 2014), GRAPE has been extended with a framework to support selection agents. Therefore, it is possible to enumerate a set of design alternatives, following an agent-specific (extensive, goal-directed, etc.) approach.

4. PROPOSED APPROACH

Shape grammars define a suitable formalism to encode design moves and allow the exploration of the language of a grammar, which defines an implicit design space. However, the overview of current shape grammar implementations discussed above indicates that the representation of the design space is often limited to a small subset of design states and paths (Tapia, 1999; Trescak et al., 2012) or even a single design state (Hoisl & Shea, 2011; Grasl & Economou, 2013). In addition, navigation is limited to the derivation of alternatives, without possibilities for backtracking and reusing stored design states or paths. In their seminal paper about design space exploration, Woodbury and Burrow (2006) argue that design space exploration using (shape) grammars might be infeasible, because shape rules do not support the representation of the explicit design space:

> A flaw in standard rule-based accounts of design space exploration in implicit space is that the usual formulation of rules cast navigation solely in terms of derivation, thus putting the landscape of the explicit space forever beyond the sight of the navigator.

In particular, shape rules of the form $A \rightarrow B$ replace a transformed version of A with an identically transformed version of B, without keeping track of the parent shape. Therefore, it is not possible to represent the explicit design space, which consists of design states that have been visited previously. However, the representation of the explicit design space is essential to support design space exploration. It allows designers to consider multiple design states or design alternatives simultaneously (Fig. 2a). The availability of design alternatives is an important aspect, because these alternatives provide multiple design paths toward (one of) the specified goal state(s). Because not all design information is available in the design space, alternatives are important to satisfice against what is known about the design space. By contrast, the explicit design space allows designers to store and reuse prior design states or design paths previously discovered. Woodbury and Burrow (2006) stress the importance of the explicit design space as an expanding library of potentially recoverable design states or explored design paths. Therefore, it is important to allow back and forth navigation in the explicit design space (Fig. 2b).

The argument of Woodbury and Burrow (2006) about the shortcomings of shape grammars in supporting design space exploration is debatable. For example, Krishnamurti (2006) mentions that it should not be difficult to envisage a shape grammar implementation that maintains a history of deriva-

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

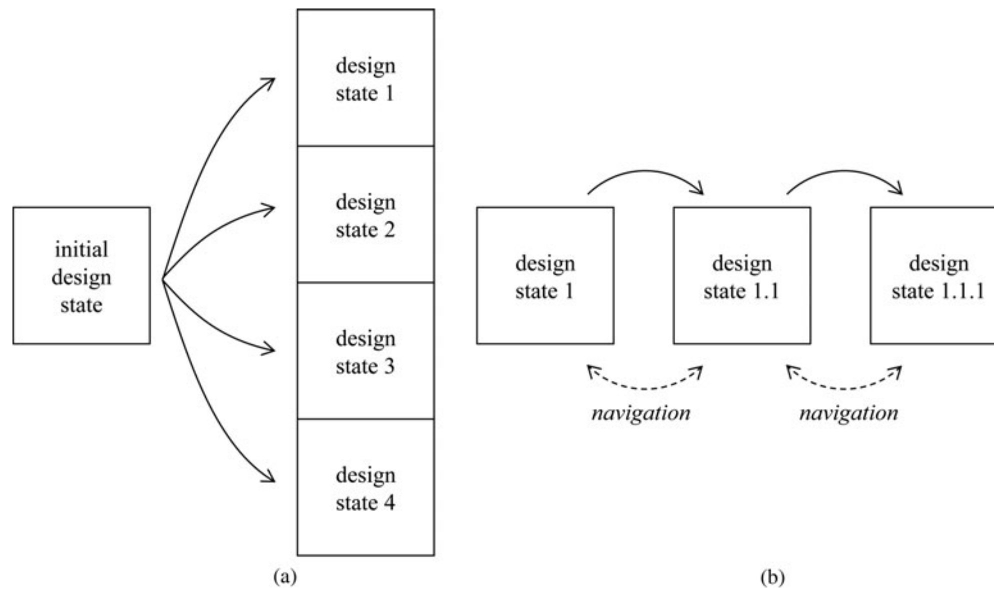

Fig. 2. The generation and representation of alternatives (a) and back and forth navigation in the derivation process (b) are two important design space exploration strategies. The forward arrows (solid) correspond to specific rule applications, and the dashed arrows correspond to navigation (i.e., returning to a previously stored design state).

tion. In this article, we follow this line of thought and propose an approach to keep track of the explicit design space. In particular, a shape C' of the language $L(G)$ is extended with a pointer to a parent shape C. This pointer corresponds to a specific rule r in the set of R that was applied to this parent shape C in order to generate the shape C'. As a result, the explicit design space is defined as a hierarchical tree structure in which the derived shapes constitute the nodes of the tree (Fig. 3). The nodes are connected through parent–child relations, in which each node (excluding the root node) has exactly one parent node. In particular, the parent–child relation is defined as a rule application $A \rightarrow B$ between a parent shape C and child shape $C' = C - t(A) + t(B)$. The current design path is indicated using a gray color (Fig. 3).

In the computer science domain, a tree is a widely used data structure for modeling hierarchical data with parent–child relations. For example, version control systems, used in software engineering, use directed tree structures to visualize one or more parallel lines of development. In the context of design space exploration, we use trees as a practical and elegant solution to keep track of the explicit design space. Moreover, several operations can easily be performed, including generating new child nodes, and walking through the nodes or "traversing" the tree. These operations correspond to the design space exploration strategies explained above: generation of alternatives and navigation and backup in the explicit design space, respectively. The operations are discussed below.

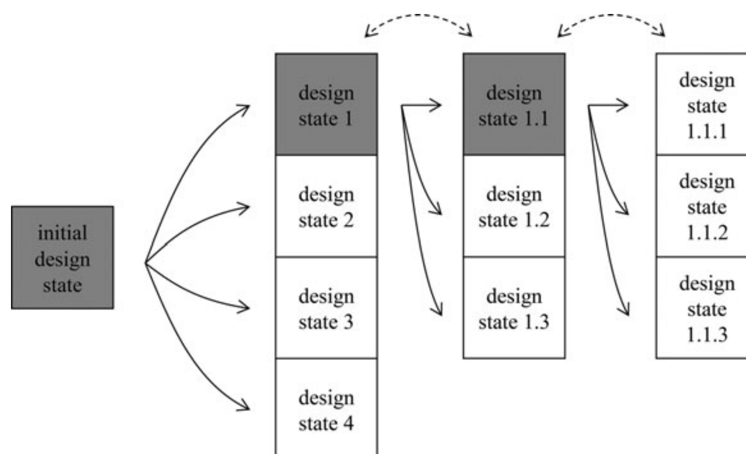

Fig. 3. The representation of the explicit design space as a tree. The forward arrows (solid) correspond to specific rule applications, and the dashed arrows correspond to navigation. The current design path is indicated in gray.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

The derived shapes constitute the nodes of a tree, while the possible rule applications constitute the relations between the nodes. We have chosen to store the full design states in the tree nodes, rather than storing the differences between the parent state and the child state. This approach has important benefits. First, design states can be recalled in a new design project without reference to its parent shape. In this case, the design state can be considered as a new root design state in an exploration process. Second, it is not necessary to reconstruct shapes, which may take a long time for shapes with a long derivation history. A possible drawback of this approach is that more space is needed to save the full design states. One possible way to reduce the size of the explicit design space is by using equivalence classes rather than individual shapes. Such equivalence classes need to store only one representative, and hence this representative has pointers to a nonempty set of parents instead of a single parent shape. When alternative rule sequences arrive at the same shape, this specific shape has multiple parent shapes. In this case, the resulting design space structure is no longer a tree, but a network. Therefore, equivalent shapes are stored only once, while in the case of trees, all generated shapes are considered to be unique, even when they are visually equivalent. In the context of this research, trees are found to be sufficiently effective in supporting several aspects of design space exploration.

4.1. Generation of alternatives

This section describes a method for the generation (and representation) of design alternatives or shapes. In order to implement this generation process, several steps must be taken:

- the automatic determination of rules that can be applied to a given shape;
- the automatic generation of children by executing the applicable rules; and
- the manual selection of a new shape that can be further explored.

The first step in the generation process is the determination of rules that can be applied to a given shape. This step corresponds to detecting the pattern shape A of the rules under a certain transformation $t(A)$ in the given shape. The detection of a pattern shape in a given shape is often referred to as the subshape detection problem. In recent work, Yue and Krishnamurti (2014) argue that subshape detection is a computationally complex task, especially in the case of parametric rules. Nevertheless, various solutions have been proposed in the literature. For example, using an underlying graph representation for shapes, subshape detection can be devised as a subgraph isomorphism problem (Grasl & Economou, 2013). While this problem has proven to be NP-complete, algorithms that calculate solutions in reasonable time are available in several graph transformation libraries (Taentzer & Rudolf, 1998). This step results in a set of applicable rules, or morphisms $f: A \rightarrow C$, between the pattern shape A of a rule and a given shape C. These morphisms describe a structure-preserving mapping between the rule and the given shape.

The second step is to generate new shapes by executing the applicable rules. Following the expression $C' = C - t(A) + t(B)$, a new shape is generated after subtracting the transformed pattern shape $t(A)$ from the given shape C and adding the transformed replacement shape $t(B)$. As demonstrated in Section 3.2, many existing implementations show a single shape that can be altered over time by applying rules. Rule application can be performed automatically by random selection, or manually by user selection. The approach described here is different, because all possible rule applications are performed iteratively. This results in a list of possible shape design alternatives, similar to the list view used in SGI (Trescak et al., 2012). In particular, the morphisms $f: A \rightarrow C$ of the first step are used to derive a set of possible shape alternatives from a given shape C. Although this is computationally inefficient, it is advantageous in the context of satisficing to provide designers with a set of possible alternatives. In order to reduce the computational complexity of this step, designers can define *a priori* which rules to consider for exploration and which rules to leave out.

The third step is the manual selection of a new shape for further exploration. As a result of the repeated application of rules in series, a list of possible alternatives is given that can be considered for further exploration. As a result, designers can manually select one of the generated child nodes. This chosen tree node can in turn be reconsidered for the generation of new child nodes. Therefore, the three steps described in this section are repeated at each level in the tree.

Following these three steps, a tree with multiple levels and branches is constructed. The different levels of the tree correspond to specific moments in time during the exploration process, while the branches of the tree correspond to the shapes that are generated at these specific moments in time. Therefore, it is possible to represent some aspects of the design space, particularly the explicit design space described above. More specifically, automatic detection and repeated application of rules in series enables the generation and representation of design alternatives as shown in Figure 2a.

4.2. Design space navigation and backup

In this section, a method is described for enabling several interactions between designers and the explicit design space: interactive navigation, backup of design states and paths, and recall of design states and paths. An important feature of trees is the possibility of traversing tree nodes by means of the connections between parent and child nodes. Once the tree is constructed, it is possible to visit tree nodes in several ways. For example, it is possible to traverse tree nodes level by level, where the root node is visited, followed by the child nodes, grandchild nodes, and so forth, until all nodes have been visited. Another way is to walk the tree nodes in the opposite direction, starting from child nodes and visiting their respective parents (ancestors). These operations correspond

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

to designers' ability to go back to previously discovered shapes or design states in the explicit design space (Fig. 2b). When designers arrive at a particular node in the tree, it is possible to select a sibling node and generate new child nodes. It is also possible to prune a specific branch or even a whole section of the tree. This is, for example, the case when designers consider a specific area of the explicit design space to be no longer relevant. Using the traversal functionalities described above together with an appropriate user interface, designers are able to navigate the explicit design space in an interactive way. An illustration of the navigation possibilities is shown in Figure 4.

The representation of the design space as a tree also enables several backup strategies. In almost all commercial CAD systems, backup along a single derivation thread is typically supported through an "undo" command. In the context of our proposed approach, the backup of design states and paths previously considered is supported. At any time, designers can choose to recover a prior shape or design state in the explicit design space. However, it is also beneficial to enable designers to recover design states that are unrelated to the current exploration process. Woodbury and Burrow (2006) define the concept of *recall* as the metaphor for such distant access. More specifically, recall enables designers to recover shapes or design states that were generated at a different time, in a different project or design task, or by a different designer. Examples of recall in commercial CAD systems include catalogs of drawings and models, libraries with specific functionalities, and so on.

In the context of grammar-based information systems, an additional "persistence" system is needed to store shapes and sequences of rule applications. A database can support such functionality, because design states can be added to and pulled from the database based on various semantic or verbal search criteria. With such a recall system in place, it is possible to reuse design states or even design paths by grafting these paths onto a new tree. In other words, shapes and grammar rules can be recalled in grammars and design projects other than those in which they were developed. A number of scenarios for recall are possible: the recall of shapes in a new design project, the recall of grammar rules in a new design project, and the recall of shape derivations in a new design project. The ability to store and recall shapes and shape derivations is an important design space exploration amplification strategy, but it also has a positive impact on the computational complexity of generating alternatives, because already visited design states can be pulled directly from the database. Therefore, an explicit design space can consist of both newly generated shapes and previously generated shapes. In the following section, we discuss a possible prototype for such a system and demonstrate its potential benefits and drawbacks.

5. IMPLEMENTATION

To evaluate the approach described in this article, we have developed a proof-of-concept software system. The implementation is based on a JAVA development environment for graph transformation called *AGG* (http://user.cs.tu-berlin.de/~gragra/agg/). The existing editor in AGG is used for developing a grammar, and the existing algorithms for automatic rule matching and rule application. Therefore, the grammars used in this research are implemented as graph grammars. The use of graph grammars to describe shapes or solids in Euclidean space was first introduced in the work of Fitzhorn (1990) and further elaborated in several other research studies, including the work of Heisserman (1994) and Grasl and Economou (2013). Due to the chosen graph environment,

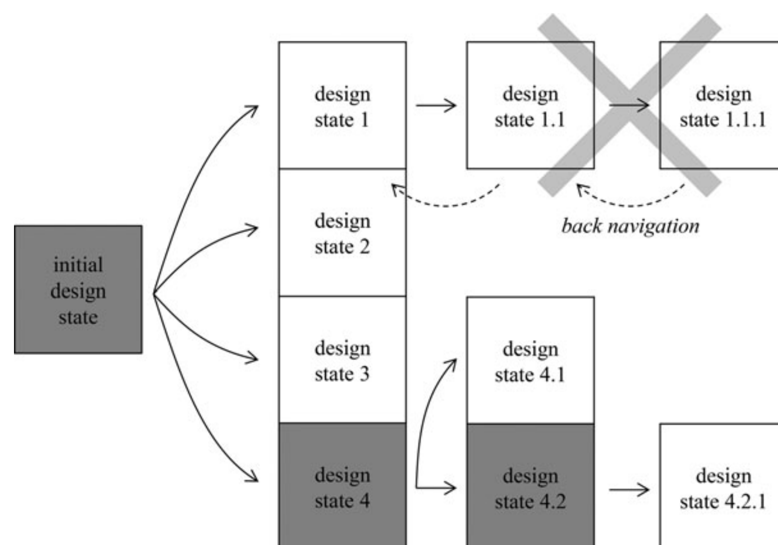

Fig. 4. An example of back navigation in the explicit design space from design state 1.1.1 to design state 1, after which new design states are generated starting from design state 4.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

shape rules must be defined as graph transformation rules, and several aspects of shape grammars are not taken into account. For example, several shape grammars use "emergent" shapes, which are shapes that are not predefined in a grammar but emerge from the shapes generated by rule applications (Knight, 2003). However, our proposed approach does not depend on the chosen underlying environment, and could be incorporated into other shape grammar implementations that do support emergent shapes.

The focus of the software system is to provide a visual, interactive interface that supports designers in exploring the language of a grammar. In particular, the system provides the following amplification functionalities:

- representation of the design space as a visual whole, allowing designers to compare multiple alternatives simultaneously;
- visualization of the explicit design space, which is the collection of design states that have been previously visited, and the current design path;
- representation of design states in both a visual and a symbolic (textual or verbal) manner;
- capacity to backup design states and recall previous design states and design paths using a relational database; and
- interactive navigation within the (explicit) design space.

5.1. Exploration

The user interface of the software system consists of several windows with specific functionalities. The "exploration" window visualizes the explicit design space and allows designers to generate new shape alternatives and navigate in the explicit design space. This window is shown in Figure 5. The user interface also provides a navigation toolbar (top), a list of rules (upper left), and a visualization of the initial design state (lower left). The navigation toolbar includes commands to reset the current exploration process, generate new shape alternatives, and recall shapes from a database system. The list of rules visualizes the set of rules R of the current grammar. The designer can select *a priori* which rules to consider for exploration, thereby reducing the computational complexity of the generation step. Currently, a short description of the rules is shown, but the implementation of an intuitive and visual rule editor (e.g., in the work of Hoisl & Shea, 2011) could improve the user's experience. The initial design state can be pulled from a database system or imported from external CAD files.

We have chosen to visualize the explicit design space as a whole, rather than displaying a single design state that can be altered over time. The shapes or design states are organized in a two-dimensional grid of $m \times n$ cells. Each cell in the grid layout contains a visual representation of the shape, and has zoom and pan functionalities, allowing designers to consider rule applications that are more subtle and harder to identify. Each column stores the n shape alternatives at a specific level in the design space. Once a specific shape is selected for further exploration by the designer, a new set of shapes is generated in a new column $m + 1$, which is added to the grid. The current design path is indicated in gray to show the position of the designer in the design space. In addition, it is possible to navigate back in the explicit design space by selecting a shape at a previous level or column $x < m$. If this shape has not been expanded yet, a new set of shape alternatives is generated in a new column $x + 1$, which is then added to the grid. If this is the case, the descendants of the selected shape or design state are no longer visible, and replaced by a new set of shape alternatives. As a result, designers are able to return to a prior design state in the design process. Perhaps more important, back navigation to prior design states makes new areas of the design space available for future exploration.

The layout of the user interface is an intuitive interpretation of the explicit design space as a tree, as shown in Figure 3. The tabular layout enables designers to scroll both in a horizontal direction and in a vertical direction, allowing them to examine multiple alternatives and shape derivations in a limited amount of screen space. In the proposed approach, only the current design path and the sibling design states are shown in the interface. While the visualization of the complete explicit design space can be useful in some cases, it quickly becomes too complex for larger design spaces. Nevertheless, it is possible to explore the complete explicit design space by selecting different shapes in the tree. Furthermore, in addition to the proposed two-dimensional grid layout, other layouts can be useful in specific cases: for example, a radial layout can be useful to show visually adjacent design states or shapes, or the combination of "inworld" views (a single design state and its immediate parent or child nodes) and "outworld" views (an overview of the whole design space), as proposed by Papanikolaou and Tunçer (1999). The different components of the layout can be changed or resized according to user preferences. Currently, new shapes are generated one step or rule application from the selected shape in the tree. This could be extended using search algorithms for the generation of new shapes at a greater depth, for example, breadth-first, depth-first, or even heuristic search algorithms, if some heuristic function is available to guide the search process. Another important issue is the generation time of new shape alternatives, because this influences the responsiveness and usability of the software system. A case study to measure the generation time of shapes of different complexity is discussed further in this article.

Design states are represented in both a visual and a textual manner. A visual representation of the shape or design state is given, which facilitates the comparison of shape alternatives. Additional information about the shape is given: the rule that was used to create this shape, the grammar that was used, the project name, and so forth. Other relevant information can also be added, because a design process is often associated with a specific program, design intent, or design reasoning. This additional information is often expressed verbally and

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

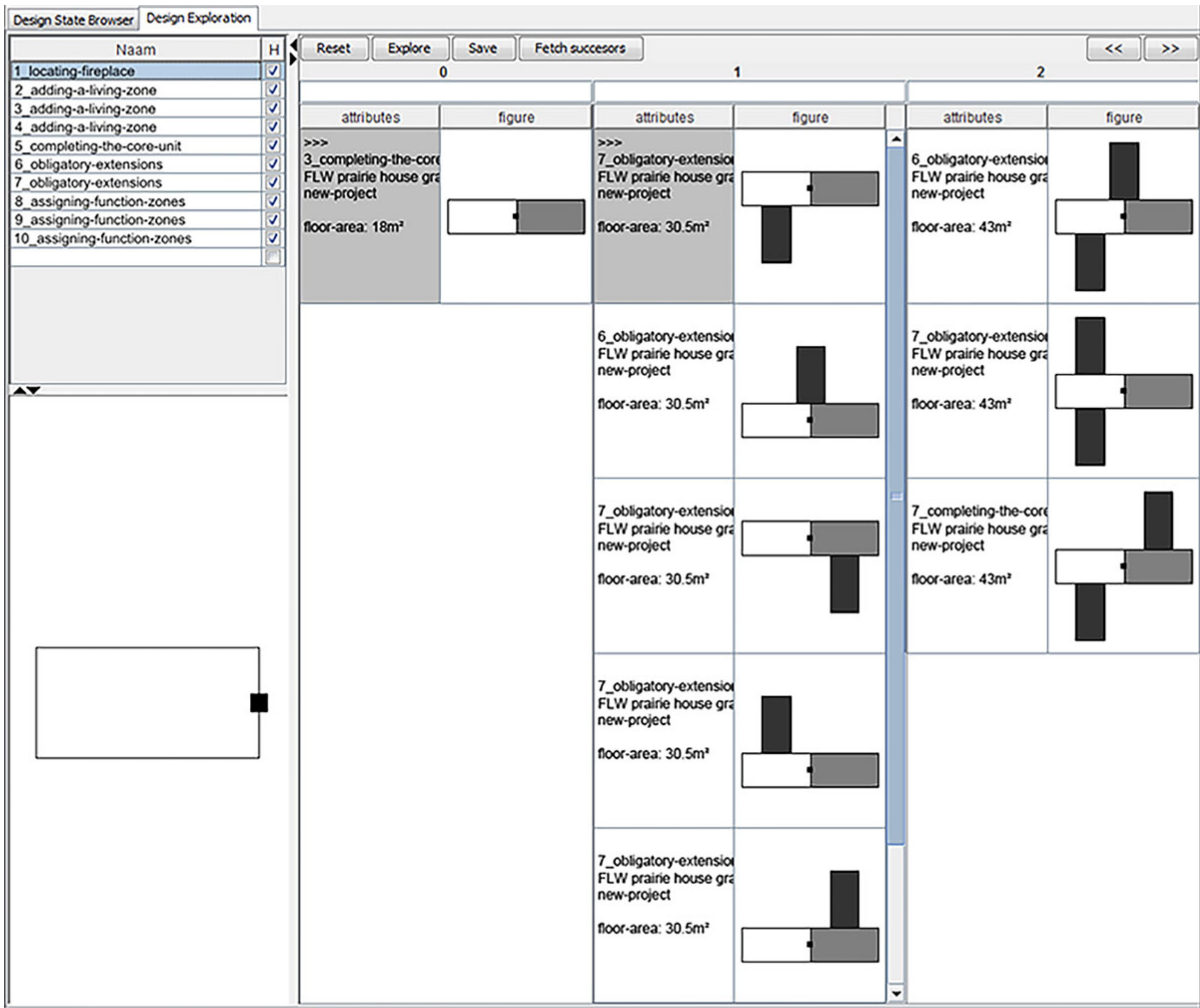

Fig. 5. Screenshot of the graphic user interface of the developed software system. The user interface also provides a navigation toolbar (top), a list of rules (upper left), and a visualization of the initial design state (lower left). The current design path is indicated in gray.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

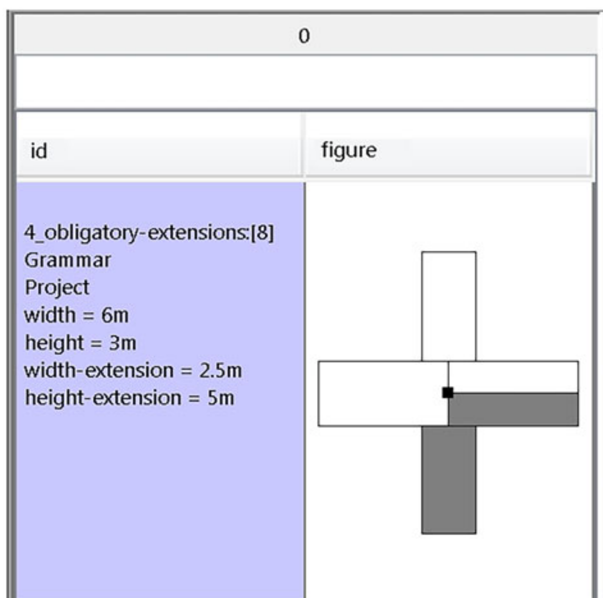

Fig. 6. Visual and symbolic (textual) visualization of a design state.

is associated with a specific design state. Therefore, this information is used to compare shape alternatives toward nonvisual design properties, as well as to facilitate design state recall at a later time. A detailed visualization of a design state is shown in Figure 6. In this example, additional information is provided about the grammar, project, and some salient design features (width and height of spaces).

5.2. Backup and recall

Through the "design state browser" window, the user interface also enables designers to store and recall shapes (design states) and shape derivations (design paths). These shapes and derivations can then be recalled in a later design process. A relational database is implemented in the open-source *MySQL* (www.mysql.com) database system in order to provide this functionality. In the context of this article, we have defined three separate entities that can be stored in and pulled from the database: design states, projects, and grammars. A design state entity represents a design at a specific moment in time. More specifically, a design state entity consist of a unique identifier, a date, a description of the shape as a graph, and a rule that was used to create the design state. A project entity represents the context in which a design state was generated, for example, a design process or a project name. A grammar entity contains the rules and the node types that are used to generate a design state. Project and grammar entities consist of an identifier, name, date, and description of the entity. This experimental setup of three entities has proven to be sufficiently effective in demonstrating the potential benefits and drawbacks of design state backup and recall. However, in order to support backup and recall in more realistic and complex design situations, the database needs to be further elaborated, particularly with regard to the semantic information associated with the shapes. The entity-relationship diagram of the database is shown in Figure 7.

Two relationships between the entities are defined. First, a one-to-many relationship is defined between a design state and a project, indicating that a design state is generated within the context of the given project. Similarly, a design state is generated using a specific grammar, which is shown by a one-to-many relationship between a design state and a grammar. When a design state is recalled, the design state is pulled from the database, together with the corresponding project and grammar. Moreover, it is possible to start a new design project and recall an existing grammar and the corresponding rules from the database. Therefore, both design states and grammars can be recalled in a design project other than the one in which they were developed. In order to recall prior design states, it is possible to search the database for design states using a project name, a grammar that was used, or a combination of both. A further elaboration of the database allows designers to specify more complex search queries.

Second, a one-to-many "predecessor–successor" relationship is mutually defined between design states. This relationship indicates that a design state can have multiple child design states, but only one parent design state. In addition, we keep track of the rule that was applied to the parent in order to generate the design

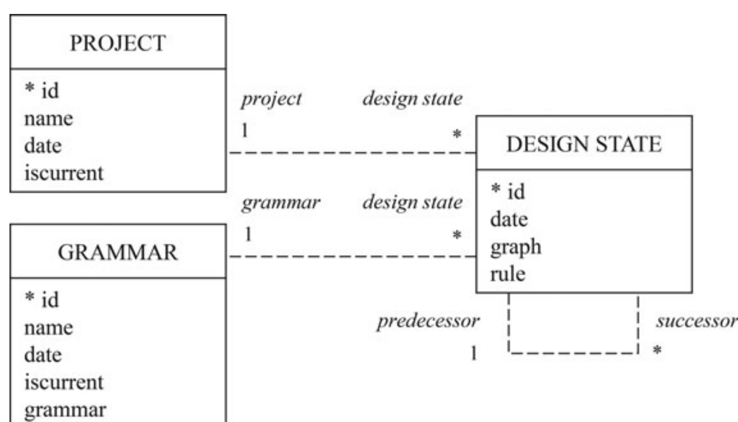

Fig. 7. Entity-relationship diagram of the database.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

state. Therefore, it is possible to fetch successors of a design state from the database without having to regenerate them. This not only allows recall of shape derivations in a new design project but also has a positive impact on the computational complexity of generating alternatives, because already visited design states can be pulled directly from the database.

6. CASE STUDIES

In order to investigate the feasibility of the proposed approach, two case studies are carried out, each implementing a different grammar from the architectural design domain. First, we implemented a subset of simplified compositional shape rules from the Malagueira grammar, originally developed by Duarte (2005). This case study is used to evaluate the generation time of new design alternatives as a measure of the system's responsiveness and the usability of the navigation method. Second, we implemented the Frank Lloyd Wright prairie house grammar, developed by Koning and Eizenberg (1981). This case study is used to demonstrate how designers can interact with the grammar by using exploration, backup, and recall of the shapes in this grammar in a different design project.

6.1. Malagueira compositional rules

In this case study, several compositional shape rules from the Malagueira grammar, developed by Duarte (2005), are implemented. The shape rules from the original grammar were translated into graph transformation rules, because of the underlying AGG environment. The visual representation of the selected rules is shown in Figure 8 (left), together with the graph representation of these rules (right). In particular, we considered rules for perpendicular dissection of rectangles (rule 1 and 2), a rule for deleting markers (rule 3), and a rule for diagonal dissection of rectangles (rule 4). Because the scope of this article is focused on the interface, rather than the underlying framework used to implement shape grammars, only the essential details about the graph-theoretic representation of shapes are included here. An extensive overview and discussion of several graph representation possibilities of shapes is given in the work of Grasl (2013). In our work, we have chosen to represent the topology of the shape in terms of its basic elements (points and maximal lines) and the relations between the basic elements (part–relation graph). The graph transformation rules shown in Figure 8 consist of point nodes (white), edge nodes (black), and label nodes (gray). Rules 1 and 2 from the original grammar are implemented using the same graph transformation rule and different constraints on the attributes of the graph nodes.

The generation time of new design alternatives is a good measure of the system's responsiveness and the usability of the navigation method. A reasonably low generation time enhances intuitive exploration of the grammar, because new design paths are quickly unfolded in the design space. Because

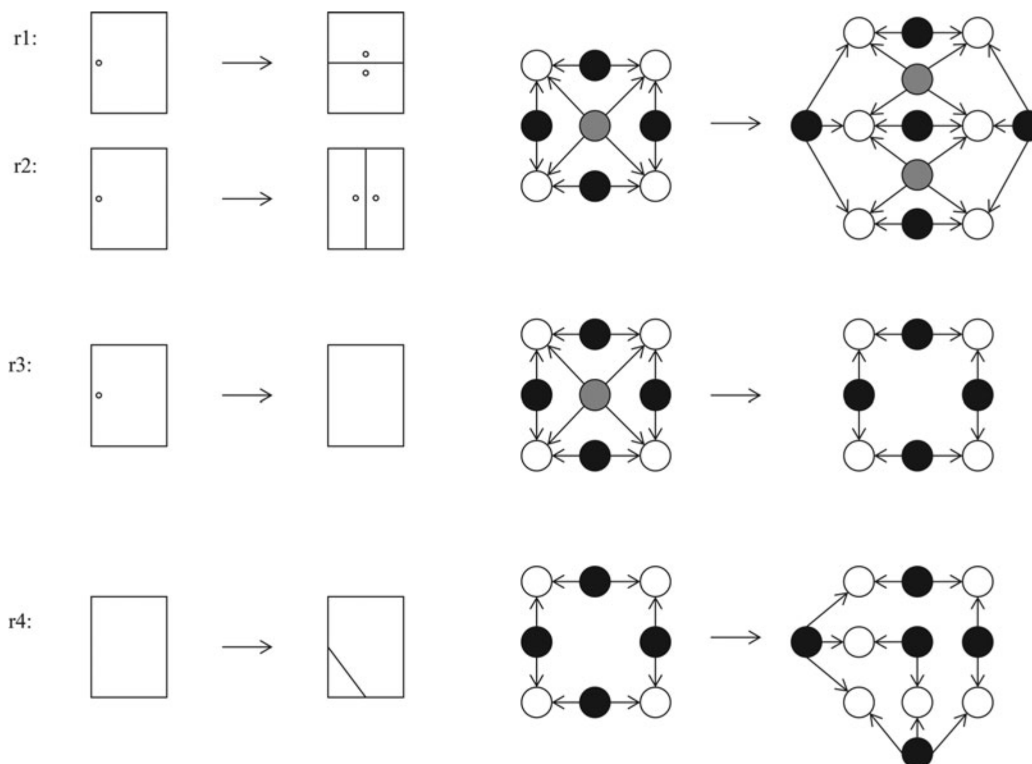

Fig. 8. Malagueira composition rules: rules for perpendicular dissection of rectangles (rules 1 and 2), rule for deleting markers (rule 3), and rule for diagonal dissection of rectangles (rule 4). The graph transformation rules (right) consist of vertex nodes (white), edge nodes (black), and label nodes (gray).

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

rule application is driven by (parametric) subshape matching, and shapes are represented as graphs, subgraph isomorphism detection is one of the most runtime-intensive steps in the process. As an example, we measured the generation time for the dissection rule of Figure 8 at different steps in a generation process. This test was performed on an Intel Core 2 Quad 3-GHz processor with 4 GB of RAM and Windows 7 (64-bit). The results of the test are shown in Figure 9. The generation time increased exponentially as the number of graphs objects (nodes and edges) became larger, but remained reasonably low (<2 s) for graphs that consisted of up to approximately 600 to 800 graph objects. Using the graph representation described here, 21 nodes and edges are required to model a rectangular space. Therefore, 600 graph objects would correspond to a medium-size floor plan with approximately 30 spaces. In benchmarks of similar graph-based implementations (Grasl, 2013), it is concluded that the generation time also depends on the complexity of the pattern graph of rules: a more constrained pattern graph results in a faster generation time. For larger floor plans that have more than 30 spaces, it is possible to use more compact graph representations or more constrained graph rules in order to keep the generation time sufficiently low.

6.2. Frank Lloyd Wright prairie house grammar

In this case study, we implemented a part of the Frank Lloyd Wright prairie house grammar, developed by Koning (1981). The original grammar was developed on paper, and several computer implementations of this grammar also exist, for example, Granadeiro et al. (2013). The grammar is a three-dimensional parametric shape grammar, in which rule application is driven by markers (V_M). In the scope of this article, the new grammar contains a number of rules equivalent to the rules of the original prairie house grammar by Koning and Einzenberg (1981). In particular, the implemented grammar contains rules to: create a fireplace (rule 1), add a living zone (rules 2–4), complete the core unit (rule 5), add obligatory extensions (rules 6–7), and assign function zones (rules 8–10). These rules generate all the basic compositions underlying prairie-style houses. The other rules from the original grammar (e.g., ornamentation, adding porches, interior details, etc.) are not considered within the scope of this article. The new rules are visualized in Figure 10. In this figure, living zones are indicated in white, service zones in gray, and the obligatory extensions in black.

There are several differences between the original grammar and the new grammar. First, some of the original rules have been slightly modified in order to make them more amenable to computer implementation. For example, rule 1 does not distinguish between a single-hearth and a double-hearth fireplace. Therefore, this rule replaces rules 1 and 2 from the original shape grammar (indicated between brackets in Fig. 10). Second, the new grammar is two-dimensional, because the floor height typically remains constant in Frank Lloyd Wright's prairie houses. Third, rule application is primarily driven by (parametric) subshape detection. However, in certain cases, the use of markers is necessary to avoid redundancy of isomorphic results and to include intentional information (e.g., the labels in rule 5 specify that the obligatory extensions can only be added once at each side). Fourth, the rules are defined as graph transformation rules, because of the underlying AGG environment. The implemented graph transformation rules are parametric rules in the sense that several linear transformations are allowed in order to match the rules to the shapes (translation, mirroring, and proportional scaling). We have chosen to demonstrate the visual representation rather than the graph representation of the rules in Figure 10, because the graph-theoretic representation of shapes is not the main focus of this article. The basic elements of the shapes and the relations between the basic elements are de-

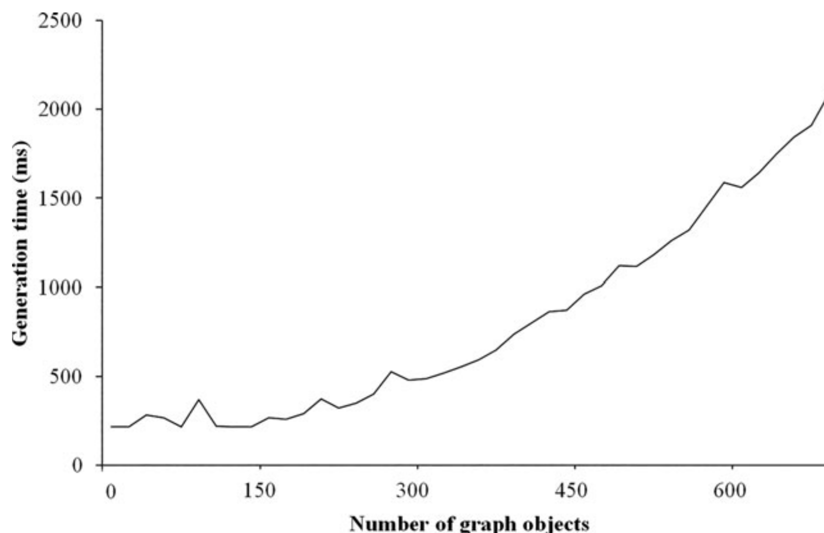

Fig. 9. The generation time of shapes increases exponentially as the number of graph objects (nodes and edges) becomes larger.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

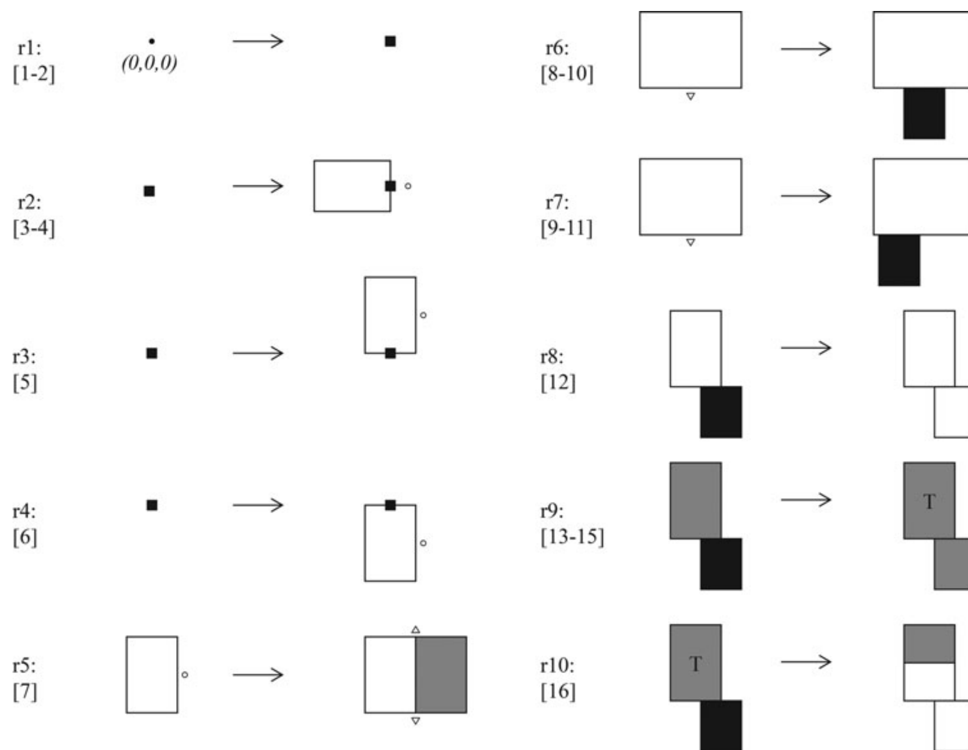

Fig. 10. Newly developed grammar rules: locating the fireplace (1), adding a living zone (2–4), completing the core unit (5), adding obligatory extensions (6–7), and assigning function zones (8–10). Living zones are indicated in white, service zones in gray, and the obligatory extensions in black. Numbers between square brackets refer to the original rule numbers in the grammar of Koning and Eizenberg (1981).

fined similarly to the rules shown in Figure 8. In addition, label nodes are used to specify the function of the spaces in the floor plan (living, service, or extension).

A derivation of the Frank Lloyd Wright prairie house grammar, defined above, is given in Figure 11. The rules that are used are indicated between parentheses. It is possible to store the current derivation of "*adding a living zone and completing the core unit*" (indicated in gray) to the database. Therefore, it is possible to recall this derivation and the generated shapes in a new context. For example, the derivation shown in Figure 12 starts from a shape taken from the database (indicated in gray). Figure 12 demonstrates a part of the explicit design space, including one shape that corresponds to a design from the catalog of basic compositions underlying prairie houses defined by Koning and Eizenberg (1981). A total of 36 compositions can be generated using the specified grammar, whereas the original grammar is able to generate 89 compositions. In order to generate all

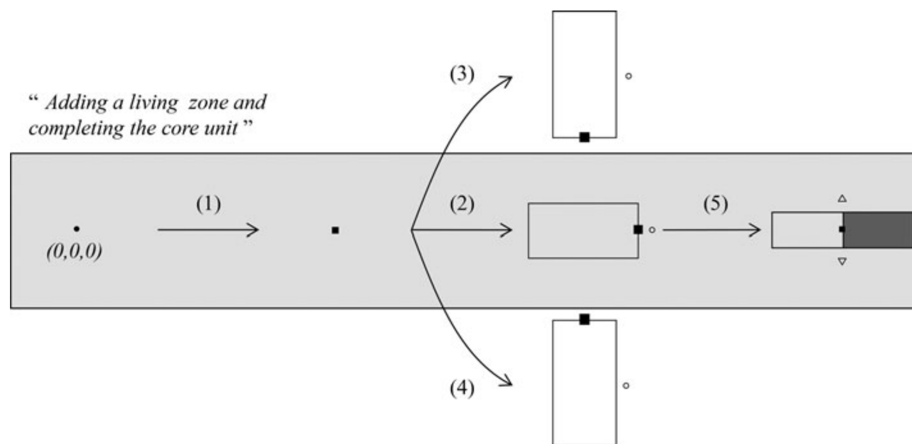

Fig. 11. Example of a derivation of the prairie house grammar. The current derivation (gray) can be stored in the database.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

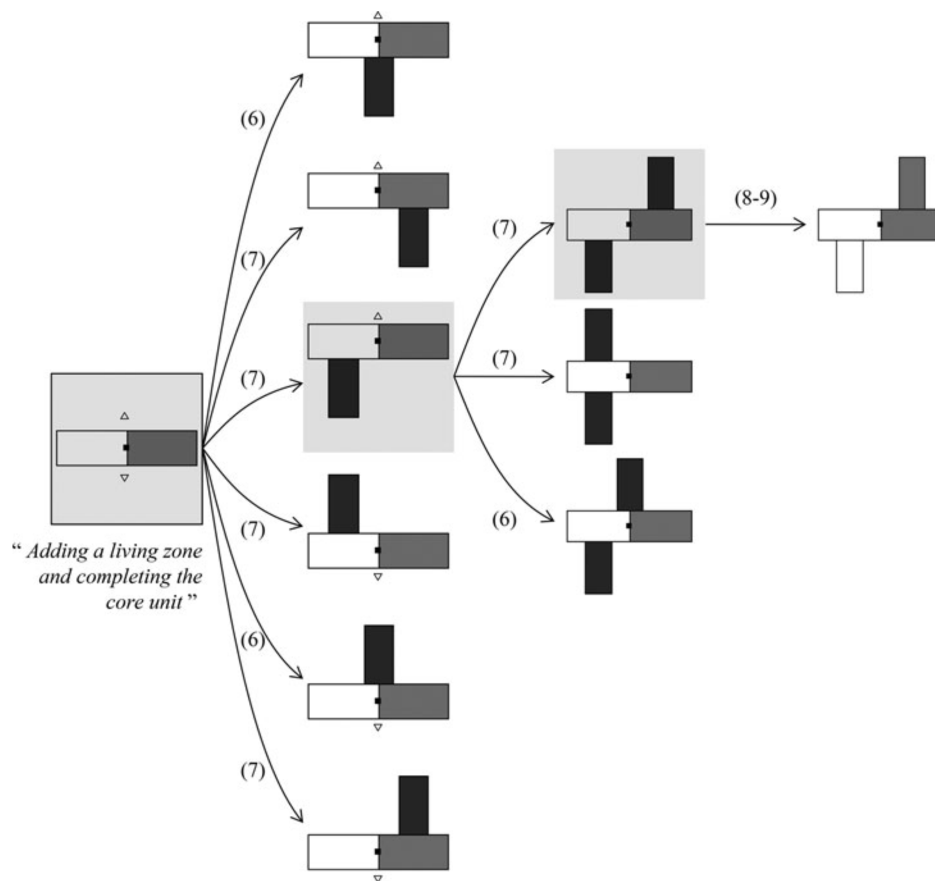

Fig. 12. Example of a derivation of the prairie house grammar, starting from a design state pulled from the database.

compositions, it is necessary to navigate back into the explicit design space, thus making new areas available for future exploration. This example demonstrates the value of representing the explicit design space as a whole. While intermediate design states may not have intentional value, they allow access to specific parts of the design space that remained inaccessible before.

7. DISCUSSION

In the Introduction of this article, we identified several key properties for supporting design space exploration that are typically not of central concern to current shape grammar implementations: the ability to represent design spaces, and the ability to explore these design spaces, including several amplification strategies. These strategies include, first, the representation of the explicit design in a structured and visual way, enabling designers to satisfice design alternatives against goal criteria. Second, back navigation is an important amplification strategy because it makes new areas of the design space available for future exploration. Third, the incorporation of an additional database system has shown to be both feasible and valuable.

Such a database system, as demonstrated by the Frank Lloyd Wright prairie house grammar case study, enables designers to recall shapes or specific shape derivations that are generated in a different context or project, such as "*adding a living zone and completing the core unit*" in Figure 11. However, the case study has also indicated that several aspects need further elaboration in order to be useful in more realistic and complex design situations. On the one hand, the use of equivalence classes rather than individual shapes could not only reduce the size of the explicit design space but also enhance the possibility to recall design states based on a measure of equivalence or similarity. This would also imply a representation of the explicit design space as a network rather than a tree. On the other hand, further elaboration of the database system shown in Figure 7 could allow the designer to specify more complex (semantic) search queries and could enhance the possibility to recall design states in an improved way. A possible way to enable recall in the design process is the use of graph databases (such as triplestores) or other types of *noSQL* databases. While search queries in relational databases are based on metadata (grammar, project, etc.), graph databases allow more complex semantic search queries based on the graph structures in the database. The potential use of elaborated database systems is part of our current ongoing research.

The case study of the Malagueira grammar demonstrates the feasibility of the proposed approach in terms of the system's responsiveness and the usability of the navigation method. The case study of the Frank Lloyd Wright prairie

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

house grammar demonstrates how designers can interact with grammars using the proposed design space exploration features. In particular, the interactive and visual user interface allows designers to engage in a dialogue or "conversation" with the design space. As described in the Introduction of this article, the design space exploration task consists of three steps: representing a design space, navigating a design space, and evaluating or satisficing design alternatives against goal criteria. Based on the observations made during the Frank Lloyd Wright prairie house case study, our proposed approach can be interpreted as a semiautomatic approach, in which design alternatives are generated automatically by the information system, but evaluation and selection is performed by the designer. In contrast to fully automated approaches, such information systems guide exploration toward specified goal criteria by proposing different paths and alternatives. Therefore, such information systems can be devised as specialized agents that support or even amplify an exploration process, resembling more closely an intelligent, agentlike role for the computer in design. If evaluation mechanisms for quantifiable aspects of designs, such as building performance, are incorporated into the system, evaluation could be performed autonomously by the computer. This would require further investigation in future research.

Based on the overview of existing shape grammar implementations in this article, the proposed design space exploration features are typically not supported in current implementations. Therefore, the approach proposed in this article stands out as a novel approach that can contribute to the current state of the art of shape grammar implementations. Because our proposed approach does not depend on a specific underlying framework, the described functionality can also be added to other existing generative design systems and shape grammar implementations. For example, the aspects of backtracking and recall through a database system can be designed as add-ons to a more general graph-theoretic implementation that supports emergence and three-dimensional shapes (Grasl, 2013). Therefore, the approach proposed in this paper can be extended with several functionalities in future work, including simultaneous manipulation of design states and the possibility to manually intervene in the exploration process by modifying design states with no regard for rules. The former would involve the use of shared variables to apply changes in one design state to other design states. The latter would enable the designer to make shortcuts in the design space, allowing "on the spot" experimentation. Such functionality is already implemented in GRAPE (Grasl, 2013) by allowing designers to switch from a grammar mode to a manual drafting mode.

8. CONCLUSION

The work presented in this article demonstrates an alternative shape grammar implementation approach that is able to support design space exploration in a visual and interactive way. The key properties of this implementation approach, as described in the Introduction of this article, are the representation of the design space and the ability to explore this design space. Because shape grammars do not inherently keep track of the explicit design space, it is necessary to extend the shape grammar formalism with relations between a parent shape and the generated child shapes, resulting in a tree structure. As a result, it is possible to represent the explicit design space and allow several design space exploration amplification strategies: generation of alternatives, interactive navigation, backup of design states and paths, and recall of these design states and paths. A proof-of-concept software system has been developed in which the following features are provided: representation of the design space as a visual whole, representation of design states in both a visual and symbolic manner, and support of exploration through the amplification strategies discussed above. The feasibility of the proposed approach and proof-of-concept software system has been shown through case studies using two existing shape grammars from the architectural domain. Future research will focus on additional design space representation possibilities, shape equivalence or similarity, and enhanced database systems.

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).

REFERENCES

Akin, O. (2006). The whittled design space. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 20(2)*, 83–88.

Charbonneau, N., & Tidafi, T. (2013). Enabling the architectural designer to move within a graph of interconnected decisions: a case study dealing with a parametric object. *International Journal of Business, Humanities and Technology 3(1)*, 42–51.

Chase, S. (2002). A model for user interaction in grammar-based design systems. *Automation in Construction 11(2)*, 161–172.

Cross, N. (1982). Designerly ways of knowing. *Design Studies 3(4)*, 221–227.

Duarte, J.P. (2005). A discursive grammar for customizing mass housing: the case of Siza's houses at Malagueira. *Automation in Construction 14(2)*, 265–275.

Fitzhorn, P. (1990). Formal graph language of shape. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 4(3)*, 151–163.

Gero, J.S. (1994). Toward a model of exploration in computer-aided-design. In *Formal Design Methods for CAD* (Gero, J.S., & Tyugu, E., Eds.), pp. 315–336. Amsterdam: North–Holland.

Gero, J.S., & Kazakov, V.A. (1996). An exploration-based evolutionary model of a generative design process. *Computer-Aided Civil and Infrastructure Engineering 11(3)*, 211–218.

Gero, J.S., Neville, D., & Radford, A.D. (1983). Energy in context: a multicriteria model for building design. *Building and Environment 18(3)*, 99–107.

Geyer, P. (2008). Multidisciplinary grammars supporting design optimization of buildings. *Research in Engineering Design 18(4)*, 197–216.

Gips, J. (1999). *Computer implementation of shape grammars*. Workshop on Shape Computation, MIT.

Goel, A.K., & Craw, S. (2006). Design, innovation and case-based reasoning. *Knowledge Engineering Review 20(3)*, 271–276.

Goldschmidt, G. (2005). How good are good ideas? Correlates of design creativity. *Design Studies 26(6)*, 593–611.

Goldschmidt, G. (2006). Quo vadis, design space explorer? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 20(2)*, 105–111.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

Granadeiro, V., Duarte, J., Correia, R., & Vitor, M.L. (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., & Economou, A. (2012). Transformational Palladians. *Environment and Planning B 39(1)*, 83–95.

Grasl, T., & Economou, A. (2013). From topologies to shapes: parametric shape grammars implemented by graphs. *Environment and Planning B 40(5)*, 905–922.

Grasl, T., & Economou, A. (2014). Toward controlled grammars—approaches to automating rule selection for shape grammars. *Proc. 32th eCAADe Conf.* (Thompson, E.M., Ed.), pp. 357–363, Newcastle upon Tyne, UK, September 12–14.

Heisserman, J. (1994). Generative geometric design. *IEEE Computer Graphics and Applications 14(2)*, 37–45.

Hoisl, F., & Shea, K. (2011). An interactive, visual approach to developing and applying parametric three-dimensional spatial grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 25(4)*, 333–356.

Jones, J.C., & Thornley, D. (1962). *The Conference on Design Methods. Proc. Conf. Systematic and Intuitive Methods in Engineering, Industrial Design, Architecture and Communications*. London: Pergamon Press.

Knight, T.W. (2003). Computing with emergence. *Environment and Planning B 30(1)*, 125–155.

Koning, H., & Eizenberg, J. (1981). The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B 8(3)*, 295–323.

Krishnamurti, R. (2006). Explicit design space? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 20(2)*, 95–103.

Lawson, B. (2005). Oracles, draughtsmen, and agents: the nature of knowledge and creativity in design and the role of IT. *Automation in Construction 14(3)*, 383–391.

Maher, M., & Poon, J. (1996). Modelling design exploration as co-evolution. *Microcomputers in Civil Engineering 11(3)*, 195–209.

McKay, A., Chase, S., Shea, K., & Chau, H.H. (2012). Spatial grammar implementation: from theory to useable software. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 26(2)*, 143–159.

Papanikolaou, M., & Tunçer, B. (1999). The Fake.Space experience—exploring new spaces. *Proc. 17th eCAADe Conf.*, pp. 395–402. Liverpool: University of Liverpool.

Radford, A.D., & Gero, J.S. (1980). On optimization in computer aided architectural design. *Building and Environment 15(2)*, 73–80.

Radford, A.D., & Gero, J.S. (1988). *Design by Optimization in Architecture, Building, and Construction*. New York: Van Nostrand Reinhold.

Rittel, H., & Webber, M. (1973). Dilemmas in a general theory of planning. *Policy Science 4(2)*, 155–169.

Russel, S., & Norvig, P. (2010). *Artificial Intelligence—A Modern Approach*. New York: Prentice–Hall.

Schaefer, J., & Rudolph, S. (2005). Satellite design by design grammars. *Aerospace Science and Technology 9(1)*, 81–91.

Schön, D. (1983). *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books.

Shea, K., & Cagan, J. (1999). Languages and semantics of grammatical discrete structures. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 13(4)*, 241–251.

Simon, H.A. (1956). Rational choice and the structure of the environment. *Psychological Review 63(2)*, 129–138.

Simon, H.A. (1957). *Models of Man: Social and Rational*. New York: Wiley.

Simon, H.A. (1973). The structure of ill-structured problems. *Artificial Intelligence 4(3–4)*, 181–201.

Stiny, G. (2007). *Shape—Talking About Seeing and Doing*. Cambridge, MA: MIT Press.

Stiny, G., & Gips, J. (1972). Shape grammars and the generative specification of painting and sculpture. *Information Processing 71*, 1460–1465.

Stiny, G., & Mitchell, W.J. (1978). The Palladian grammar. *Environment and Planning B 5*, 5–18.

Taentzer, G., & Rudolf, C. (1998). AGG-approach: language and tool environment. In *Graph Grammar Handbook 2: Specification and Programming* (Rozenberg, G., Ed.). Singapore: World Scientific.

Tapia, M.A. (1999). A visual implementation of a shape grammar system. *Environment and Planning B 26(1)*, 59–73.

Tidafi, T., Charbonneau, N., & Araghi, S.K. (2011). Backtracking decisions within a design process: a way of enhancing the designers thought process and creativity. *Proc. 14th Int. Conf. Computer Aided Architectural Design* (Pierre Leclercq, A.H., & Martin, G., Ed.), pp. 573–587, Liege, Belgium, July 4–8.

Trescak, T., Esteva, M., & Rodriguez, I. (2012). A shape grammar interpreter for rectilinear forms. *Computer-Aided Design 44(7)*, 657–670.

Turrin, M., von Buelow, P., & Stouffs, R. (2011). Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms. *Advanced Engineering Informatics 25(4)*, 656–675.

Woodbury, R., & Burrow, A. (2006). Whither design space? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 20(2)*, 63–82.

Yue, K., & Krishnamurti, R. (2014). A paradigm for interpreting tractable shape grammars. *Environment and Planning B 41(1)*, 110–137.

Tiemen Strobbe is a PhD Researcher in the Ugent SmartLab research group in the Department of Architecture and Urban Planning and the Department of Electronics and Information Systems of Ghent University. He graduated as an engineer–architect from Ghent University, Belgium, with a Master dissertation on the applicability of parametric design strategies. His interests and current PhD research focus on design space exploration, generative design, shape grammars, and the application of information technology in architectural design in general.

Pieter Pauwels is a Postdoctoral Researcher in the Department of Architecture and Urban Planning at Ghent University. He holds Master's and PhD degrees (2012) in engineering–architecture, both obtained at Ghent University. During his PhD research, he investigated how information system support can be provided for architectural design thinking. Pieter was previously a Postdoctoral Researcher at the Institute for Logic, Language and Computation in the University of Amsterdam. Dr. Pauwels is now working full-time on topics affiliated with building information modeling, linked building data, and linked data in architecture and construction.

Ruben Verstraeten is as an Assistant Professor in the Department of Architecture and Urban Planning at Ghent University. He graduated as an engineer–architect from Ghent University. His PhD dissertation was focused on the automated compliance checking mechanisms of architectural designs. Dr. Verstraeten teaches several courses in computational design, including three-dimensional modeling, parametric design, and digital fabrication.

Ronald De Meyer is a Senior Lecturer in the Department of Architecture and Urban Planning of Ghent University and a Lecturer in the Department of Architecture and Design of Hasselt University. He graduated from the Hoger Architectuurinstituut van het Rijk, Antwerp, and received his PhD with a dissertation on the development of the 19th-century Antwerp town district Het Zuid from Leuven University. His research involves 19th- and 20th-century construction history, more specifically, the role of concrete and iron structures in Belgium, and the intelligent deployment of ICT technology within architectural design.

Jan Van Campenhout is a member of the Department of Electronics and Information Systems of the Faculty of

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press

Engineering at Ghent University. He received a degree in electromechanical engineering from the University of Ghent and MSEE and PhD degrees from Stanford University. Dr. Van Campenhout's research interests include the study and implementation of various forms of parallelism in information processing systems, currently focused on the modeling and design of short-range parallel optical interconnects from a systems perspective. In recent years, his interests also include the computer support of the design methodology in other areas, such as architectural design exploration.

https://doi.org/10.1017/S0890060415000475 Published online by Cambridge University Press