Multiagent Shape Grammar Implementation: Automatically Generating Form Concepts According to a Preference Function

Seth Orsborn
Department of Interdisciplinary Engineering,
Missouri University of Science and Technology,
Rolla, MO 65409
e-mail: orsborns@mst.edu

Jonathan Cagan
Department of Mechanical Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213
e-mail: cagan@cmu.edu

In new product development, quickly generating many product form concepts that a potential consumer prefers is a challenge. This paper presents the inaugural multiagent shape grammar implementation (MASGI) to automatically generate product form designs according to a preference function that can represent designer or consumer design preference. Additionally, the multiagent system creates a flexible shape grammar implementation that enables modifications to the shape grammar as the form design space changes. The method is composed of three subprocesses: a shape grammar interpreter that implements the shape grammar, an agent system that chooses which shape grammar rules to implement and the parametric design choices according to a preference function, and a preference investigator that determines the preference function, which constraints the automated form design process. [DOI: 10.1115/1.4000449]

Keywords: shape grammar, multiagent system, preference function, automated product form design

1 Introduction

In 2006, Orsborn and Cagan [1] first used shape grammars to explore various nonobvious forms within vehicle classes and then created novel forms of crossover vehicles. But it is not enough to create novel form designs. These form designs should match stakeholder preferences, be they designers or consumers. It was then demonstrated that form preference can be quantified in a utility function, which could guide the design process to create forms that corresponded to a derived consumer preference [2]. This whole method, while valuable, was limited in that the decision making and analysis was conducted by humans. The research presented in this paper leverages the previous work in an automated multiagent system (MAS): a multiagent shape grammar implementation (MASGI). MASGI represents a form design space with a shape grammar, uses design of experiments to determine preference within the design space, and then automatically generates appropriate form design concepts through a MAS.

This paper first provides a background on preference functions, shape grammars, and software agents. MASGI is next introduced with an overview, and then each of the three subsections is discussed in detail. The MASGI methodology is applicable to any grammar-based form design language, but a case study will be used throughout the paper as a running example with the software *agents* implementing a vehicle *shape grammar* to automatically generate form designs according to consumer aesthetic form *preference*.

1.1 Preference. Computational representations of products enable the automatic generation of designs. Not all designs are valuable, but only the designs that meet the required criteria. When considering product form concepts, the "good" designs are those that match consumer aesthetic preferences. The aesthetic form of a product influences consumer purchasing [3] and is a factor in the success, or failure, of consumer products [4]. In an attempt to determine which forms are preferred by the consumer, product designers generate product form concepts and test them against potential consumers through tools such as focus groups and consumer surveys. A utility function is a specific type of preference function used by economists to describe a person's utility, a measure of happiness, or satisfaction gained by using a certain good or service [5]. Utility functions were used in engineering design to quantify consumer and designer preference [6,7]. It was demonstrated that consumer form preference can be summarized in a utility function [8]. This utility function can then be used as a preference function to constrain automated concept design generation. In the concept generation phase of new product development, many form concepts need to be quickly generated. In this instance, they can be generated according to a derived utility function, and thus, are likely to be preferred by the potential customer. This paper shows that a shape grammar implemented in a MAS can quickly and automatically create many concept designs within a form design space while adhering to the constraints of a preference function.

1.2 Shape Grammars. For automated concept generation, it was shown that a large number of divergent product form concept designs within a specified design space can be easily created through a parametric manipulation of a morphable model [9]. By contrast, shape grammars enable form design exploration through rule choices in addition to parametric variations, potentially in new or creative ways. Shape grammars were used in a variety of product design applications such as: architectural floor plans [10], Chinese ice-ray lattice windows [11], coffee makers [12], Harley–Davidson branding [13], the Buick vehicle brand [14], and vehicle product classes [1].

Shape grammars are a geometry based production system created by taking a sample of the whole, for which one is trying to write a language [15]. From this sample, a vocabulary of shapes

Contributed by the Design Theory and Methodology Committee of ASME for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received January 25, 2008; final manuscript received August 27, 2009; published online November 12, 2009. Editor: Panos Y. Papalambros.

can be written that represents all the basic forms of that sample. By defining the spatial relationships between those forms and how the forms are related to each other, shape rules can be written. A shape rule consists of a left and right side. If the shape in the left side matches a shape in a drawing, then the rule can be applied, and the matching shape changes to match the right side of the rule. The shape rules allow the addition and subtraction of shapes, which in the end are perceived as shape modifications. There are multiple ways to drive a shape grammar. The simplest is to utilize labels and markers. Labels are typically alphanumeric characters attached to a shape. Markers are similar to labels, but are symbols attached to a shape. In both cases, they are used to limit which rules can and cannot be applied to the shape. If an appropriate marker or label exists, then a rule can be applied.

Since shape grammars were first invented in the 1970s [15], computational implementation was a challenge. The fundamental problems related to shape arithmetic [16], the complexities in computationally implementing straight line shape grammars [17–19], and issues with curve recognition and implementation [20,21] were all addressed. Each of these were limited by using a traditional software architecture to implement a fixed shape grammar, which constrained future modifications to the prescribed grammar and thereby the design space. The introduction of a shape grammar interpreter enabled addition and subtraction of shape grammar rules [21]. The shape grammar interpreter required a *human agent* to make the shape grammar rule choices, while *software agents* made parametric choices based upon shape optimization [22]. Implementing a shape grammar by integrating it with software agents provides the flexibility of a shape grammar interpreter and the speed of automated design generation, previously not available together. Subdividing tasks and assigning them to different agents facilitates modification of the shape grammar, and thus, the form design space. The goal for this work is a dynamic shape grammar interpreter architecture that enables automatic form design generation. The methodology utilizes an iterative process to filter and guide form designs to prune and optimize the number of potential solutions. A vehicle form shape grammar is used throughout this paper as a demonstration, but the method is abstract enough that any shape grammar could be used instead.

1.3 Software Agents. The research presented in this paper is built upon the software architecture often referred to as MASs or simply agents. Software agents provide a discrete architecture that facilitates future modifications. Unlike more common optimization techniques (e.g., simulated annealing or genetic algorithms), this discrete architecture enables changes to be easily made, e.g., adding or subtracting agents, without having to modify other sections of the code. The field of software agents is incredibly diverse. Recent research included programming languages that simplify agent interactions with each other and their environment [23], and simplifying agent interactions by creating communication structures similar to human conversations [24]. Agents are proving their effectiveness in traditional optimization problems like flexible job shop scheduling [25], and are even being used in conjunction with game theory to accomplish complex problems such as traffic light coordination [26].

The intention of this research is not to further the work done in MASs, but to utilize existing software agent concepts in a new way: shape grammar implementation. These software agents implement a shape grammar to automatically create form designs according to a preference function. If this preference function summarizes a consumer's preference for product aesthetic form, as is the case in this work, the generated designs are assured to be preferred by the consumer.

It is quite common to utilize existing agent modeling tools such as SeSAm [27] when conducting agent research. Existing agent modeling tools were found to be insufficient in this work, due to the unique interactions between the agents, the shape grammar interpreter, and the preference investigator. The software architecture introduced in this paper was created specifically for this research. The genealogy of this architecture starts with A-Teams [28,29], where agent optimization is detailed and contrasted with respect to other more common optimization techniques such as simulated annealing and genetic algorithms [30]. More recently, software agents were used to optimize the design parameters in an engineering shape grammar [22], though not to implement the shape grammar. A-Design also demonstrated the effectiveness of agent architecture in design optimization [31], which was further improved by including agent interactions [32]. In each of these, the software architecture is composed of a manager agent and several task-specific agents. The manager oversees the design process and maintains a library of all necessary information such as constraints, current designs, completed designs, and allowable agent interactions. The task-specific agents are each responsible for a single aspect of the design process and must report their optimization decisions to the manager and other agents, as allowed.

One approach to using MASs for design exploration is for design team simulation. The MAS represents each design constraint with a single agent. Each agent can be responsible for product design and work cooperatively [32], or a single agent can be responsible for one part of the design process [33]. The MAS architecture presented in this paper establishes each agent as responsible for a particular product form feature, dividing the design process into discrete product choices. These *collaborative agents* enable the MAS to function beyond the sum of the individual members [34]. This system model is similar to industry practice where one agent, e.g., an engineering designer, may be responsible for a particular product feature, e.g., the headlight of a vehicle. Section 2 will detail how the agent definitions and construct above will be used, uniquely, to create MASGI.

2 MASGI Structure

Prior to this research, computational shape grammar implementation required manual rule choices. In this work, we demonstrate that a MAS can automatically implement a shape grammar, creating form concept designs through rule choices without human input. This section introduces the relationship between the MAS structure and the form design space, and then some basic agent properties. The MASGI will then be described, generally at first and then in detail, in conjunction with an example of its implementation related to a case study of the vehicle form design.

The MASGI is intended to reflect a design team [32], and therefore should also match the design space representation. Agents are used so that future modifications to the product form design space can be easily implemented through the addition or subtraction of agents or the design features they implement.

Any product or system can be represented with a hierarchy. Specific to the example, the representation of a complex form with lesser features is called atomization [35]. For example, the form of an automobile can be separated into product characteristics like the headlight, grill, and fender. Then each vehicle characteristic can be subdivided into a set of curves that best represent that characteristic. The relationship between these characteristic curves can then be summarized with distinguishing atomic attributes such as the vertical height of the grill. These atomic attributes can be modified to change the overall form of the vehicle: the gestalt, while the consumer sees only a change in product form.

The MASGI structure maps directly to the hierarchical atomic sequence. Figure 1 shows both sequences, the atomic sequence on the left, and the MASGI sequence on the right. The overall product form and the design progression is handled by a single manager agent, similar to the manager of a design team. Each one of the many product characteristics is assigned a characteristic agent, similar to an individual designer responsible for a certain aspect of a new product. The design space is represented with characteristic curves. The characteristic agent is allowed to access certain shape

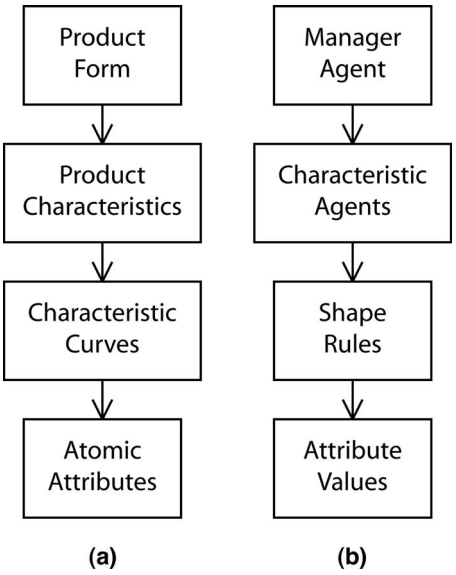

Fig. 1 Atomic and MASGI sequences

rules from a shape grammar that create the characteristic curves. When the curves are created, parametric attribute values are selected for each of the atomic attributes.

Herein lies the advantage of using a MAS to implement a shape grammar: *flexibility*. Each step in the atomic sequence (Fig. 1(*a*)) can be mapped to a specific component in the MASGI sequence (Fig. 1(*b*)). Therefore, future updates and modifications within the atomic sequence do not require a rewrite of the algorithm, but can be addressed by adding (or subtracting) components in the MASGI sequence. For example, more product characteristics can be added to the design space through the addition of more characteristic agents without modifying the base code. With a MAS architecture, future adjustments at any level can be made easily.

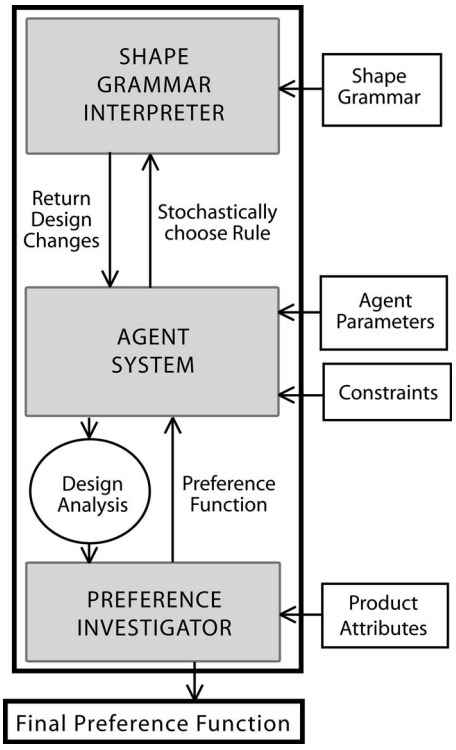

Fig. 2 Method overview

This architecture is applicable to any product (not just vehicles) as long as the form of the product is broken into its key characteristics, and a shape grammar is created to explore the product form design space.

3 General Method

The overall method (Fig. 2) is composed of three main sections: the shape grammar interpreter that seeks out the left hand side of shape grammar rules and transforms to the right hand side, the agent system that chooses shape grammar rules to apply and automatically generates and optimizes product designs, and the preference investigator that seeks out and mathematically represents form design preferences. Each part is crucial to the implementation of the shape grammar, the facilitation of modifications to the shape grammar, and the automatic design of product forms that match consumer preferences. A MAS keeps each process separate, which facilitates future modifications and updates. This constraint requires a consistent communication protocol between the different sections.

3.1 Shape Grammar. Beyond the three main process sections, additional inputs are required. The shape grammar needs to be created for the design space. Traditionally, and in this instantiation, this is done manually. Once the rule set is created, it is passed to the shape grammar interpreter.

The shape grammar implemented for the case study was created to demonstrate the features of this method. The chosen vehicle characteristics are complex enough to be visually interesting yet simple enough to represent with atomic attributes. The headlight, grill, and related features (darkened lines in Fig. 3(*a*)) are represented with the atomic attributes in Fig. 3(*b*). These few vehicle characteristics provide a large space for exploration with many varied results.

The vehicle shape grammar in the Appendix is composed of an initial shape, 14 shape insertion rules, and 6 shape modification rules. The initial shape is simply the coordinate axes in Fig. 3. The 14 shape insertion rules each insert all or part of a vehicle characteristic in both the front and side view. Each shape insertion rule is assigned to its characteristic agent: rules 1–2 to the ground, rules 5 and 6 to the hood, rules 7–12 to the grill, and rule 13 to the headlight. Rules 4 and 14 exist to complete the form design and are assigned to the manager for this instantiation. Each shape insertion rule creates a set of four-control-point Bezier curves, which represent that product characteristic. The characteristic

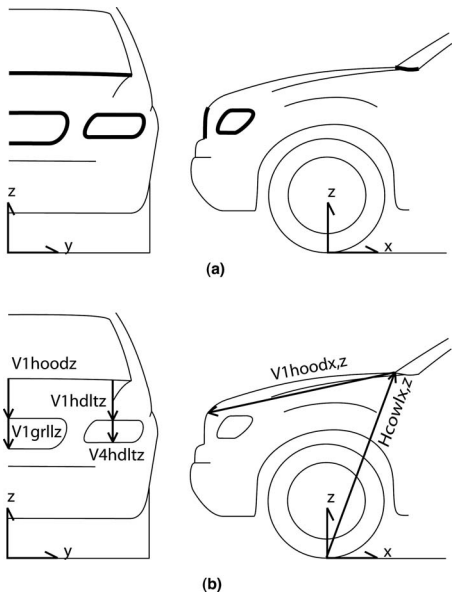

Fig. 3 SUV characteristics and attributes

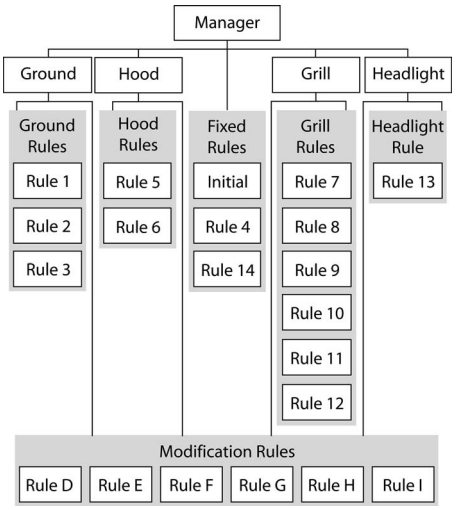

Fig. 4 Characteristics and rules tree for implementation

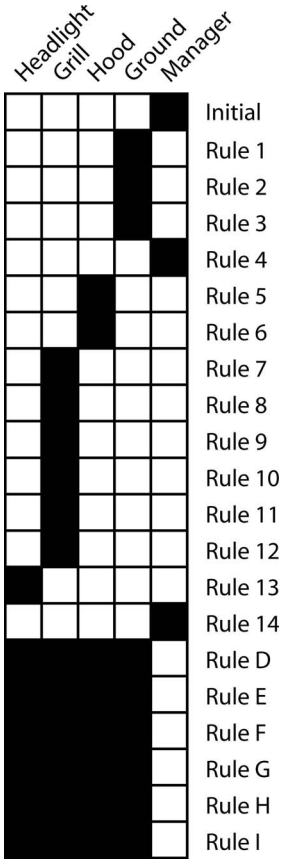

Fig. 5 Agent-rule matrix

agent chooses a value for a particular attribute (Fig. 3(*b*)) according to its objective function. The actual shape of the curves (the location of their control points) are then calculated, based upon the attribute value. Finally, the shape modification rules can be accessed by any of the characteristic agents to make further changes to their shapes, based upon their objective functions.

3.2 Agent Parameters. Figure 4 demonstrates the relationship between the manager agent and the characteristic agents, just as the MASGI sequence introduced in Fig. 1. The manager oversees the design process by coordinating the four characteristic agents: ground, hood, grill, and headlight. Additionally, in regards to this study, the manager is responsible for characteristics and shapes that facilitate the form design called fixed rules. Each of the characteristic agents make design choices by implementing the shape grammar rules specific to their characteristic and can make design modifications by implementing the modification rules.

The relationship between the agents and the shape grammar rules is the agent parameters in Fig. 2, which is an input to the method. The agent-rule matrix (Fig. 5) communicates the content of the MASGI sequence (Fig. 4) to the manager agent by indicating which agents can apply which grammar rules. For example, in Fig. 5, ground can only apply rule 1, but all the agents, except the manager, can apply rule D. Each agent has very specific rules based upon their product characteristic. Depending on the product form design space and how the shape grammar is written, there may be overlaps between agents and rules.

3.3 Constraints. There are two types of constraints passed in to the agent system. The first type constrains the form design choices. There are parametric constraints for each attribute, which can be derived from sample data that is also used to create the grammar rules. For example, the vehicle shape grammar was created based upon a product sample of SUVs from the 2003 model year. The parametric constraints from the product sample, such as the largest and smallest wheel radii, are used as input constraints. These constraints are not necessarily fixed and may be modified by the designer for the sake of design exploration.

The second type constrains the design process. Information passed in to the manager agent include the number of designs requested, optimization constraints, and the results from the design of experiments. The preference function is determined through an initial design analysis and a verification design analysis. The design of experiments is used as the input for creating the initial design analysis. In the case study, the design analysis is a survey administered to potential consumers. The respondent results from the initial survey are used to determine the weights for the preference function. The experiment may be designed by existing software packages or it could be random. Future research will incorporate the design of experiments within the preference investigator. Likewise, an additional agent could be created just for this task. In this instantiation, the design of experiments is created using SAS, a business analytics software, and is then input to the manager agent as a data file.

3.4 Attributes. The final input to the method are the attributes used to describe the product form. The information transferred is the shape information used to create the atomic attributes (Figs. 1 and 3). In this instance, the atomic attributes determine the initial form of the preference function, which communicates to the agents which curves should be explored.

4 Method Details

4.1 Agent System. The agent system (Fig. 6) is the heart of the method, but is supported by and integrated with the shape grammar interpreter and the preference investigator. The agents explore the design space, choose shape grammar rules, and create form designs based upon the utility function.

There are four specific attributes for a multiagent design system, which are all taken into account in this MASGI architecture [31]. First, the system is designed to create and improve upon form design alternatives. This is accomplished through an internal iteration that allows the selection of shape modification rules. Second, the shape grammar is the representation of the conceptual design space, constrained by the preference function. This representation is understood by the agents and is the framework in which they automatically create form design concepts. Third, the preference function serves as a means for multiobjective decision making, in that each characteristic agent is designed to optimize its individual preference functions for the attributes in its shape

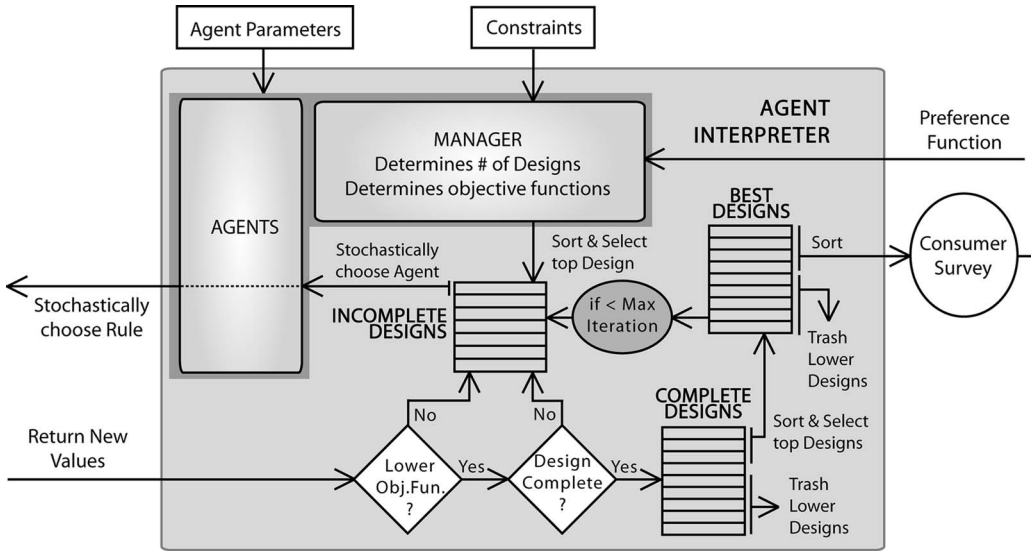

Fig. 6 Agent system process

characteristics. The MASGI allows for future modifications, not only to the agents and the shape grammar, but also to the preference function itself. Fourth, if the design preference (either consumer or designer) is to change, further iterations in the program can account for those changes and update the preference function as necessary.

The agent system is separated into two segments: the agents themselves and the sorting system. The manager agent is static and does not change regardless of the product form space or the shape grammar rules. The manager receives the constraints, as described previously, including several additional pieces of information, which affect design optimization, and will be discussed at the end of this section.

Because the characteristic agents are collaborative, the manager plays a crucial role. The manager oversees the design process by determining which agent should be chosen at the beginning of each iteration. Each stage must start with the manager. This is best demonstrated by stepping through one iteration of the method:

1. The manager selects the top design from the incomplete designs list. This list is initially populated with blank designs.
2. The manager checks a list of shape grammar labels and markers attached to the top design to see which ones are active. The active labels and markers indicate which shape grammar rules can be applied to which product characteristics. The manager then stochastically chooses a single characteristic agent from all the potentially active characteristic agents.
3. The characteristic agent checks to see which labels and markers are active, and stochastically chooses a rule based upon the agent-rule matrix (Fig. 5). The characteristic agent then stochastically chooses the attribute values passed to the shape grammar interpreter with the rule choice.
4. The attribute values and the rule choice are passed to the shape grammar interpreter.
5. The shape grammar interpreter returns the design changes based on those attribute values; the values associated with the top design are updated and then evaluated to determine if the objective function has improved, e.g., higher utility. If not, the design is passed back to the incomplete designs list so that the agents can further improve the design.
6. If the objective function has improved, all the labels and markers are checked to determine if the design is complete. If any labels or markers still exist, the design is passed back to the incomplete designs list so that the agents can apply more shape grammar rules.
7. If the design is complete, i.e., no more shape rules can be applied and has an improved objective function, it is passed to the complete designs list. The complete designs list stores all the completed designs in one design iteration. Once the complete designs list is full, it is sorted, and the top designs are passed to the best designs list. The best designs list stores the best designs found through multiple design iterations.
8. If the desired number of design iterations (an input variable) has not been completed, the incomplete designs and complete designs lists are reinitialized and the process continues. If the desired number of iterations has been completed, the best designs list is sorted, and the final set of designs is passed to the design analysis for evaluation.

There are two advantages to sorting the complete designs list, best designs list and the incomplete designs list. First, by passing designs with lower objective function values back to the incomplete designs list; they can continue to be modified. Since the agents are implementing a shape grammar, this promotes design exploration. Designs with lower objective function values can be improved upon by characteristic agents, choosing modification rules, and thus making design modifications. Even if a design is complete, the agents can continue to modify various attributes until the objective function is improved. When implementing the shape grammar, there is no limitation as to the number of times an agent may choose to modify an attribute. This modification iteration can go on indefinitely. This actually could become a problem in practice, which will be discussed in Sec. 4.1.1.

Second, the sorting of the complete designs and best designs lists ensures that only the best designs within a single iteration are kept. After many iterations, the near-optimum designs will eventually be found. Since the process is always looking to maximize the objective function, as the top designs within a set get closer to the optimum, more iterations will be required with more modification rules making changes to the designs. The number of design iterations does not need to be large. If the design lists are large enough and each design is required to be better than the previous one, eventually, designs near the optimum will be found. All of this should be kept in the perspective that the intention of this MASGI is not to find the optimum, but to find a set of designs that are optimally directed, and hence, match the preference function.

4.1.1 Input Variables for List Sorting. In the agent system (Fig. 6), the manager accepts six variables that affect design list sorting: Direction, PercentRestart, Quantity, MaxIteration, and MaxTries.

Direction determines which way the design lists (incomplete designs, complete designs, and best designs) are sorted. If intending to create form designs that maximize the preference function (e.g., designs that the consumer will prefer), the lists are sorted so that the design with the largest positive value is first, then the next largest is second, and so on. When looking for form designs that minimize the preference function (e.g., designs that the consumer will not prefer), the lists are sorted so that the design with the largest negative objective function value is first, then next largest is second, and so on.

The *objective function* is created by multiplying PercentRestart by the objective function value of the previous design. In an optimization sequence, PercentRestart was typically set at 90% to encourage finding form designs close to the optimum, but also to allow exploration. If the only intention was to get as close to the optimum as possible, PercentRestart is set to 100% to ensure that each successive design in an iteration has a higher absolute objective function value than the previous design.

Quantity is the number of form designs created for each iteration, which is also the length of the design lists. If the number for Quantity is large, a form design with a large absolute objective function value is more likely to be found in a few iterations, depending on PercentRestart. The tradeoff is that the sorting time is increased due to more designs, and the overall time efficiency decreases. In the case study, it was found that a Quantity of 14 designs in the incomplete designs and complete designs lists was usually a good balance.

MaxIteration is the number of iterations before the best designs list is sent to the design analysis. This was found to be dependant upon the number of attributes being optimized. It was found in the case study that when a set of 55 attributes was used, 200 iterations were usually sufficient. For seven attributes, 20 iterations were usually sufficient. In general, to get a set of good designs, the minimum number of iterations should be three to four times the number of attributes. If not enough good designs were created in the assigned number of iterations, the method could be run again.

MaxTries is the fifth and final variable added to the manager input. As noted earlier, due to the nature of the shape modification rules, the agents can continue to modify form designs indefinitely, trying to find a design better than the optimum. MaxTries sets the limit as to how many rule applications can take place for a single form design. In the case study, it was found that setting MaxTries to 1000 iterations was sufficient, and very few designs used even half that many shape grammar rule applications. This number would vary greatly depending on the implemented shape grammar and the number of atomic attributes needed for a completed form design. Another technique would be for the iteration to exit if the objective function value for a modified design has not improved upon the previous top design within a certain percentage.

4.1.2 Effect of Input Variables for Sorting. To demonstrate how the iterative sorting methods guide and prune the designs, consider a typical design from the case study, in which the agents choose random values for the seven attributes in Fig. 3(*b*). The inputs to the manager are: Quantity=14 designs, MaxIteration =200 iterations and PercentRestart=90%. MASGI found a design with utility of 0.795, 99% of the maximum utility, using 101,966 rule applications to create 2800 complete form designs. MASGI automatically generates considerably more concept form designs than a human could, whether from their experience or through a manual implementation of the shape grammar. The number of rule choices needed for convergence is also highly dependent upon the objective function that is being optimized. If the objective function is linear, traditional optimization techniques would be sufficient. MASGI's iterative method is most beneficial for nonlinear objective functions.

4.2 Shape Grammar Interpreter. The shape grammar interpreter receives design choice information from the agent system, applies chosen shape grammar rules within parametric constraints, and passes the design changes back to the agent system. In this research, the unique contribution is the MAS, automatically implementing the shape grammar to create form designs according to an objective function. No significant changes beyond its adaptation to the MASGI are added to the shape grammar interpreter architecture of McCormack and Cagan [20], and this implementation does not include the use of emergence. The shape grammar interpreter receives the shape grammar rules (Fig. 7), which consist of curve creation rules and curve modification rules, as detailed in the Appendix.

The process sequences as follows.

1. Before the process starts, the shape grammar interpreter receives the full set of build and modify rules.
2. During the design process, the shape grammar interpreter receives a signal from the agent system that a certain rule has been chosen stochastically.
3. With the indication of a chosen rule comes the parametric values for the atomic attributes related to that rule; the design decision is made by the characteristic agent.
4. The shape grammar interpreter applies the chosen rule (build or modify), and removing and adding shapes as indicated.
5. If there is any change in labels or markers, the shape grammar interpreter indicates these.
6. All of the design changes (attributes, labels, and markers) are then passed back to the agent system as a completed design choice.

The shape grammar interpreter is independent of the agent system, in that it only receives design information, applies a rule, and returns design information. This isolation, made possible by the MAS, is advantageous, in that changes to the shape grammar can be made independently with respect to the rest of the MAS. The computational implementation of the shape grammar is best explained through the use of a pseudocode, detailed in Figs. 8 and 9, and explained through a running example. Recall that an existing multiagent architecture was not used, but this architecture was created especially for this MASGI.

The manager (Fig. 6) checks the top design from the incomplete designs list and sees that labels 6 and 7 are active, and markers exist for the ground, wheels, grill, and hood. The manager then stochastically chooses between characteristic agents (Fig. 5), and chooses the headlight agent. The headlight agent checks the labels and markers list for the top design (Fig. 8) and determines that labels 6 and 7 exist and the headlight markers do not exist. The headlight agent chooses to insert the shape and determines that it can apply rule 13. The hood agent then chooses

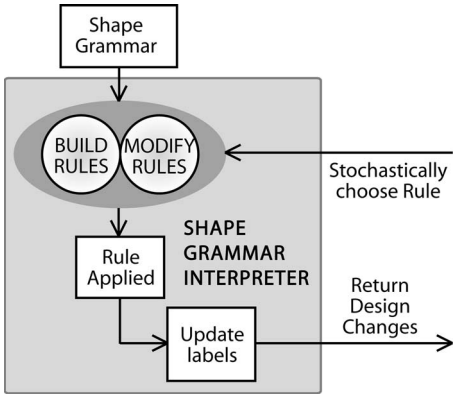

Fig. 7 Shape grammar interpreter process

```
//CHARACTERISTIC AGENT
get(labels and markers for top design from design list)
choose(insert or modify)
if(insert)
    determine(which rules can be applied based upon labels and markers)
    choose(rule)
    choose(parametric values for attributes)
    send(values to Rule)
if(modify)
    determine(which curves exist based upon markers)
    choose(rule)
    choose(curve)
    get(parametric values for attributes)
    modify(parametric values for attributes)
    send(values to Rule)
```

Fig. 8 Characteristic agent pseudocode

```
//SHAPE GRAMMAR RULE
receive(parametric values from characteristic agent)
get(top design from design list)
    put(parametric values to top design from design list)
get(labels and markers for top design from design list)
    update(labels and markers)
    put(labels and markers to top design from design list)
```

Fig. 9 Shape grammar rule pseudocode

to apply rule 13 (Fig. 10) and stochastically chooses the parametric value for V1hdltz. The headlight agent then passes these values to rule 13.

Rule 13 receives the attribute values from the headlight agent (Fig. 9). Rule 13 then gets the top design, as shown in Fig. 11, from the incomplete designs list (Fig. 5). Rule 13 takes the new attribute values, inserts the curve, and replaces these values in the top design. Rule 13 then gets the labels and markers for the top design and replaces them, based upon the application of rule 13, removing labels 6 and 7, and inserting markers for the headlight curves, as shown in Fig. 12. The updated top design with the updated labels and markers is then passed back to the agent system for design evaluation (Fig. 6).

The vehicle shape grammar allows for some nonlinearity in its application. Which order the characteristic rules are implemented does not need to be sequential. Additionally, since modification rules can be chosen at any time in the form design (as long as the curve being modified exists and the markers have not been removed), the number of rule choices can vary greatly per design. Table 1 shows the minimum number of rule choices needed for the characteristic agents to create a completed form design. The rules chosen by the manager are not counted as a step, in that they

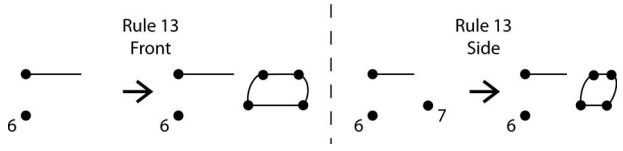

Fig. 10 Headlight rule

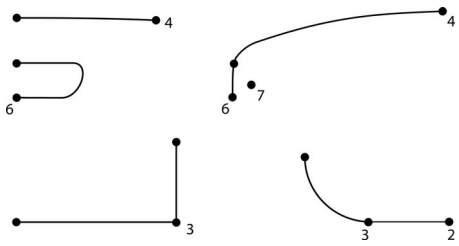

Fig. 11 Before headlight rule application

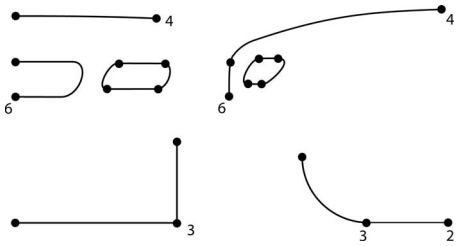

Fig. 12 After headlight rule application

Table 1 Most concise design sequence

	Agent	Rule	Verbal description
Step	Manager	Initial shape	Insert coordinate system
1	Ground	Rule 1	Set track width
2	Ground	Rule 2	Set wheel base
3	Ground	Rule 3	Insert ground curves
	Manager	Rule 4	Insert front wheel
4	Hood	Rule 5	Set cowl position
5	Hood	Rule 6	Insert hood
6	Grill	Rule 7–12	Insert grill
7	Headlight	Rule 13	Insert headlight
	Manager	Rule 14	Insert rest of vehicle
8	Any	Rule I	Remove markers

are fixed for each design.

In step 6 of Table 1, the grill agent can choose any one of rules 7–12 (a variety of grill forms) to get a completed form design and not change the total quantity of rule choices. Many different designs can be generated from just eight rule applications by varying which grill rule is chosen and by varying the attribute values. For each shape modification rule, the number of rule applications increases. As the agents are attempting to satisfy the objective function, they can continue to select shape modification rules until the form design improves.

To summarize, for each design decision, the implementation of any single rule is a cooperative effort between the characteristic agent and the rule. The agent determines which rules can be applied. Once a rule is chosen, the related attribute values are determined and passed to the shape grammar rule. The shape grammar rule accepts the modified attribute values and makes the appropriate changes to the labels and markers.

4.3 Preference Investigator. The preference investigator (Fig. 13) accomplishes the task of determining weights for the preference function and verifying that the chosen weights match the intended preference, e.g., a utility function [8]. How it accomplishes this will now be detailed by stepping through the process.

1. The choice of product attributes is an input to the preference investigator and becomes the attributes for the preference function.
2. This preference function is then passed to the agent system (the manager in particular) for form design generation.
3. Once the agent system has a finished set of designs, the designs are evaluated in the design analysis: a consumer survey in the case study.
4. The results from the design analysis are then analyzed. If this is the first iteration through the process, the weights for the attributes in the preference function have not yet been estimated.
5. The preference investigator determines these weights using LOGIT, PROBIT, Luce, or any applicable statistical analysis method.
6. These weights are included in the preference function and then passed back to the agent system for further form design exploration.

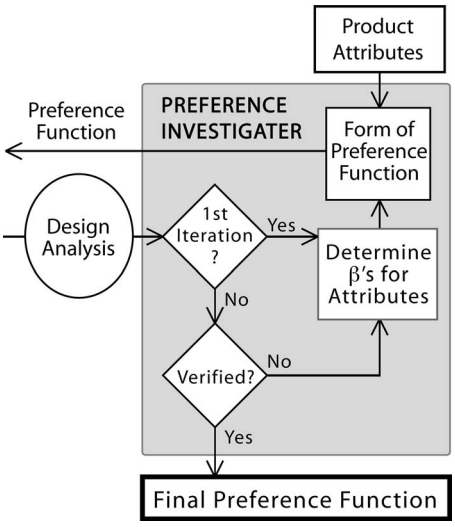

Fig. 13 Preference investigator process

7. After the second set of designs go through design analysis, the results determine if the weights for the preference function are verified.
8. If the preference function is not verified, then the weights are adjusted accordingly and the preference function is passed back to the agent system for further form design exploration and generation.
9. When the preference function is considered to sufficiently capture the intended preference, the process exits and the final preference function is determined.

The design analysis in step 3 could be any number of persons or programs, depending on the scenario. If the intention is to capture consumer preference, as in a new product form conceptualization, the design analysis could be a consumer survey. The design analysis could be a designer if the designer's preference is being evaluated, as when trying to understand a designer's personal style. The design analysis could even be computer programmed to analyze form designs according to an existing brand definition. In step 5, determining the attribute weights can be done iteratively or just once. The more iterations, the more likely the preference function truly represents the design preference.

Table 2 summarizes the attribute weights for respondent 30 from the consumer survey in the case study (refer to Fig. 3(*b*) for the attribute positions on the vehicle), where the preference function used was a quadratic utility function u_i (Eq. (1)), where x_i is an attribute from Fig. 3(*b*) and β_{ij} is a preference weight for that attribute. These weights, as part of the preference function, were passed through the MASGI to automatically generate the form design concepts shown in Figs. 14–16. As can be seen, the preference function causes a strong visual difference between the vehicle form designs

$$u_i = \beta_{i1}x_i^2 + \beta_{i2}x_i + \beta_{i3} \tag{1}$$

Table 2 Respondent 30 preference function weights

	β_1	β_2	β_3
Hcowlx	0	−0.0069	0.4395
Hcowlz	−0.0023	0.232	−5.735
V1hoodx	0	0	0
V1hoodz	−0.0033	−0.0776	−0.4049
V1grllz	0	0.0058	0.0607
V1hdltz	0.0046	−0.0802	0.3196
V4hdltz	0.0097	0.2718	1.5518

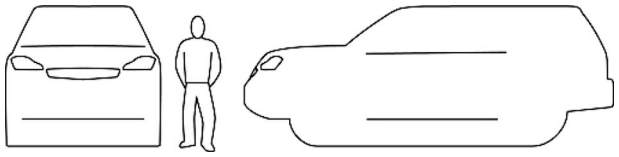

Fig. 14 Respondent 30 high utility design 1

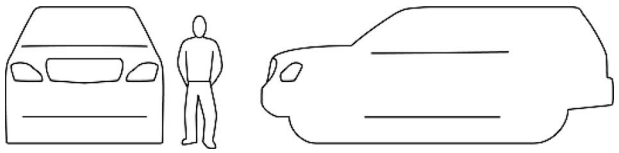

Fig. 15 Respondent 30 neutral utility design 2

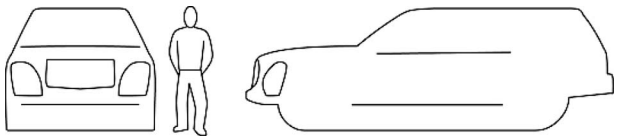

Fig. 16 Respondent 30 low utility design 6

5 Discussion

In summary, MASGI is an automated MAS that explores form design spaces and creates new designs through implementing a shape grammar while meeting the constraints of an objective function. The entire product form is represented with specific product attributes that are created through shape grammar rules. Each product attribute has a software agent responsible for its design decisions. All the agents work together to create a completed form design. Each "near-optimum" design is automatically created to match a preference function, e.g., a utility function representing a person's form preference. Figures 14–16 demonstrate just a few concept form design examples. MASGI is able to quickly produce a large number of concept form designs, which can then be refined in the early stages of new product development.

The next level of research should address the refinement of this methodology in three ways. First, the agent system and shape grammar implementation are automated, but the preference investigation was implemented independently through existing software packages. The independent analysis slowed down the overall form concept generation method so that only a single design analysis was possible within a reasonable amount of time. For this method to produce better results, through iteration, the preference analysis needs to be automated.

Second, preference modeling is still quite limited by the number of attributes that can be analyzed efficiently and effectively. In general, preference modeling research must create new methods for accurately determining preference for large design spaces, given a minimum amount of data. This, combined with the automation of the preference investigator, will shorten the design analysis time and enable this methodology to be used effectively for complex product form examples.

Third, while the agent system is intended to be as robust as possible, further refinement in the architecture will maximize the potential of this methodology. MASGI is currently a collaborative agent system, where each agent makes choices independent of one another and function sequentially. As the agents are refined to better simulate design teams, the agents will become cooperative, functioning in parallel and negotiating design decisions.

6 Conclusions

The form of a product must match consumer preference if it is to succeed in the marketplace. The challenge is to quickly create

product form concepts according to consumer preference so that these concepts can be evaluated and used as a foundation for further product design refinement. This paper introduced a MASGI for creating product form concepts based upon a derived consumer preference. The MASGI takes shape grammar rules, an agent-rule matrix, design constraints, and design attributes as input. A preference investigator determines an appropriate preference function by presenting a design analysis created using design of experiments. Software agents in an agent system explore and choose designs based upon the preference function. The software agents implement a shape grammar to automatically generate the chosen form design concepts. These product form designs can then be presented to the potential consumer for evaluation.

The MAS construction enables a flexible, automated shape grammar implementation. Should the design space change, product characteristic agents and shape grammar rules can be added or removed without needing to modify the entire program. Additionally, we have shown that in the implementation of the shape grammar, the software agents can choose many different rule sequences and still arrive at form designs that match the preference function. This demonstrates the creative potential built into the MASGI, which facilitates design exploration through the shape grammar and characteristic agents. Form designs are generated automatically and quite efficiently. The preference function provides a filter and guidance to produce a preferred design. As the design space becomes larger and more complicated, different simplified parametric representations will need to be explored.

Acknowledgment

Funding for this research was partially provided by the National Science Foundation under Grant No. DMI-0245218.

Appendix

Figures 17–23 show the initial shape, ground rules, wheel and hood rules, grill rules, headlight rule, final rule, and general modification rules, respectively.

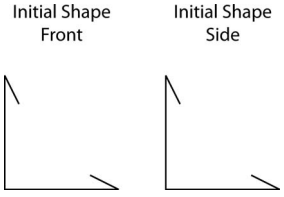

Fig. 17 Initial shape

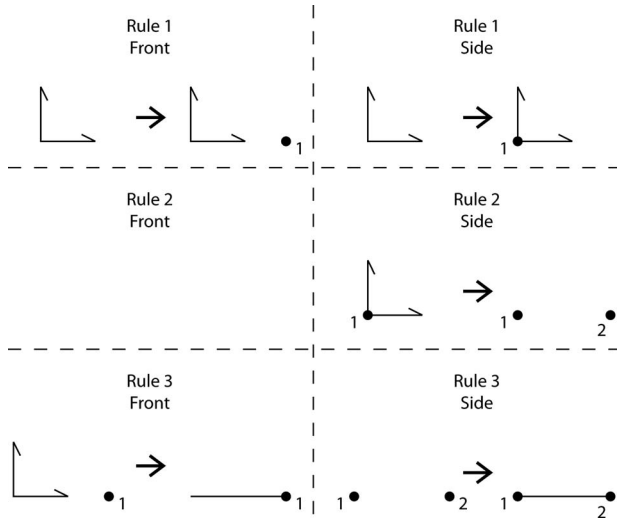

Fig. 18 Ground rules

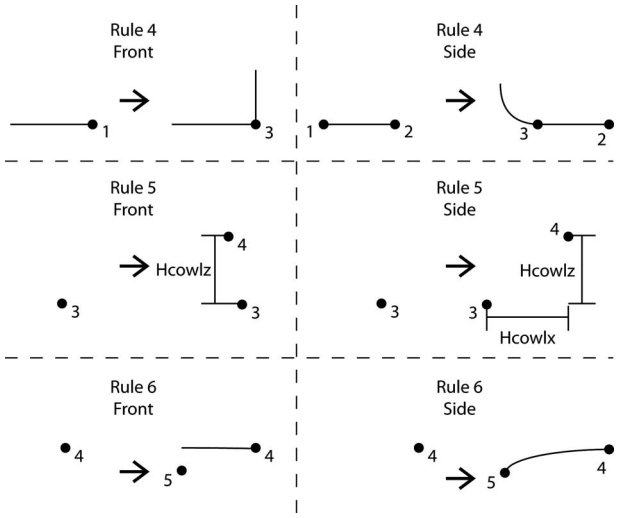

Fig. 19 Wheel and hood rules

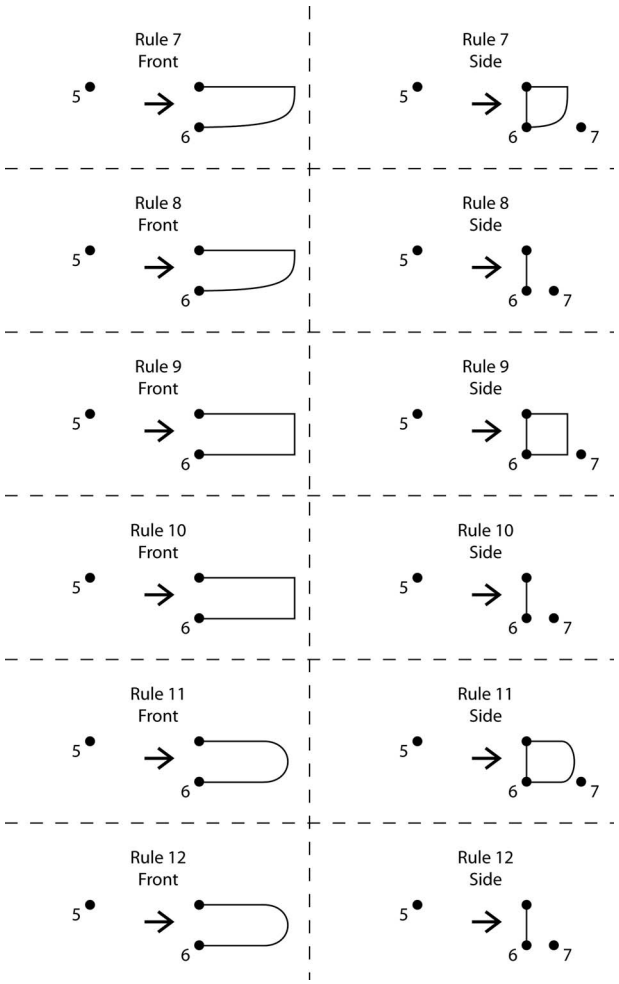

Fig. 20 Grill rules

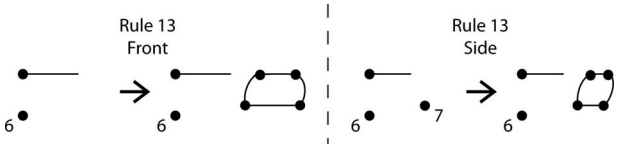

Fig. 21 Headlight rule

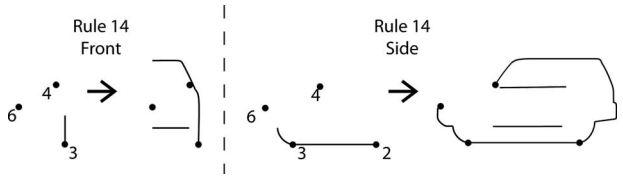

Fig. 22 Final rule

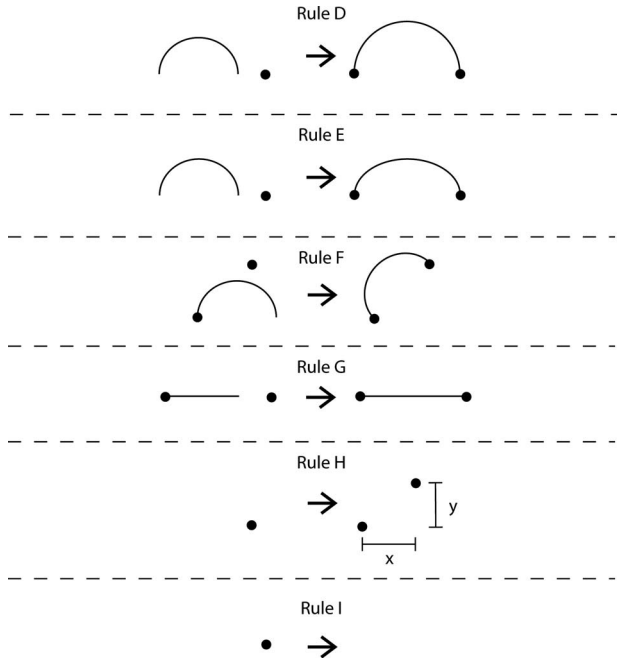

Fig. 23 General modification rules

References

[1] Orsborn, S., Cagan, J., Pawlicki, R., and Smith, R. C., 2006, "Creating Cross-Over Vehicles: Defining and Combining Vehicle Classes Using Shape Grammars," Artif. Intell. Eng. Des. Anal. Manuf., **20**(3), pp. 217–246.

[2] Orsborn, S., Boatwright, P., and Cagan, J., 2009, "Quantifying Aesthetic Form Preference in a Utility Function," ASME J. Mech. Des., **131**(6), p. 061001.

[3] Berkowitz, M., 1987, "Product Shape as a Design Innovation Strategy," J. Prod. Innovation Manage., **4**(4), pp. 274–283.

[4] Bloch, P. H., 1995, "Seeking the Ideal Form: Product Design and Consumer Response," J. Marketing, **59**(3), pp. 16–29.

[5] Von Neumann, J., and Morgenstern, O., 1944, *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, NJ.

[6] Chen, W., Wiecek, M., and Zhang, J., 1999, "Quality Utility—A Compromise Programming Approach to Robust Design," J. Mech. Des., **121**(2), pp. 179–187.

[7] Callaghan, A., and Lewis, K., 2000, "A 2-Phase Aspiration-Level and Utility Theory Approach to Large Scale Design," *Proceedings of the ASME DETC 2000*, Baltimore, MD.

[8] Orsborn, S., Boatwright, P., and Cagan, J., 2008, "Quantifying Aesthetic Form Preference in a Utility Function," *Proceedings of the ASME Design Engineering Technical Conferences: Design Theory and Methodology Conference*, Brooklyn, NY.

[9] Smith, R. C., Pawlicki, R., Kokai, I., Finger, J., and Vetter, T., 2007, "Navigating in a Shape Space of Registered Models," IEEE Trans. Vis. Comput. Graph., **13**(6), pp. 1552–1559.

[10] Stiny, G., and Mitchell, W. J., 1978, "The Palladian Grammar," Environ. Plann. B, **5**(1), pp. 5–18.

[11] Stiny, G., 1977, "Ice-Ray: A Note on the Generation of Chinese Lattice Designs," Environ. Plann. B, **4**(1), pp. 89–98.

[12] Agarwal, M., and Cagan, J., 1998, "A Blend of Different Tastes: The Language of Coffeemakers," Environ. Plan. B: Plan. Des., **25**(2), pp. 205–226.

[13] Pugliese, M., and Cagan, J., 2001, "Capturing a Rebel: Modeling the Harley-Davidson Brand Through a Motorcycle Shape Grammar," Res. Eng. Des., **13**(3), pp. 139–156.

[14] Mccormack, J., Cagan, J., and Vogel, C., 2004, "Speaking the Buick Language: Capturing, Understanding, and Exploring Brand Identity With Shape Grammars," Des. Stud., **25**(1), pp. 1–29.

[15] Stiny, G., 1980, "Introduction to Shape and Shape Grammars," Environ. Plann. B, **7**(3), pp. 343–351.

[16] Krishnamurti, R., 1980, "The Arithmetic of Shapes," Environ. Plann. B, **7**(4), pp. 463–484.

[17] Krishnamurti, R., and Earl, C., 1992, "Shape Recognition in Three Dimensions," Environ. Plan. B: Plan. Des., **19**(3), pp. 267–288.

[18] Chase, S. C., 1989, "Shapes and Shape Grammars: From Mathematical Model to Computer Implementation," Environ. Plan. B: Plan. Des., **16**, pp. 215–242.

[19] Tapia, M. D., 1999, "A Visual Implementation of a Shape Grammar System," Environ. Plan. B: Plan. Des., **26**(1), pp. 59–73.

[20] Mccormack, J., and Cagan, J., 2002, "Supporting Designers' Hierarchies Through Parametric Shape Recognition," Environ. Plan. B: Plan. Des., **29**(6), pp. 913–931.

[21] Mccormack, J., and Cagan, J., 2006, "Curve-Based Shape Matching: Supporting Designer's Hierarchies Through Parametric Shape Recognition of Arbitrary Geometry," Environ. Plan. B: Plan. Des., **33**(4), pp. 523–540.

[22] Mccormack, J., and Cagan, J., 2002, "Designing Inner Hood Panels Through a Shape Grammar Based Framework," Artif. Intell. Eng. Des. Anal. Manuf., **16**(4), pp. 273–290.

[23] Dastani, M., 2008, "2apl: A Practical Agent Programming Language," Auton. Agents Multi-Agent Syst., **16**(3), pp. 214–248.

[24] Kagal, L., and Finin, T., 2007, "Modeling Conversation Policies Using Permissions and Obligations," Auton. Agents Multi-Agent Syst., **14**(2), pp. 187–206.

[25] Ennigrou, M., and Ghedira, K., 2008, "New Local Diversification Techniques for Flexible Job Shop Scheduling Problem With a Multi-Agent Approach," Auton. Agents Multi-Agent Syst., **17**(2), pp. 270–287.

[26] Bazzan, A. L. C., 2005, "A Distributed Approach for Coordination of Traffic Signal Agents," Auton. Agents Multi-Agent Syst., **10**(1), pp. 131–164.

[27] Klugl, F., Herrier, R., and Fehler, M., 2006, "Sesam: Implementation of Agent-Based Simulation Using Visual Programming," *Proceedings of the AAMAS*, Hakodate, Japan.

[28] Talukdar, S. N., 1993, "Asynchronous Teams," *Proceedings of the Fourth International Symposium on Expert Systems Applications to Power Systems*, LaTrobe University, Australia.

[29] Talukdar, S. N., 1999, "Collaboration Rules for Autonomous Software Agents," Decision Support Sys., **24**(3–4), pp. 269–278.

[30] Sachdev, S., 1998, "Explorations in Asynchronous Teams," Ph.D. thesis, Carnegie Mellon University.

[31] Campbell, M., Cagan, J., and Kotovsky, K., 1999, "A Design: An Agent-Based Approach to Conceptual Design in a Dynamic Environment," Res. Eng. Des., **11**(3), pp. 172–192.

[32] Olson, J. T., and Cagan, J., 2004, "Interagent Ties in Team-Based Computational Configuration Design," Artif. Intell. Eng. Des. Anal. Manuf., **18**(2), pp. 135–152.

[33] Shaker, C., and Brown, D. C., 2004, "Constructing Design Methodologies Using Multiagent Systems," Artif. Intell. Eng. Des. Anal. Manuf., **18**(2), pp. 115–134.

[34] Singh, M. P., and Huhns, M. N., 1994, "Automating Workflows for Service Order Processing: Integrating Ai and Database Technologies," IEEE Expert, **9**(5), pp. 19–23.

[35] Durgee, J. F., 1988, "Product Drama," Journal of Advertising, **17**, pp. 42–49.