

EXPLORATION, EXPLOITATION AND OPTIMIZATION OF ENGINEERING
DESIGN SPACE USING GRAPH GRAMMARS.

by

Pranay Kilaru

A project

submitted in partial

fulfillment of the requirements for the degree of

Master of Science in Computer Science

in the Department of Computer Science

California State University, Fresno

05/06/2010

APPROVED

For the Department of Computer Science:

We, the undersigned, certify that the project of the following student meets the required standards of scholarship, format, and style of the university and the student's graduate degree program for the awarding of the master's degree.

Pranay Kilaru

Project Author

Dr. Shih-Hsi Liu

Date

AUTHORIZATION FOR REPRODUCTION

_____ I grant permission for the reproduction of this report in part or in its entirety without further authorization from me, on the condition that the person or agency requesting reproduction absorbs the cost and provides proper acknowledgment of authorship.

_____ Permission to reproduce this report in part or in its entirety must be obtained from me.

Signature of author: _____

ABSTRACT

EXPLORATION, EXPLOITATION AND OPTIMIZATION OF ENGINEERING DESIGN USING GRAPH GRAMMARS

- This project presents a new way of automating and optimizing the conceptual designs with the help of existing tools such as Solid Works, GraphSynth and a proven optimization algorithm SHCLVND. Conceptual design involves development of 100's of iterative designs and is very complicated and time consuming to carry out manually, so automation of concepts is extremely significant in engineering design. There are some previous works regarding the same which give a random model in some graph or mathematical representations, where users can neither properly visualize the end-concept nor give a correct feedback. Compared to existing automation projects, this project presents users/designers with not just some mathematical representations of a concept but an instant 3D model by integrating with Solid Works using C# under .NET framework. Such a model enables users to interact and compare to other concepts to provide a feedback, which is then used to build some knowledge on users' inclination by implementing some existing search and optimization algorithms. Finally, the rules and parameter selection for the future concepts are influenced in such a way that they are much closer to users' preferences. This project is experimented with a shampoo bottle example and the experimental results shows that the concepts are consistent with users' preference.

Author: Pranay Kilaru.

ACKNOWLEDGMENTS

I would like to fully acknowledge my project guide Dr. Shih-Hsi “Alex” Liu and co-advisor Dr. Rahul Rai for their support and patience, which helped in the successful completion of my project. I would also like to thank all those who directly or indirectly helped me in modeling this project into a comprehensive one. I am indebted to the authors of the books and journals that helped me as a reference throughout the project.

TABLE OF CONTENTS

LIST OF FIGURES	7
LIST OF TABLES	8
1. Introduction.....	9
2. Background.....	12
3. Literature Review	15
4. Research Methodology	18
4.1 Design Approach	18
4.2 Developing Grammar	21
4.3 Evaluation	27
4.4 Optimization	30
5. Results.....	34
6. Conclusion	37
References.....	39

LIST OF FIGURES

Figure	Page
1. GraphSynth screen shot with Seed Graph, Rule-Set and Some Rules displaying their LHS and RHS.	17
2. A completely random model and the rules that made it..	23
3. GUI window displaying four models side by side for visual comparison.	24
4. Comparer window in which rating is given by comparing candidate pairs.	25
5. Pseudo Code showing parameter optimization and rule selection involved in each generation.	27
6. Bottle designs developed using Random Candidate Generator..	29

LIST OF TABLES

Table		Page
1.	Rule number with their LHS-RHS and screen shots of an example scenario is presented	19
2.	Table displaying the probabilities of the rules	33
3.	Table displaying the Mu and Sigma of the parameters	34

1. Introduction

Engineering design [14] is a process, which develops a product and provides it to the customer. It consists of phases such as customer research, conceptual design, evaluation, detailed design-production, and marketing phase. Conceptual design involves development of 100's of iterative designs and is very complicated and time consuming to carry out manually, so automation of concept generation is extremely significant in engineering design and Graph Grammars [17] is one of the popular approaches used to automate this conceptual design process. Compared to some existing automation projects which use Graph Grammars, the main objective here is to present users/designers with not just some mathematical or graph representations of a concept but an instant 3D model, which enables users to interact and compare to other designs.

Agarwal and Cagan [1] are the people who first used Graph Grammars approach to generate a design. They introduced a grammar for coffee maker, which can be used to develop a large number of new designs of coffee makers. Later, Campbell and Rai [16] are the first people who used a Graph Grammars tool called GraphSynth to introduce a grammar for neck ties, which generates all the possible tie knots.

Graph Grammars can be used to create thousands of new designs/solutions but there should be some sort of evaluation techniques to get the optimal solution. Automated programs can be used to calculate solutions, but they cannot handle multi-objective

decision-making, which can only be handled by a human being. For this purpose an end user/customer is asked to give his/her feedback on a set of designs based on aesthetics, shape, ergonomics, and cost, among others. Then the feedback is converted into a form of weight-age to the rules used and the program gives best final design depending on the user's preferences.

In this project the possibility of automation of concept generation is shown by taking shampoo bottle as an example. The existing automation projects were only successful in generating the graph representations of the actual design. If Campbell and Rai's [16] earlier work on neck tie grammar is considered, the user has no way to visualize how a particular tie-knot model will look like. The user has to manually do all the tie-knots according to the generated graph representation to actually see and evaluate. Also there is no scope for parameters in the tie-knot grammar. In contrast the shampoo bottle makes it possible to have all the basic geometric shapes in the generative grammar and also there is scope for parameters to work on optimizing these parameters.

This project provided a new way for automating and optimizing the concept generation process, which can be applied in many situations where user interactions and contributions are vital. The most important aspect of this project is the end-user doesn't need to know or involve in creation of grammar rules. If an expert in that domain already created a set of generative grammar rules, all the user needs to do is to customize the concept according to his preferences.

Section 2 presents the background of all the tools and concepts used in this project. Section 3 provides a review of related work in this area. The methodology adopted to develop this project is discussed in Section 4. Experimental results are shown and explained in Section 5. Section 6 presents the conclusion and direction of future work.

2. Background

This section provides detailed explanation of all the tools, languages and algorithms used in this project. This project is developed using Solid Works [8], GraphSynth [3] and .NET [21] Framework's Base Class Libraries such as graphic libraries, LINQ-XML libraries [12] and COM interoperability libraries [6]

The front end of this system is developed using the Microsoft Visual Express Edition 2008 and eDrawings viewer component [10]. Microsoft Visual C-Sharp Express edition is a free integrated development environment (IDE) developed by Microsoft to provide an efficient and very simple IDE for use in academia and also for hobbyists. The important and very useful feature provided by the code editor present in the IDE is an automatic code completion feature called IntelliSense, which greatly reduces the time wasted in looking for available functions and properties while coding. The eDrawings viewer component is a free to use software available for viewing the 3D models generated by various CAD tools, which in our case is Solid Works.

C#, commonly pronounced as C-Sharp [25], is a multi-paradigm programming language on the .NET Framework developed by Microsoft to provide a simple, common and a powerful programming language. C# syntax is highly expressive. In fact much of the syntax style of C-Sharp resembles C++ and Java. C# being a multi-paradigm

language has many innovative language constructs and provides all the building blocks of programming such as encapsulation, inheritance and polymorphism.

The .NET Framework [21] is a large collection of software libraries developed by Microsoft for use on Windows operating systems. This provides a large range of features for programmers, who develop applications which involve designing user interface, creating and retrieving from XML files, writing highly complex numeric algorithms and integration with legacy COM software applications such as Solid Works.

The back end of this system involves making calls to Solid Works API and retrieving data that is stored in XML files. Solid Works is 3D mechanical computer aided designing (CAD) software developed by Dassault Systemes [8]. Solid Works is COM based software, so the Solid Works API is accessed with the help of COM interoperability library of the .Net Framework. The data in the XML files is retrieved and modified using .Net Framework's LINQ-XML libraries.

GraphSynth [3] is a publicly available graph grammar software tool developed at UT, Austin by Dr. Matt Campbell to make and visualize graphs, and there are many tools available to create these ranging from GraphViz to Visio. What makes GraphSynth unique is the option to define grammar rules. GraphSynth is mainly developed to abstract any engineering design process in to a set of grammar rules and then to automate the process of generating these designs. GraphSynth basically has 3 important components, namely, seed graph, rule and rule set.

GraphSynth's seed graph is nothing but an abstract graph representation of the initial state of an object's design process. A typical rule in GraphSynth consists of two main components, namely, Left Hand Side (LHS) and Right Hand Side (RHS). The condition required for a transformation rule to be applied on a graph is displayed on the LHS and the new state of the graph after the transformation is displayed on the RHS. All the rules are compiled together in to a collection called rule set.

To start generating a design in GraphSynth, a seed graph and a rule set should be loaded in GraphSynth window. GraphSynth provides 3 options to generate a design which are "User choose apply", "Random choose apply" and "Custom search". "User choose apply" gives control to the user to select the rules and generate the design of his/her interest. Conversely, "Random choose apply" chooses rules randomly and generates a design. "Custom search" is the process where the selection of the rules is controlled by a custom algorithm that meets the user's needs.

3. Literature Review

This section examines some of the existing systems and technologies previously explored by the researchers and effectively used in the real world.

Graphs and graph transformations are usually used by mathematicians to perform operations such as additions, subtractions and mappings, to name a few. Engineering designers after thorough research came to know that graph transformation systems can be effectively used to represent almost all complex engineering systems.

The main advantage of graph grammars is that we can instantly create the designs/solutions using the grammar instead of storing them in memory. Design of systems/processes using GraphSynth usually starts with a seed graph and after application of several transformations leads to final design. Grammar kind of approach for a design gives chance to designers to interpret the design, make modifications and develop a creative yet compliant design [4].

Agarwal and Cagan are the people who first used graph grammars to generate a design [1]. They introduced a grammar for making a coffee maker, which can be used to develop a large number of new designs. Dorina and Shen [7] used graph grammar method to transform UML models. Schurr used graph grammars to define the syntax of

visual languages and generate visual language parsers [2]. Siddique and Rosen [26] developed a common platform for similar product design using graph grammar approach. Du, Tsang and Jiao used graph grammars to transform customer requirements in customer view to product family design in technical view [11].

Evaluating and selecting a design from a pool of designs is equally difficult as designing a process/system. The most important and old evaluation method is Pugh's concept selection method [15]. This process mainly deals with qualitative evaluation in which a matrix is used to store various alternatives and take final decision. Other important evaluation techniques developed by researchers include numerical concept scoring [24], analytical hierarchy process [20], among others. All these methods are similar in evaluating: They go like storing the solution in a matrix, weigh or score them according to the features/properties, and aggregate the final score which give the best solution out of all designs.

Many evaluation techniques were developed but none of these methods has human interaction involved in them. So when it comes to situation of multi-decision-making situations, programs cannot give effective results due to lack of human interaction. This project starts with a heuristic algorithm that provides the user with a variety of designs/solutions. User carefully studies all available solutions and gives his/her rating. These data are converted into a preference model, which selects final design from the design space. Genetic Algorithms (GA) are modified into Interactive Genetic Algorithms

(IGA) by replacing the traditional genetic algorithm with interactive user feedback [19]. But the main problem with IGA's is that they cannot search design trees. One should always remember that for a tree, solutions exist at different levels and genetic algorithms are good for fixed length strings, so for this reason Artificial Intelligence (AI) should be used for evaluating design trees [5][13].

4. Research Methodology

The system is developed using Solid Works, GraphSynth, .Net Framework's Base Class Libraries such as graphic libraries, LINQ-XML libraries [12] and COM interoperability libraries [6]. This provides a large range of features for programmers, who develop applications which involve designing user interface, creating and retrieving from XML files, integration with Solid Works through API and writing highly complex numeric algorithms.

4.1 Design Approach

The main approach taken for this project is as follows,

1. Define grammar rules for a particular domain.
2. Generate some models.
3. Present those models to users for feedback.
4. Based on this feedback, make modifications to probabilities of rule selection and optimize the values parameters if any.
5. Iteratively do steps 2-4 with updated probabilities and ranges, until a user preferred optimal solution is generated.

Shampoo bottle domain is taken as an example for this project. The main reason behind choosing shampoo bottle is all the existing automation projects were successful in generating the mathematical / graph representations of the actual design. If we consider

Campbell and Rai's work, the user cannot visualize how a particular tie-knot model will look like, he/she has to manually do the tie-knots to actually see and evaluate it and also there are no parameters involved in the tie-knot grammar. In contrast the shampoo bottle makes it possible to have all the basic geometric shapes in our grammar and also it involves parameters to work on the optimization part.

A Shampoo bottle is one of the common objects that we use on a daily basis. Various designs of bottles are available in market today differentiating one from another. Each company has its own special design and tries to retain the design with time. In contrast to this, some companies try to attract customers by releasing new designs. When we talk about new designs they can be of any design available in the market to a completely new design that we have never seen. New designs can be drawn manually by selecting each and every feature of the design. If we continue doing this, we might end up with some hundreds of designs, which are impossible to draw. In addition to this, we might not also get all possible designs. Instead of drawing all possible designs, another way is to find out the grammar for drawing the bottle. It might be completely new to hear the phrase "grammar of a design", but it is true that any design/process can be broken down into some grammar rules, which will be easy to interpret.

Any design/object/process has a specific grammar; if one can code that grammar it will be very easy to get new innovative designs by just manipulating the grammar. Now this concept is explained by taking an example of a shampoo bottle design. A shampoo

bottle can be easily designed using 29 grammar rules. By manipulating these 29 rules, thousands of new designs can be developed. But it is not advisable to draw all the possible designs and store them somewhere as it is equally cumbersome and occupies huge memory. If thousands of designs are available, it is also very difficult to search the space for best optimal design. Multiple decisions making is mandatory to select the best design and it can be done only by human interaction.

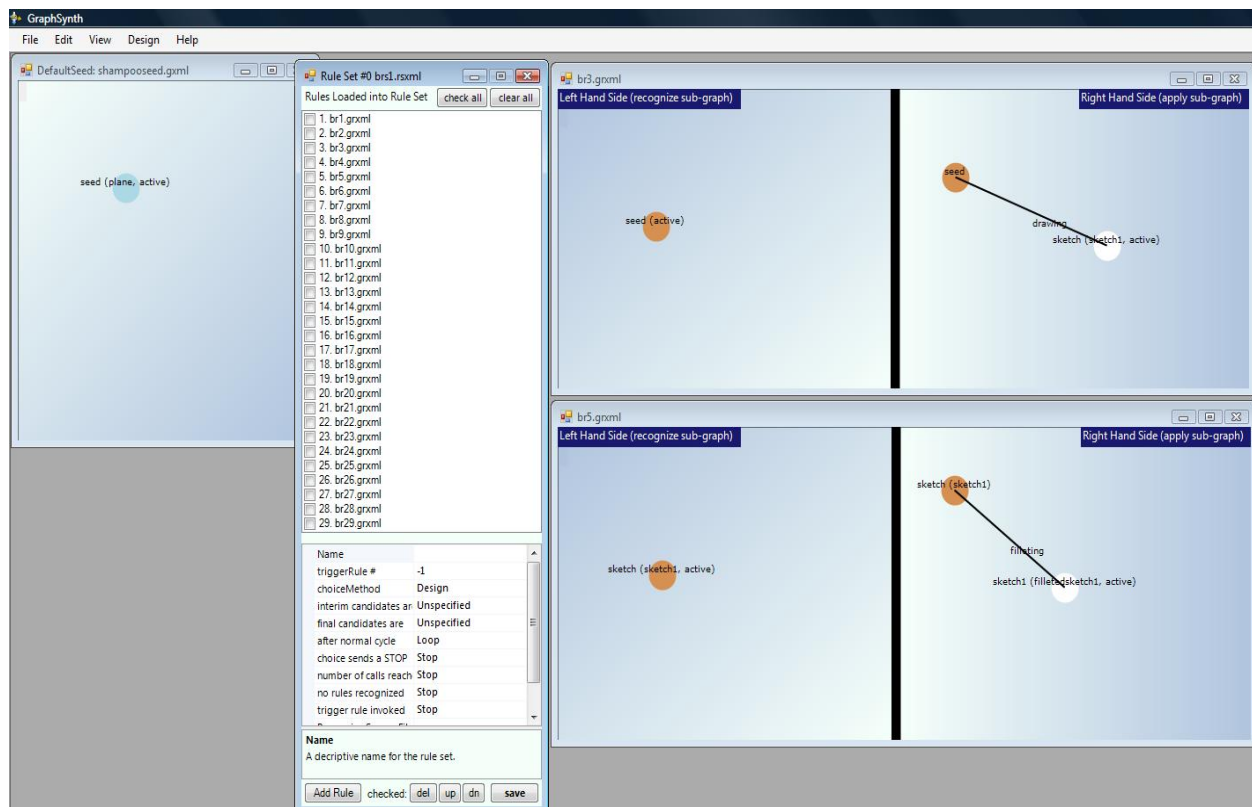


Figure 1: GraphSynth screen shot with Seed Graph, Rule-Set and Some Rules displaying their LHS and RHS

To start generating a design, the bottle grammar seed graph and the corresponding rule set are loaded in GraphSynth window. GraphSynth provides 2 options to generate a


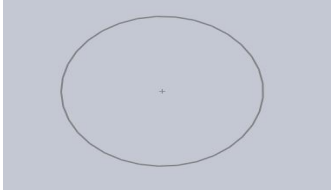

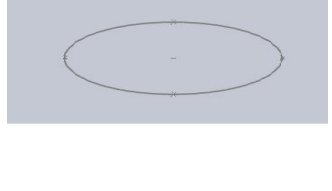
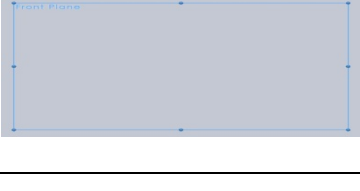
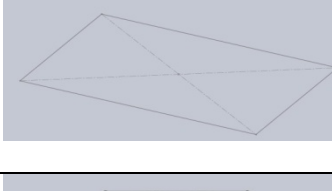
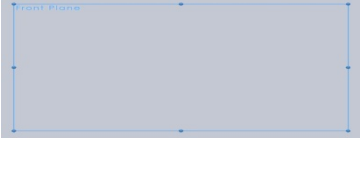

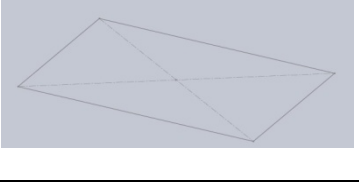
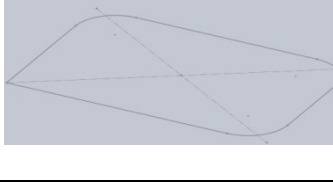
design which are “User choose apply” and “Random choose apply”. “User choose apply” gives control to the user to select the rules and generate the design of his/her interest. Conversely, “Random choose apply” chooses rules randomly and generates a design.

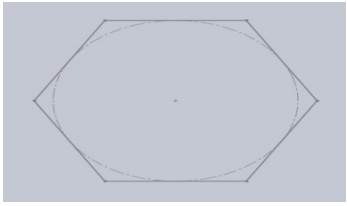
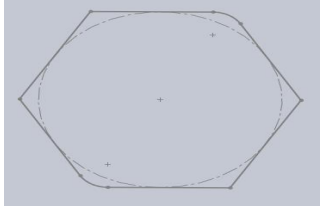
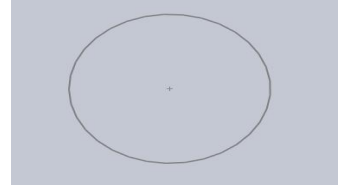
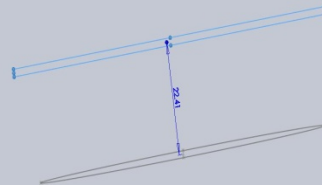
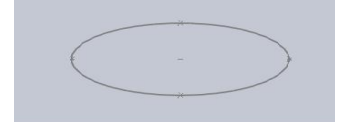




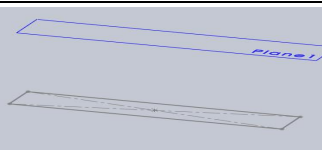
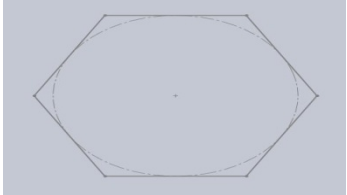
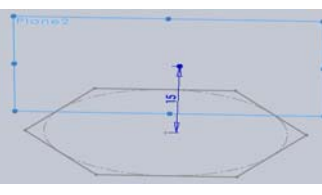

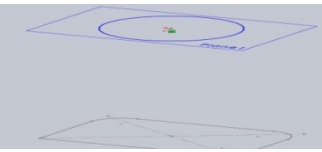
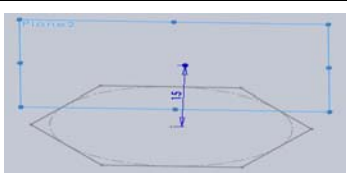

4.2 Developing Grammar

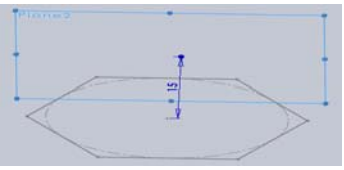
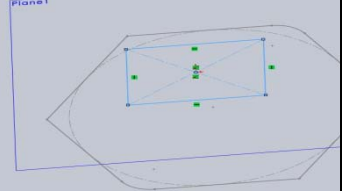
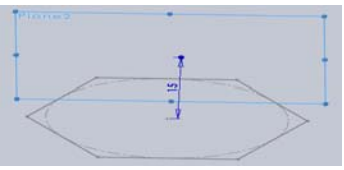
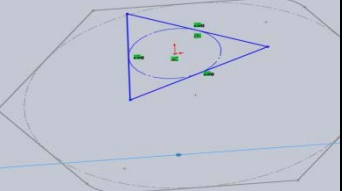
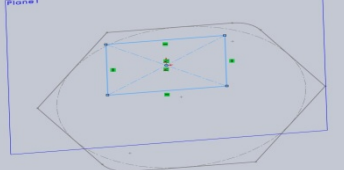
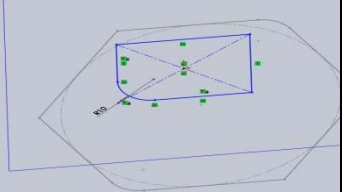
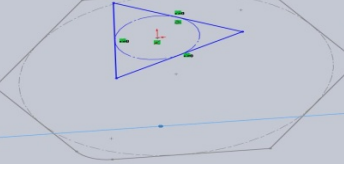
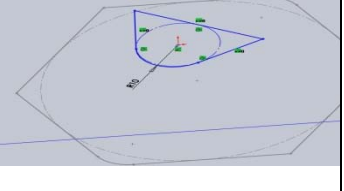

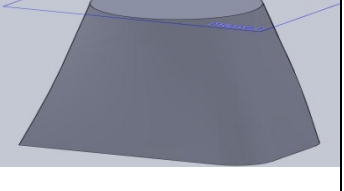
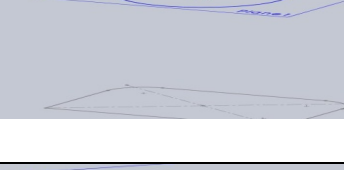



Design of a shampoo bottle can be divided into 2 parts, namely, the body and the neck. Both body and neck parts are drawn on the same basis i.e., they both have same steps except for variations in dimensions. First rule of the design starts with selection of a plane, this is where the base part of the bottle lies on and it can be any of the 3 planes top, front or side planes. Once plane is selected, next rule is to draw a base shape on the selected plane. Base shape of a bottle can take any shape such as circle, ellipse, triangle or polygon. After drawing the base shape, a plane should be selected at a required height equal to length of the body of bottle. Then again on the selected plane one more basic shape can be drawn to represent the top part of the body and it can be any of the above mentioned shapes. Here if the base shape drawn is either triangle or polygon one more rule comes into play, called fillet rule, which makes the sharp edges as curves. After drawing the 2 shapes on 2 different planes our next rule is to connect these shapes to make it a 3D shape. This can be accomplished either by lofting both the base shapes on different planes or drawing guide curves such as straight lines/arcs to connect the 2 shapes and then lofting the shapes using the curves. Once this part is done a basic shape of cylinder-like object is formed. Then following the same rules but changing the parameters gives a similar shape on top of the body which is called neck. And the last

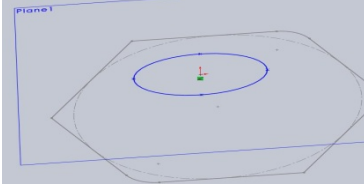
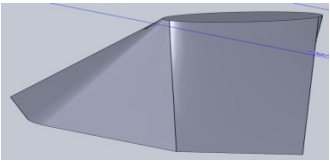
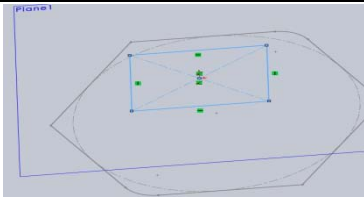
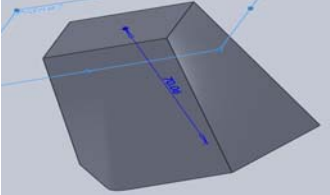
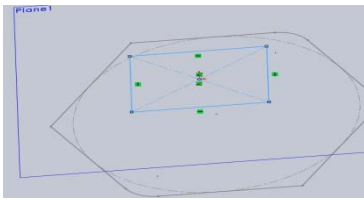
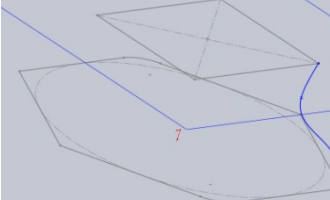
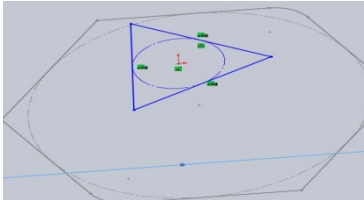
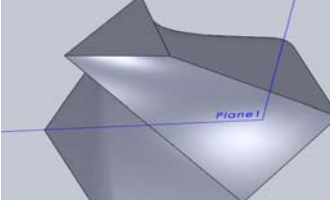
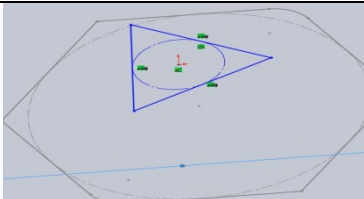
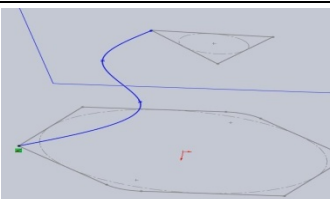

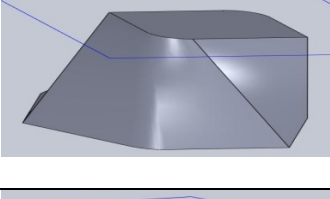

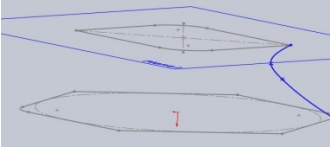
rule after forming the neck is shelling the whole shape to make it hollow. This set of 29 rules provides a way to draw random shapes of bottles instantly which will be both new and compliant.

All the above mentioned rules are stored in a set, called rule set. This rule set along with a seed graph is loaded in GraphSynth and are invoked to generate bottle designs. All the rules with their application in the Solid Works are presented in Table 1.

Rule 1	Seed(Plane)	Seed(Plane) → Sketch(circle)		
Rule 2	Seed(Plane)	Seed(Plane) → Sketch(ellipse)		
Rule 3	Seed(Plane)	Seed(Plane) → Sketch(rectangle)		
Rule 4	Seed(Plane)	Seed(Plane) → Sketch(polygon)		
Rule 5	Sketch(rectangle)	Sketch(rectangle) → Sketch(filleted)		

Rule 6	Sketch(polygon)	Sketch(polygon) → Sketch(filleted)		
Rule 7	Sketch(circle)	Sketch(circle) → Plane(Reference plane)		
Rule 8	Sketch(ellipse)	Sketch(ellipse) → Plane(Reference plane)		
Rule 9	Sketch(filleted)	Sketch(filleted) → Plane(Reference plane)		
Rule 10	Sketch(rectangle)	Sketch(rectangle) → Plane(Reference plane)		
Rule 11	Sketch(polygon)	Sketch(polygon) → Plane(Reference plane)		
Rule 12	Plane(Reference plane)	Plane(Reference plane) → Sketch(circle2)		
Rule 13	Plane(Reference plane)	Plane(Reference plane) → Sketch(ellipse2)		

Rule 14	Plane(Reference plane)	Plane(reference plane) → Sketch(rectangle2)		
Rule 15	Plane(Reference plane)	Plane(Reference plane) → Sketch(polygon2)		
Rule 16	Sketch(rectangle2)	Sketch(rectangle2) → Sketch(filleted2)		
Rule 17	Sketch(polygon2)	Sketch(polygon2) → Sketch(filleted2)		
Rule 18	Sketch(circle2)	Sketch(circle2) → Feature(Loft)		
Rule 19	Sketch(circle2)	Sketch(circle2) → Feature(Guide)		
Rule 20	Sketch(ellipse2)	Sketch(ellipse2) → Feature(Loft)		

Rule 21	Sketch(ellipse2)	Sketch(ellipse2) → Feature(Guide)		
Rule 22	Sketch(rect2)	Sketch(rect2) → Feature(Loft)		
Rule 23	Sketch(rect2)	Sketch(rect2) → Feature(Guide)		
Rule 24	Sketch(poly2)	Sketch(poly2) → Feature(Loft)		
Rule 25	Sketch(poly2)	Sketch(poly2) → Feature(Guide)		
Rule 26	Sketch(filleted2)	Sketch(filleted2) → Feature(Loft)		
Rule 27	Sketch(filleted2)	Sketch(filleted2) → Feature(Guide)		

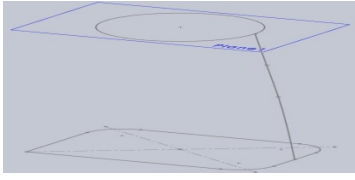
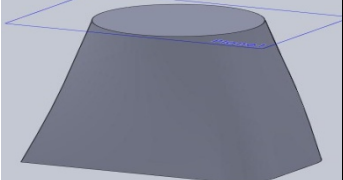
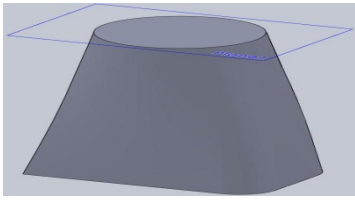
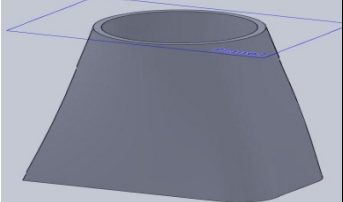
Rule 28	Feature(Guide)	Feature(Guide)→Feature(Loft)		
Rule 29	Feature(Loft)	Feature(Loft)→Feature(Shell)		

Table 1: Rule number with their LHS-RHS and screen shots of an example scenario are presented.

All the 29 rules presented above are not involved in a single model. Most of the above rules are parallel rules, i.e., they are siblings in a tree and only one will get selected based on user preferences. Figure 2 shows how a complete random model is generated from a seed graph and all the available rules at any step in the generation and the green color is given to the rule number that was applied at that step. At first step, four options are available, rules 1, 2, 3 and 4 of which rule 3 is selected. The screen-shot beside the rule shows what the shape generated after applying that particular rule. A final model of the generated shampoo bottle after applying 7 rules can be seen in the Figure.2.

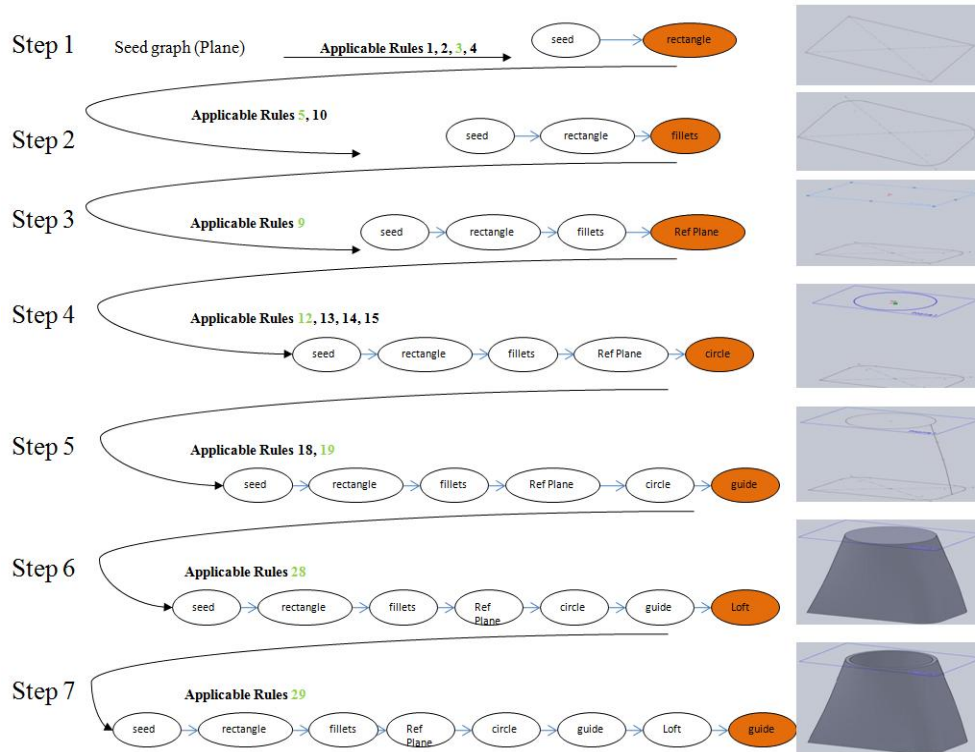


Figure 2: A completely random model and the rules that made it.

4.3 Evaluation

Evaluating and selecting optimal designs from design space is an important part in the process. Feedback from a customer is integrated with a computer program to get the best design. In order for a customer to give feedback, the final design should be shown. As both Solid Works and GraphSynth can be integrated using C#, a link is formed between them. Parameters from GraphSynth rules are sent to Solid-Works through C#, and Solid Works generates designs according to the data given, four at a time, these four generated models are programmatically saved as JPG formatted images and are displayed to the user using eDrawings viewer components as shown in Figure 3.

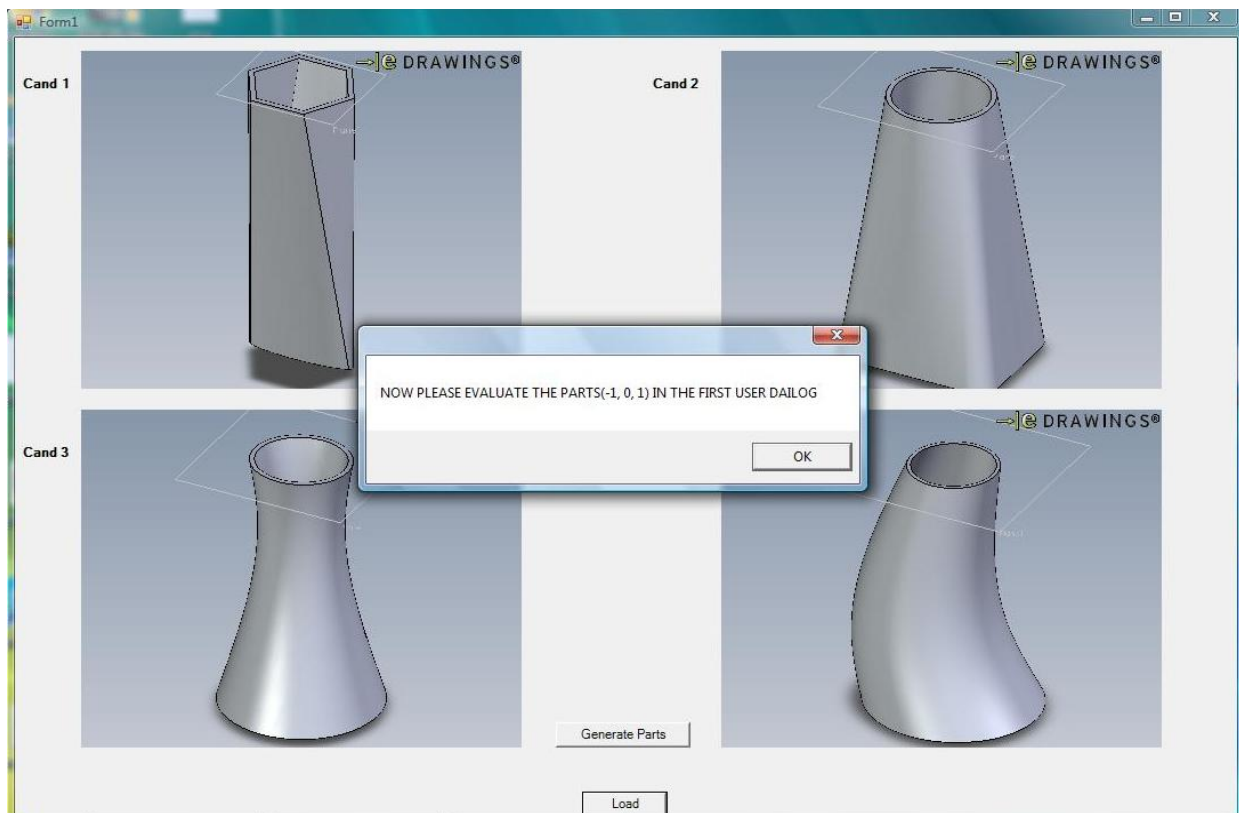


Figure 3. GUI window displaying four models side by side for visual comparison

The user compares these 4 designs displayed on the screen and gives his/her feedback taking into account features such as aesthetics, ergonomics, size, among others, using a comparer window as shown in Figure 4.

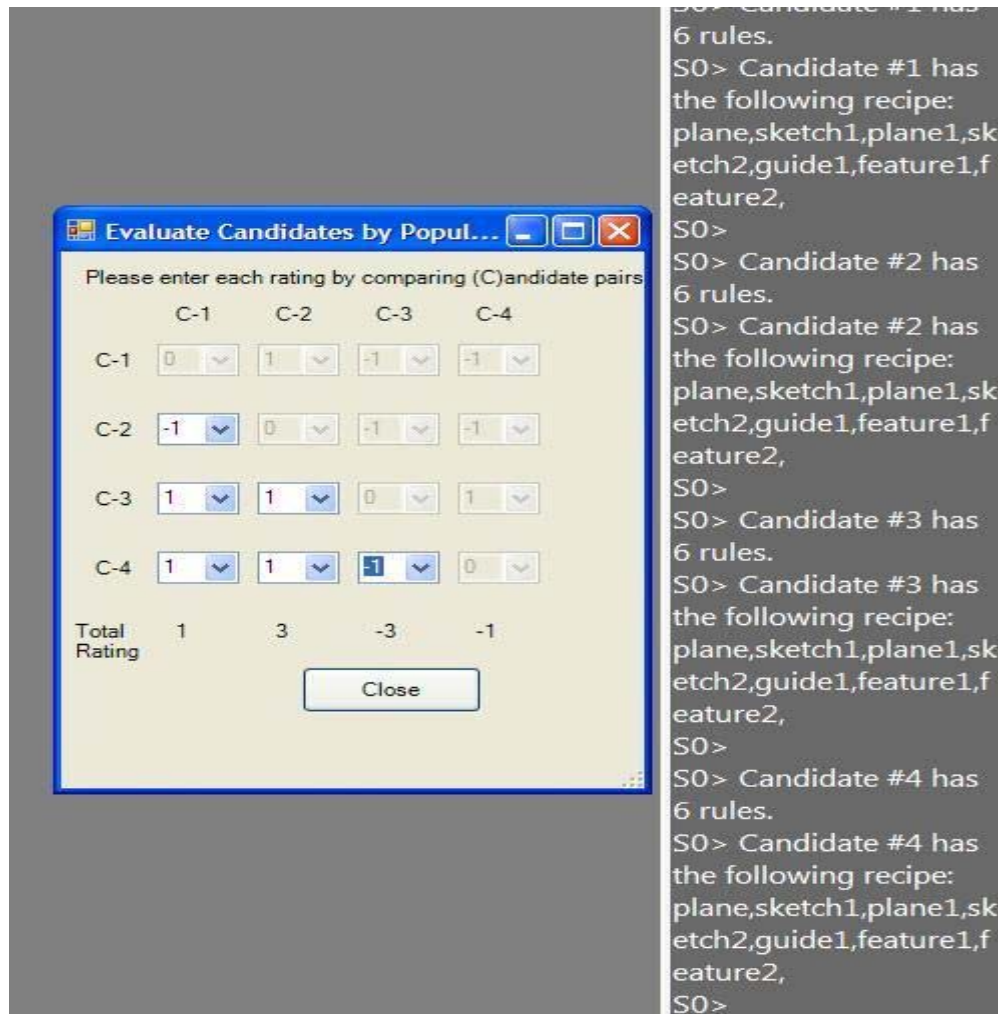


Figure 4. Comparer window in which rating is given by comparing candidate pairs.

User feedback is used along with computer program to develop an algorithm. This human interactive algorithm gives importance to the rules and the order of the rules in a user preferred design. This data is used by the human interactive program to randomly generate a new optimal design close to the user's preference.

4.4 Optimization

The optimization part involves two steps: Rule selection and parameter optimization. The rule selection can be further divided as exploration and exploitation depending on the designer's need. Exploitation is giving more preference to the rule which is selected most number of times in the previous generations. This is nothing but the selection of the rule which has high fitness (U-fit), fitness is the cumulative rating calculated from the user feedback. In the case of exploration the rule that is least selected in the previous generations has much higher chance of getting selected in the present generation. This is nothing but the selection of the rule which is least popular (Upop). The combined user preference (U) is calculated according to Rai's [16] method used in neck-tie grammar rule selection work as shown below, where 'B' is a knob to choose between exploration and exploitation in rule selection process.

$$\text{Combined Preference } U = B * U_{\text{fit}} + (1-B) * U_{\text{pop}},$$

The parameter optimization is a process of optimizing the values given to each of the parameters or variables at each of nodes involved in the path to candidate. Each variable's value is in a certain range and in the process of optimizing we are moving towards the best value in that range according to the Stochastic Hill Climbing with Learning by Vectors of Normal Distributions (SHCLVND) algorithm [18]. The key purpose of the SHCLVND algorithm is to generate more and more better values for the

parameters in each successive generation by using normal distribution for the probabilities.

The pseudo code explaining the whole process of optimization is shown in the following figure.

```

FOR N Iterations
  STEP 1: FOR M Iterations (Presenting M solutions for user input)
    STEP 1.1: IF  $N \neq 1$  (i.e. if it is the first iteration, there is no user feedback yet)
      FOR each option in the identified set of valid transitions at a particular state
        STEP 1.1.1: Identify the production rules for the option
        STEP 1.1.2: Compute rule fitness
        STEP 1.1.3: compute rule popularity
        STEP 1.1.4: compute  $U(\text{fit})$ 
        STEP 1.1.5: compute  $U(\text{pop})$ 
        STEP 1.1.6: compute combined preference,  $U$ 
        STEP 1.1.7: compute option probability,  $P(U)$ 
      ENDFOR
    STEP 1.2: ELSE
      STEP 1.2.1: Select a random option in the identified set of valid transitions at a
        particular state
    STEP 1.3: FOR each parameter involved in the selected option with best  $P(U)$ 
      STEP 1.3.1: Compute the parameter value based on the  $\mu$  and  $\sigma$ 
    ENDFOR
  ENDFOR
  STEP 2: Given user input create knowledge nuggets
  STEP 3: FOR each rule in the set of rules in the best fit candidates collection
    STEP 3.1: Identify all the parameters involved
    STEP 3.2: Optimize the  $\mu$  value of all parameters
    STEP 3.3: Optimize the  $\sigma$  value of all parameters
  ENDFOR
ENDFOR

```

Figure 5. Pseudo Code showing parameter optimization and rule selection involved in each generation

According to the SHCLVND algorithm by Stephen and Koppen [18], the values for the parameters in a given range are generated as follows

For a given R-max (maximum value of that range), R-min (lowest value of that range), Mu-move (the learning rate), Sigma-reduce (the narrowing of search space with each generation) and Const-range2sigma (scaling factor for initial Sigma computation) the Mu and Sigma values for each parameter are calculated using the below equations [18].

$$\vec{r}_{range} = \vec{r}_{max} - \vec{r}_{min}$$

$$\vec{\mu} = \vec{r}_{min} + \frac{\vec{r}_{range}}{2}$$

$$\vec{\sigma} = \vec{r}_{range} * \overline{Const_{range2sigmaFactor}}$$

$$\vec{b}_{middle} = \frac{1}{|B|} \sum_{\vec{b} \in B} \vec{b}$$

$$\vec{\mu} = \vec{\mu} + \mu_{move} (\vec{b}_{middle} - \vec{\mu}),$$

$$\vec{\sigma} = \vec{\sigma} * \sigma_{reduce}$$

After having updated Mu and Sigma, an optimal random number is calculated as below

$$R_{opt} = (R_{norm} * \text{Sigma}) + \text{Mu}$$

where Rnorm is a normal distributed random number calculated according to Box-Muller formula [16].

5. Results

Using just the random candidate generator part of the project with no optimization of rule selection and parameter selection has yielded a lot of good and some weird shapes in the shampoo bottle domain. Some of the screen shots of the bottle designs generated are given in Figure 6.

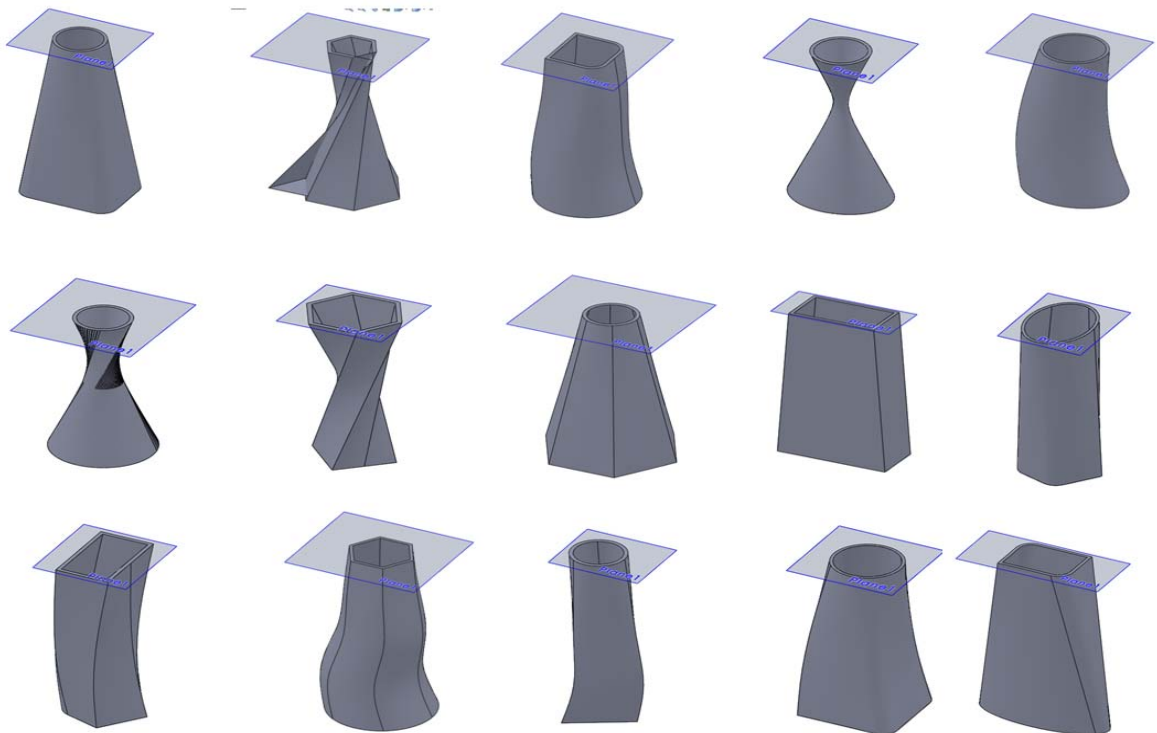


Figure 6. Bottle designs developed using Random Candidate Generator.

The parameter optimization part of the project looks promising; the change in the parameters values are not significant enough for human eyes to notice but the numerical values shows a sufficient shift in the values of the parameters that can be significant

enough for many other domains. Table 2 presents the probabilities assigned to each of the 29 rules in the shampoo bottle grammar, after the end of an iteration based on the user feedback. The higher the probability of the rule, the greater the chance of it getting selected when presented with an option of selection between the siblings in the search tree. Certain rules have in their fields NaN values in the table means Not a Number i.e., the search process has never got the chance to visit these rules in these iterations.

	Iteration 1	Iteration 2	Iteration 3
Rule 1	0.015053	0.9599374	0.99682
Rule 2	0.015053	2.4435739	1.57E-16
Rule 3	0.95484	0.0400624	0.00277
Rule 4	0.015053	2.27E-07	0.00041
Rule 5	0.5	0.9989066	0.99891
Rule 6	0.5	0.5	0.5
Rule 7	1	1	1
Rule 8	1	1	1
Rule 9	NaN	1	1
Rule 10	0.5	0.0010934	0.00109
Rule 11	0.5	0.5	0.5
Rule 12	0.05472	0.0075907	0.00271
Rule 13	0.83584	0.9848128	0.34119
Rule 14	0.05472	0.0075907	0.65521
Rule 15	0.05472	5.84E-06	0.0009
Rule 16	0.333333	7.65E-05	0.49898
Rule 17	0.333333	0.3333333	0.33333
Rule 18	0.98448	1	1
Rule 19	0.01552	5.65E-34	5.65E-34
Rule 20	NaN	0.1357994	1.67E-10
Rule 21	NaN	0.8642006	0.98888
Rule 22	0.333333	7.65E-05	0.49898
Rule 23	0.333333	0.9998469	0.00203
Rule 24	0.333333	0.3333333	0.33333
Rule 25	0.333333	0.3333333	0.33333
Rule 26	NaN	NaN	NaN
Rule 27	NaN	NaN	NaN
Rule 28	1	1	1
Rule 29	1	1	1

Table 2. Table showing the probability values for the rules at the end of the iterations

Table 3 presents the mean (Mu) and standard deviation (Sigma) values of the parameters at the end of iterations in the design process. After the end of iteration, based on the user feedback, the Mu and the Sigma values for each parameter in the best rated candidate only gets modified, so that the interval in which the parameter's value resides gets condensed towards an optimal value i.e. the Mu value which is nothing but the mean of the normally distributed interval will move either way (small or big) towards the optimal value for that parameter where as the Sigma, the standard deviation will get reduced each time a little bit to concentrate around the optimal value for that parameter

	Iteration 1		Iteration 2		Iteration 3	
	μ	σ	μ	σ	μ	σ
Circle	3	1	3.02742657	0.95	3.0274	0.95
RectangleX	3	1	3	1	2.9521	0.95
RectangleY	3	1	3	1	3.0289	0.95
Ploygon	3	1	3	1	3	1
PloygonS	5	2	5	2	5	2
EllipseX	3	1	3	1	3.0222	0.95
EllipseY	3.5	1.5	3.5	1.5	3.464	1.425
Circle2	5	1	5	1	4.9872	0.95
Rectangle2X	4	2	3.94269168	1.9	3.9427	1.9
Rectangle2Y	4	2	4.09295199	1.9	4.093	1.9
Polygon2	4.5	1.5	4.5	1.5	4.5	1.5
Polygon2S	5	2	5	2	5	2
Ellipse2X	5.5	1.5	5.5	1.5	5.5	1.5
Ellipse2Y	4.5	2.5	4.5	2.5	4.5	2.5
refPlane	17	3	16.94269	2.85	16.993	2.7075
Splinew	5	3	5	3	4.97	2.85
SplinePts	2.5	1.5	2.5	1.5	2.475	1.425
SplineMir	0.5	0.5	0.5	0.5	0.0122	0.475
Shell	3	1	2.8677426	0.95	2.7403	0.9025

Table 3. Table displaying the Mu and Sigma values for each of the parameter intervals

6. Conclusion

This project presented a novel approach for automating and optimizing the generation of conceptual design process with the help of existing tools such as Solid Works, Graphsynth and a proven optimization algorithm SHCLVND. Generation of conceptual designs involve development of 100's of iterative designs and is very complicated and time consuming to carry out manually, so automation of concepts is extremely significant in engineering design. There are some previous works regarding the same which give a random model in some graph or mathematical representations, where users can neither properly visualize the end-concept nor give a correct feedback. Compared to existing automation projects, this project presented users/designers with not just some mathematical representations of a concept but an instant 3D model. Such a model enabled users to interact and compare to other concepts to provide a feedback, which is then used to influence the generation of future concepts. This project is experimented with a shampoo bottle example and the experimental results shows that the concepts are consistent with users' preference.

This project can be further extended by introducing some more complex constraints. For example, this shampoo bottle domain can be improved by introducing new constraints in terms of volume. Due to the integration with COM software and ActiveX components the application is crashing some times during the generation

process but as there is no other option for working together with a CAD software and GraphSynth, the robustness of the application is sacrificed for now.

References

- [1] Agarwal, M. and J. Cagan (1998). “A blend of different tastes: the language of coffeemakers.” *Environment and Planning B: Planning and Design* Vol 25(No. 2) pp. 205-226.
- [2] Andy Schurr Introduction to Progress, an attribute graph grammar based specification language.
- [3] Campbell, M. I., 2006. The official GraphSynth Site, <http://www.graphsynth.com>, University of Texas at Austin.
- [4] Cagan, J., 2001, “Engineering Shape Grammars”, *Formal Engineering Design Synthesis*, Antonsson, E. K., and J. Cagan, eds., Cambridge University Press.
- [5] Chandrasekaran, B., 1990, “ Design problem solving: a task analysis” AI Magazine, Winter 1990: pp.59-73.
- [6] *Com interoperability*. (2005). Retrieved from <http://msdn.microsoft.com/en-us/library/bd9cdfyx.aspx>
- [7] Dorina C. Petriu and Hui Shen Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications
- [8] *Dassault systemes*. (2008). Retrieved from <http://www.3ds.com/>
- [9] *Engineering design process*. (2009). Retrieved May 12, 2010 from Wikipedia: http://en.wikipedia.org/wiki/engineering_design_process.

- [10] *Edrawings*. (2008). Retrieved from <http://www.edrawingsviewer.com/>
- [11] X. Du, J. Jiao, M. M. Tseng Product Platform Representation: A Graph Grammar Approach
- [12] Linq to xml. (2007). Retrieved from <http://msdn.microsoft.com/en-us/library/bb387098.aspx>
- [13] Maher, M. L., 1988, "Engineering Design Synthesis: A Domain-Independent Representation", Artificial
- [14] Pahl, Gerhard, Beitz, W., Wallace, Ken, Feldhusen, J., & Blessing, Lucienne. (2007). Engineering design. Springer Verlag.
- [15] *Pugh, S., 1991, Total Design: Integrated Methods for Successful Product Engineering. Addison-Wesley Publishing Company, Workingham, UK*
- [16] Rai, R., Kurtoglu, T., and Campbell, M., 2009,"Stochastic interactive graph grammar search for conceptual design" ASME Journal of Mechanical Design (Submitted).
- [17] Rozenberg, Grzegorz, Division, British, Bibliography, Council, & Hirsch, Martin. (2008). Graph Grammars and Computing by Graph Transformation. Logos Verlag Berlin GmbH.
- [18] Rudlof, S., Koppen, S., 1997,"Stochastic Hill Climbing with Learning by Vectors of Normal Distributions", IPK, Berlin.
- [19] *Semet, Y., 2002, "Interactive Evolutionary Computation: A Survey Of Existing Theory", University of Illinois.*

- [20] Saaty, T., 1980. *The Analytic Hierarchy Process*, McGraw-Hill
- [21] *The .net framework*. (2002). Retrieved from <http://www.microsoft.com/net>
- [22] Troelsen, A. (2003). *C# and the .NET Platform*. Berkely, CA: Apress.
- [23] Troelsen, A. (2005). *Pro C# 2005 and the .NET 2.0 Platform*. Berkely, CA: Apress.
- [24] Ullman.D., 1995.*The Mechanical Design Process*, NewYork:McGraw-Hill
- [25] *Visual c# developer center*. (2002). Retrieved from <http://msdn.microsoft.com/en-us/vcsharp/default.aspx>
- [26] Zahed Siddique David W. Rosen Product Platform Design: A Graph Grammar Approach.