Separation of Concerns: 
strategies for complex parametric design modelling

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Alexander Peña de Leon
B.Arch

School of Architecture and Design
Design and Social Context Portfolio
RMIT University
March 2014
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>THESIS STATEMENT AND AIMS</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>RESEARCH MOTIVATION</td>
<td>8</td>
</tr>
<tr>
<td>1.3</td>
<td>DESIGN MODELING INFLEXIBILITY</td>
<td>11</td>
</tr>
<tr>
<td>1.4</td>
<td>RESEARCH CONTEXT</td>
<td>13</td>
</tr>
<tr>
<td>1.5</td>
<td>RESEARCH STRUCTURE</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>STATE OF PLAY</td>
<td>17</td>
</tr>
<tr>
<td>2.1</td>
<td>BACKGROUND</td>
<td>17</td>
</tr>
<tr>
<td>2.1.1</td>
<td>THE INDUSTRY</td>
<td>17</td>
</tr>
<tr>
<td>2.1.2</td>
<td>RATIONALISATION</td>
<td>32</td>
</tr>
<tr>
<td>2.1.3</td>
<td>SUMMARY OF BACKGROUND</td>
<td>36</td>
</tr>
<tr>
<td>2.2</td>
<td>THE PHENOMENON</td>
<td>38</td>
</tr>
<tr>
<td>2.3</td>
<td>THE GRAPH</td>
<td>57</td>
</tr>
<tr>
<td>3</td>
<td>STANDARD APPROACH</td>
<td>69</td>
</tr>
<tr>
<td>3.1</td>
<td>INTRODUCTION</td>
<td>69</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>3.2</td>
<td>INTERNAL CODE STRATEGY</td>
<td>73</td>
</tr>
<tr>
<td>3.2.1</td>
<td>OVERVIEW OF INTERNAL CODE STRATEGY</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>ADVANTAGES OF INTERNAL CODE STRATEGY</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>LIMITATIONS OF INTERNAL CODE STRATEGY</td>
<td>76</td>
</tr>
<tr>
<td>3.2.2</td>
<td>SAGRADA FAMILIA HYPERBOLIC BRIDGE</td>
<td>79</td>
</tr>
<tr>
<td>3.2.3</td>
<td>SAGRADA FAMILIA PASSION FAÇADE</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>PROJECT DESCRIPTION</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>FORENSIC PARAMETRIC MODEL</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>DIRECT MODELING</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>DISCUSSION OF SAGRADA FAMILIA PASSION FAÇADE</td>
<td>88</td>
</tr>
<tr>
<td>3.2.4</td>
<td>SUMMARY OF INTERNAL CODE STRATEGY</td>
<td>91</td>
</tr>
<tr>
<td>3.3</td>
<td>EXTERNAL CODE STRATEGY</td>
<td>92</td>
</tr>
<tr>
<td>3.3.1</td>
<td>OVERVIEW OF EXTERNAL CODE STRATEGY</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>ADVANTAGES OF EXTERNAL CODE STRATEGY</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>LIMITATIONS OF EXTERNAL CODE STRATEGY</td>
<td>95</td>
</tr>
<tr>
<td>3.3.2</td>
<td>YAS ISLAND</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>YAS ISLAND PROJECT DESCRIPTION</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>WIREFRAMES</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>INDEXING AND NAMING CONVENTION</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>COMPUTATION</td>
<td>113</td>
</tr>
<tr>
<td></td>
<td>SUMMARY OF YAS ISLAND</td>
<td>117</td>
</tr>
<tr>
<td>3.3.3</td>
<td>MUSEO SOUMAYA</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>MUSEO SOUMAYA PROJECT DESCRIPTION</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>DESIGN PROBLEM</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>SPHERE-PACKING</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>CLUSTERING</td>
<td>133</td>
</tr>
<tr>
<td>3.3.4</td>
<td>FABPOD</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>FABPOD PROJECT DESCRIPTION</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>AUTOMATION WORKFLOW</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>FILE TRANSFER PROTOCOL</td>
<td>145</td>
</tr>
<tr>
<td></td>
<td>ROBUST USER FEATURE</td>
<td>146</td>
</tr>
<tr>
<td></td>
<td>TECTONICS</td>
<td>147</td>
</tr>
<tr>
<td>3.3.5</td>
<td>GLORY FACADE HELICAL STAIR</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>GLORY FACADE HELICAL STAIR PROJECT DESCRIPTION</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>DESIGN PROBLEM</td>
<td>154</td>
</tr>
<tr>
<td></td>
<td>FLEXIBLE COMPONENTS</td>
<td>156</td>
</tr>
</tbody>
</table>
CONTENTS

RHUMBLINE ............................. 157
THE STAIR STEP ........................... 158
INSTANTIATION AND CONFIGURATION .... 159
3.3.6 SUMMARY OF EXTERNAL CODE STRATEGY .............. 165

4 BESPOKE APPROACH 167
4.1 STANDARD LIBRARY ....................... 168
4.1.1 OVERVIEW ............................. 168
ADVANTAGES .............................. 173
LIMITATIONS .............................. 174
4.1.2 THE DERMOID ......................... 177
PROJECT DESCRIPTION ................. 177
UV SPACE MAPPING ..................... 179
SPHERE PACKING ....................... 181
DYNAMIC RELAXATION ................. 183
4.2 STANDARD LIBRARY WRAPPER .......... 193
4.2.1 RESPONSIVE ACOUSTIC SURFACING ............... 200
PROJECT OVERVIEW .................... 200
HYPERBOLOID PATTERNING ............ 201
4.3 SCRIPTING INTERFACE ................. 207
4.4 SUMMARY OF BESPOKE APPROACH ............ 219

5 SUMMARY OF FINDINGS 223
5.1 OVERVIEW .............................. 224
5.2 UNDERSTANDINGS OF THE PROBLEM ............. 224
5.3 EVALUATING INTERNAL AND EXTERNAL CODE SOLUTIONS ........ 227
5.4 THE PLACE OF ALTERNATIVE BESPOKE SOLUTIONS ....... 236
5.5 SUMMARY .............................. 239

6 CONCLUSIONS 243

REFERENCES 249

GLOSSARY 263

A Appendix 1
A.1 SAGRADA FAMILIA HYPERBOLIC BRIDGE ............. 1
CONTENTS

A.1.1 Reaction:Build_Panelproxies ........................................... 2
A.1.2 Reaction:Build_InstantiatePanels ................................... 9
A.1.3 Reaction:Build_MakeBodies ........................................... 16
A.1.4 Reaction:Build_SolveSplit ........................................... 18
A.2 FABPOD ................................................................. 25
A.2.1 Fabpod_GUI ......................................................... 25
A.3 DERMOID ............................................................... 33
A.3.1 Pattern Maker ....................................................... 33
A.3.2 Relax Mesh Node .................................................... 39
A.3.3 Dynamic Relax Mesh Node ......................................... 45
A.3.4 Dynamic Relax Mesh Graph ....................................... 51
A.4 SCRIPTING INTERFACE ............................................... 58
A.4.1 Sample Scripts ..................................................... 59
A.4.2 Script Widget ...................................................... 64
A.4.3 HybridShape Interface ............................................ 71
A.4.4 ShapeFactory ....................................................... 78
List of Figures

Figure 1.1  Cadenary Tool, developed by Axel Kilian. This custom bespoke tool was developed using the Processing language for the design of funicular structures inspired by the inverted hanging models of Antoní Gaudí. 11

Figure 3.1  Three approaches to parametric modelling. Indicating which areas of the design logic are maintained by either a scripter or an end-user, as well as who maintains topological variations in the parametric model. 72

Figure 3.2  Screen capture from CATIA’s™ knowledgeware reaction code editor. As it can be seen from this capture, the code editor is quite basic. 77

Figure 3.3  Visual Studio code editor showing the auto-complete drop-down menu, facilitating the selection of a CATIA function. 78

Figure 3.4  Sagrada Família Basilica Hyperbolic Bridge responsive parametric model. In this figure the user control parameters are shown, providing the user interface to the tool. 79

Figure 3.5  Sagrada Família Basilica Bridge Variants. This figure indicates different variants obtained from simply changing parameters in the user-control inputs. Because of the use of “Reactions” the system can produce extreme topology variations. 81

Figure 3.6  Photograph of part of Gaudi’s last known drawing for the facade taken in 1917 from a photographic plate surviving after the destruction of the original drawing. 86
<table>
<thead>
<tr>
<th>Figure 3.7</th>
<th>The Reaction Based System for the Passion Façade. This figure shows in the CATIA specification tree the levels of control which where added to the model to provide “responsiveness”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3.8</td>
<td>An Elevation drawing of the Yas Island Digital Mock-up. The three regions rendered above are zoom levels of the same drawing. As the Zoomlevel increases, we can see the high levels of detail and resolution at which the Yas Island Digital Mock-up represents the design intent</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>Wireframe geometry of the YAS Island Marina project. This image shows the node points, node normals and beam elements. Different beam types are indicated by the difference in colour.</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>The detailed bottom-up assemblies (top) were automatically instantiated dressing-up the wireframes (bottom). The assembly on the right is a sample of the result of executing the code on 12 panels.</td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>The wireframe shown left was built from mounting curves and points on the rationalised surfaces (top left), the instantiated beams and nodes dressing up the wireframe (right).</td>
</tr>
<tr>
<td>Figure 3.12</td>
<td>A non exhaustive catalogue of the various re-usable parametric models of the YAS ISLAND gridshell.</td>
</tr>
<tr>
<td>Figure 3.13</td>
<td>Indexing and Nomenclature system for the YAS Island Marina. The system indicates how panels, beams and nodes all share a common naming system which indicates where within the gridshell the component is located.</td>
</tr>
<tr>
<td>Figure 3.14</td>
<td>A photograph of the completed Museo Soumaya as seen from street level.</td>
</tr>
<tr>
<td>Figure 3.15</td>
<td>Comparison of Museo Soumaya Digital Mock-up with real-word Mock-up</td>
</tr>
<tr>
<td>Figure 3.16</td>
<td>An exploded axonometric indicating the various layers involved in the metal cladding façade of the Museo Soumaya.</td>
</tr>
<tr>
<td>Figure 3.17</td>
<td>Diagram indicating the relation between the diameter of spheres, the gap and the hexagons inscribed.</td>
</tr>
<tr>
<td>Figure 3.18</td>
<td>Diagram illustrating the polar furrow concept described by Darcy Thompson in the book “On Growth and Form”</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 3.19  Sphere-Packing algorithm for developing the circle mesh grid from which to inscribe hexagons. .................................................. 128

Figure 3.20  Screenshot of the sphere-packing tool overlay. This system provides a user interface developed with VisualBasic™ to execute a sphere-packing pattern over any doubly curved surface within Digital Project™ ............................... 129

Figure 3.21  An example of a circle-mesh grid executed on the lower half of the Museo Soumaya Façade. ............................................................... 130

Figure 3.22  Hexagon mesh before stretch (top) Hexagon mesh after stretch (bottom). 131

Figure 3.23  Diagram indicating the filtering process of standard versus custom panels and how the custom panels are clustered into families by measuring the panel-to-panel differences. ......................................................... 132

Figure 3.24  Hexagonal mesh stretching process. Indicating how rails are first mounted on the squashed panels (left) and how the panels are vertically stretched while mounted on the rails, leaving the width of the panel unchanged. This vertical stretching transforms the squashed hexagons into uniform hexagons (right). ....... 132

Figure 3.25  Two optimisation routines executed on the Museo Soumaya. First, to identify the origin of the sphere-packing (top) by finding the flattest point on the waist of the facade. Second, to determine the best sphere radius for the sphere-packing (bottom). ................................................................. 134

Figure 3.26  Example of “Lloyd” K-means clustering code for the statistical programming language R ............................................................. 136

Figure 3.27  21 family k-means clustering result. This clustering uses the area as part of the calculation and thus produces a banding effect. ............... 136

Figure 3.28  49 family k-means clustering result. This clustering does not use the area as part of the calculation and thus produces a more patch-like result. This result is more faithful to the original pattern in comparison to other family distributions. 137

Figure 3.30  Blowup screenshot of the FABPOD level cell to cell complexity. This figure indicates the internal complexity of the fabrication cells of the FABPOD. .. 141
Figure 3.31 The FABPOD automation workflow. This diagram shows how wireframe definitions from Grasshopper™ flow to Digital Project™. In Digital Project™ these wireframe definitions increase in resolution and are made ready for machining as solid models. ................................................................. 142

Figure 3.32 FABPOD custom user interface to Digital Project™, for the instantiation of the product model assembly. ................................................................. 144

Figure 3.33 An example of the custom CSV file format developed to transfer information from Grasshopper™ to Digital Project™. Each line in the file represents a cell, each wall was represented as a single text file. ................................. 145

Figure 3.34 Assembly product for Wall003. This figure shows the assembly complexity of a single wall of the FABPOD. In this image one of the fabrication cells is highlighted ................................. 148

Figure 3.35 FABPOD fabrication. This figure shows the complexity of a single fabrication unit, and how the timber frames must be mitred and bolt holes inserted, as well as scoop holes removed from the timber on each fabrication cell to facilitate hand assembly ................................. 149

Figure 3.36 Timber frame geometry. This figure indicates the complexity of the timber frame geometry. The timber frame is trimmed by many cutting features, including the hyperboloid front facing panel, the two adjacent timber frames, and the fabrication cell back panel. The green surface shown above is a linear swept surface approximating the hyperboloid for machining purposes. This surface is also at an offset distance from the hyperboloid panel to accommodate for fitting tolerances during assembly. ................................. 150

Figure 3.37 Unfolded fabrication cells of the FABPOD project. The automated workflow of the FABPOD project simultaneously creates unfolded data as it creates the 3D assembly units. ................................. 152

Figure 3.38 Elevation screenshot of the Sagrada Familia Basilica Glory Façade Helical Spiral Staircase in context. ................................. 153

Figure 3.39 Diagram indicating the geometrical differences between a between an ellipsoid and a cylinder. ................................. 156
LIST OF FIGURES

Figure 3.40 Rhumbline wireframe system on the left, indicating how the rhumbline preserves the angle of bearing as it wraps along the ellipsoid shaft. On the right the step geometry is mounted directly on top of the rhumbline, guiding the steps around the ellipsoid shaft. .............................. 158

Figure 3.41 Geometry of the step for Sagrada Família Basilica Glory Façade Helical Spiral Staircase. The step is a solid model, trimmed between the two hyperboloids of the stairwell. The bottom side of the step is a helicoid surface. ........ 159

Figure 3.42 Sagrada Família Basilica Helical Spiral Staircase User Interface. This figure indicates the script overlay develop for the instantiation of the stair system. 162

Figure 3.43 Sagrada Família Basilica Helical Spiral Staircase Excel Configuration File. This figure indicates the Excel configuration files together with the user interface to instantiate stair variants. ............................................. 163

Figure 3.44 VisualBasic code for the propagation of the Sagrada Família Basilica Helical Spiral Staircase. This system takes the User-defined features created by hand, and instantiates them based on algorithmic logic and the configuration from Excel. ............................................................... 164

Figure 3.45 Sagrada Família Basilica Helical Spiral Staircase design variant. This figure shows a design variant instantiated from the scripting overlay tool. ... 164

Figure 4.1 Reciprocal frame-truss based on pentagon distributions, the geometry was wrapped on the UV space of the surface. This figure indicates how the pattern degenerates near the poles of the surface. ......................... 180

Figure 4.2 Example of sphere-packing the Dermoid surface. As the circles step away from the initial row, the circle-mesh deviates from a uniform distribution... 181

Figure 4.3 Pseudo algorithm for the dynamic relaxation of two-dimensional meshes on curved surfaces. ......................................................... 185

Figure 4.4 Bespoke application developed for the Dermoid project to compute the dynamic relaxation over the ellipsoid. This tool is a node-based parametric model which enables importation of any two-dimensional mesh design from a custom CSV file format and interactive manipulation of the relaxation process. ........ 191
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>Dynamic Relaxation Process of pattern one. This figure shows different time snapshots of the dynamic relaxation process of the Dermoid in 3D space (bottom) and UV space (top). The pattern begins in a distorted state, as the process converges, the pattern becomes uniform, removing the distortion at the “poles”.</td>
</tr>
<tr>
<td>4.6</td>
<td>Dynamic Relaxation Process of pattern two. This figure shows different time snapshots of the dynamic relaxation process of the Dermoid in 3D space (top) and UV space (bottom). With this pattern the process of relaxation becomes more evident.</td>
</tr>
<tr>
<td>4.7</td>
<td>Example of Wrapped C++ code and Non Wrapped C++ code.</td>
</tr>
<tr>
<td>4.8</td>
<td>Example of Meta-Programming, showing the differences between C++ normal syntax and the syntax extension provided by creating a meta-program.</td>
</tr>
<tr>
<td>4.9</td>
<td>The workshop space and participants of the Smart Geometry 2011 at CITA in Copenhagen.</td>
</tr>
<tr>
<td>4.10</td>
<td>Sample application of Ben Coorey’s image based bespoke software developed for the Smart Geometry 2011 “Responsive Acoustic Surfaces” cluster. The custom interface on the left was developed using the Qt GUI editor.</td>
</tr>
<tr>
<td>4.11</td>
<td>An earlier version of Ben Coorey’s image based bespoke software (Top) and the result of the Digital Project boolean operations (Bottom).</td>
</tr>
<tr>
<td>4.12</td>
<td>Diagram indicating the slider-variable relation between sliders in the slider-deck and variables in the computer code.</td>
</tr>
<tr>
<td>4.13</td>
<td>The Scripting Integrated Development Environment, showing the front-end sliders.</td>
</tr>
<tr>
<td>4.14</td>
<td>The Scripting Integrated Development Environment, showing the back-end scripting code.</td>
</tr>
<tr>
<td>4.15</td>
<td>Diagram indicating the successive abstraction concept. Script command “make-circle” is an abstraction of the C++:Binding Code “make3pointcircle” which is an abstraction of the C++:Wrapper “AddNewCircle”.</td>
</tr>
</tbody>
</table>
ABSTRACT

This thesis considers the extent to which conventional parametric technologies *de-augment*\(^1\) the traditional process of design delivery. The design delivery process aided with parametrically enabled technologies can allow architectural practices to manage higher levels of complexity, stricter design guidelines and tighter design schedules compared to the design delivery process unaided by parametric technologies. But while the benefits of these technologies are higher than their drawbacks, this enhanced delivery process is not devoid of risks and imperfections. This thesis deals with strategies to manage these *inflexibilities* through the use of existing technology and knowledge available to design professionals today.

The adoption of parametrically enabled technologies by design professionals is a necessary step into advancing design practice to more complex project work, but the adoption of these technologies must be taken with caution.

\(^1\) coined within the field of Human Computer Interaction by Douglas Carl Engelbart director of the Augmentation Research Center, inventor of the mouse and the precursor to the graphical user interface as we know it today. Who was inspired by Vannevar Bush’s article “As We May Think”.

tice has the potential to introduce an unwanted creative “bottleneck” during crucial decision making moments of an advanced architectural project.

For these reasons parametric design modelling alone will not be sufficient in supporting designers through the delivery process of geometrically complex buildings. Parametric design software and its associated design protocol impede architects in the flexible management of changes throughout the lifetime of a project. This thesis will explore and compare two strategies to overcome the inflexibility of parametric modelling systems. These strategies have been developed through the use of a mixed-methods research methodology which combines the action research method with the case-study method.
DECLARATION

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

Alexander Pena
28.03.2014
ACKNOWLEDGEMENTS

Throughout the 4 year span of this thesis, I have benefited from the support of many individuals and institutions. I would like to thank all those who have directly or indirectly contributed to the work presented and documented in this thesis.

I would like to thank firstly my supervisors Jane Burry and Mark Burry for showing me by example how to become a better scholar and practitioner. For giving me this unique opportunity and for being there through thin and thick throughout these 4 years. Thank you as well for allowing me to participate in the Sagrada Familia Project.

To my dearest friend Julian Canterbury and his partner Lucette Blum, thank you for introducing me to Australia and convincing me to pursue the PhD.

To Nicola Narayan, Brad Marmion, Margaret Woods thank you for making the ride less bumpy.

To my colleagues, Dr. Daniel Davis, Dr. Sascha Bohnenberger, Dr. Chin Koi Khoo, Dr. Kamil Sharaidin, Rafael Moya, Flora Salim, Andrew Burrow, Andy Miller and Michael Wilson, for pushing the boundaries of what is possible, thank you for all those wonderful discussions and alternative views of the world.

To my friends, Jason Sidelko, Eddy Slim, Herminio Gonzalez Huizar, John
Klein, Chikara Inamura, thanks.

To the Australian Research Council and SIAL for funding my research.

To Gehry Technologies Inc, for the unique opportunity of allowing me to work on the Yas Island Marina project and the Museo Soumaya Project. To Dennis Shelden, Sameer Kashyap, Rolando Mendoza, Lee Warren, Jason Sidelko, Eddy Slim, Kayla Farrell, Nicholas Pisca, Neil Meredith, Neil Thelen, Susan Constantin, Santino Medina, Lionel Lambourne, Ahmed Reyad, James Kotronis, Jesse Louis Rosenberg, Nicola Davolio, Tim Porter, Victor Keto, Alan Hudin, Chi Ng, thank you all, I have learned so much from all of you.
To CITA at the Royal Danish Academy of Fine Arts, thank you for inviting us to Denmark and for allowing us to collaborate with your team. To Mette Ramsgard Thomsen, Martin Tamke, Phil Ayres, Brady Peters, Jacob Riiber, Tore Banke, Anders Holden Deleuran, Stig Anton Nielsen, Aron Fidjeland, thank you.

To Dr. Rafael Torres, Dr. Luz Hernandez and Rafael Torres Hernandez thank you for your support.

To my mother (Mally M. de Leon Adames), for inspiring us to work hard. To my father (Rodolfo R. Pou Martinez) and my brother (Leamlly Peña de Leon), who inspired me to follow their footsteps into architecture. To my grandmother (Altagracia Lopez) for her unconditional love, and for giving us all, the opportunity to travel and explore the world. To my siblings (Amaly Abreu, Rosanna Abreu, Leandro Pou, Camila Pou) thank you for your love and support. To my aunts and uncles (Bierka Martinez, Mercedes Nicasio, Sofia Adames, Maria Rosario, Rene Garcia, Jose Martinez, Ramon Rosario, Oriol de Leon, Elizabeth Garcia) thank you for all the support. To my Cousin Rebecca Pou, thank you. To Lourdes Cuervo Pou, thank you.

And lastly to my dearest and beautiful wife (Marcela Torres Hernandez), this thesis would not have been possible without you. Thank you for your patience and support.
Chapter 1

INTRODUCTION

My research considers the extent to which the parametric design paradigm can accommodate disruptive change within flexible design environments.

My central research proposition is that a strategic use of support technologies to aid the parametric modelling paradigm help designers overcome inflexibility and greatly enhance their flexible design modelling environments.

I have developed my thesis around eight longitudinal case-studies. All of the case-studies analysed in this thesis have used a parametric model to help designers conceive enhanced iterations of design, with an increased throughput. The parametric models have been used to transform initial designs into detailed designs, and with this enabling the physical realisation of these complex designs by facilitating the search of solutions to these challenging design problems. Therefore the case-studies presented in this thesis demonstrate how precisely design environments can be improved under complex design contexts and how designers can enhance the design environment through both the design of technological developments and the effective application of strategic practices.
The case-study projects selected for this thesis present differing scenarios where the complexity within each project challenged designers and challenged the flexibility of parametric design models. This thesis acknowledges that while the parametric model is crucial in the physical realisation of the case-studies presented, the technology and its inherent design process suggest improvement. This thesis addresses how designers can effectively improve the flexibility of parametric modelling environments in supporting complex design project work and thus enabling an enhanced flexible design environment. The research addresses how designers can accomplish enhancements to parametric modelling technologies to enable a higher-level of control and therefore flexibility in design modelling. It will also answer the question of how designers devise practices which apply parametric design technology to complex design situations effectively.

I have explored the problem framed by this thesis through considerations of research and practice.

1.1 THESIS STATEMENT AND AIMS

Although great advancements have enabled the use of parametric modelling technology in realising complex building designs, there remains an open question of whether these technologies can be made to cope better with the higher-levels of complexity required in the development phases of challenging design problems. Particularly in the ways these technologies can allow designers to improve upon designs which seem to challenge traditional understandings of design expertise and design practice, challenging fundamental assumptions of the
AEC (Architecture, Engineering and Construction) disciplines, of the architecture profession and of the methods by which architectural designers transform ideas of form into physically realisable constructions.

The principal objective of this research is to develop a deeper understanding of how parametric design technologies can be enhanced to facilitate the design of complex design projects from schematic design. These projects call for experimentation in the absence of a rule-book on how to make these unprecedented forms, material systems and innovative fabrication processes physically and economically realisable within the rational limits of practice.

My research targets the development of workable strategies to overcome the inflexibility inherent in off-the-shelf parametric modelling systems. To avoid the possibility of developing a skewed approach\(^1\), I also developed a strategy which does not use standard parametric design software. This strategy develops custom software from a blank-slate using a software development process. This balanced set of strategies, first the standard and second the bespoke, allows me to test comprehensively a variety of opportunities designers can take with the use of parametric design software in enhancing flexibility while undertaking complex design work.

\(^1\)By developing a strategy which is independent from the standard parametric modelling software, the research avoids the development of a one-size-fits-all solution to the problem of parametric modelling inflexibility. For example, by investigating other means of developing flexible models which do not depend on the use of a parametric model. This avoids framing the research only from the perspective of standard off-the-shelf parametric modellers and provides the basis for a wider definition of softwares which provide flexible modelling and yet are not parametric in a stricter sense. For example, softwares which do not use Acyclic Graphs as the main model persistence and interaction mechanism.
I have developed a conceptual framework from the considerations of and observations drawn from the literature, as well as the considerations and observations made from the eight case-studies selected in this study. The case study considerations respond to both my literature review and reports from my practice.

There are indications within the literature that researchers are finding the parametric modelling paradigm restrictive in enabling design flexibility. Examples of evidence supporting this statement are described in the work of Prof. Mark Burry, Dr. Jane Burry, Dr. Robert Aish, Prof. Robert Woodbury, Dr. Axel Kilian and Dr. Dennis Shelden. These researchers all promote the uptake of parametric design but also warn about its potential restrictions to design practice.

While there are hints from the literature on the limitations to flexibility within parametric design models, there seems to be a gap in the parametric design research reported, as I was unable to source any research which directly addresses how designers can overcome parametric modelling inflexibility. The authors do not address how to develop flexible design environments from the use of parametric modelling systems. The question remains unanswered: can the parametric modelling paradigm be more effectively augmented through a substantially

---

2Mark Burry is Professor of Innovation and Director of the Design Research Institute at RMIT University and executive architect and researcher at the Temple Sagrada Família in Barcelona.

3Jane Burry is Director of the Spatial Information Architecture Laboratory at RMIT University in Melbourne.

4Robert Aish is Director of Software Development, Platform Solutions at Autodesk.

5Robert Woodbury is Professor at Simon Fraser University, Canada.

6Axel Kilian is an Assistant Professor at Princeton University.

7Dennis Shelden is the Chief Technology Officer of Gehry Technologies Inc.
revised technology and strategy? To what extent will this augmentation enhance the flexibility in which designers model complex design work?

In chapter two, I discuss the limitations of parametric design from the perspective of leading researchers. Therefore, it will not be necessary to expand on the known limitations. Instead, this thesis tests if the parametric modelling environment can give designers a higher-level of control and flexibility, by using parametric design models strategically. These higher levels of control and flexibility are sought with the aim of testing whether the strategies outlined in this thesis can assist designers in overcoming the parametric modelling inflexibilities identified in the literature review and from the reports of practice.

This research takes two approaches to exploring this topic and answering the questions. The first approach reflects on scholarship about parametric modelling systems to date. The research then gathers some general considerations of practice, general observations and generalisations which can be made from employing parametric technology in both practice and academic research.

The second approach combines the case-study method with action research for the development of two major strategies and eight sub strategies which are reported on this thesis. It does this by providing two general strategies to achieving flexibility with the strategic use of parametric modelling systems.

The two central strategies are the standard approach (chapter 3) and the bespoke approach (chapter 4). The standard strategy deals with how to overcome inflexibility by using standard parametric design software. The bespoke strategy deals with how to overcome inflexibility by using bespoke software prototypes. The two general strategies used an action research method as the methodology for its development. This method positions the researcher as insider within the
development effort and can provide opportunities to gain insights that would not be possible through interviews and surveys. The action research method also shifts the emphasis of the research from hypothesis testing research towards more experiential learning research [Kolb and Kolb, 2009, Kolb, 1984]. Through action research, I am able to provide an insider’s view to a complex practice while also developing methods of learning from my practice and relating this to the wider literature on these subjects. [Raelin, 2009] presents common aspects of the various modes of action research perspectives:

- Action Research develops contextualised and useful theory rather than test decontextualized and impartial theory.
- Action Research invites learners to be active participants, leading often to change in the self and the system.
- It endorses reflection-in-action rather than reflection-on-action.
- It welcomes the contribution of tacit knowledge to learning.
- Its measured learning outcomes are more often practice-based rather than academic.
- It is more comfortable with tentativeness rather than certainty.

The case-study research is an empirical inquiry within its real-life context, particularly important when the boundaries between phenomena and context are not clearly evident [Yin, 2009]. The case-study method is used to analyse both the projects and the strategies. The projects and strategies were developed using the action research in combination with the case-study method. Combining
the case-study method and the action research method enables the researcher to develop practical research in its real-life context. This combination guarantees the developed research is usable in practice. By combining both research methods the researcher can develop observations as a detached observer, which is the default position for the researcher within the case-study method, and also to develop observations as an insider interactive with the research context, which is the default position of the researcher within the action research. This balanced method enables the researcher to observe phenomena for which he has little control of the central variables while also developing research where he can influence the variables and in this way, the researcher interacts with his research context. Through the case-study method, I have been able to document and observe the eight selected case-studies. Through the action research method, I have been able to explore opportunities for reflection-in-action within the context of the eight selected case-studies produced.

These two general strategies are themselves subdivided to respond to a continuum which graduates from the standard towards the bespoke.

This continuum puts the pure parametric design model on one end of the spectrum, and pure bespoke softwares on the other end of the spectrum. In-between the two ends of this spectrum, there are graduations which are either closer to the standard or closer to the bespoke. The graduations which are closer to the standard software range from minor customisation to the standard software towards major customisations to the standard software. The graduations closer to the bespoke software range from using standard software development practices towards customised software development practices. The standard continuum begins with parametric design models using embedded computer code and
parametric design models being controlled from an external software tool. The bespoke continuum encompasses bespoke softwares using standard software libraries, bespoke softwares with custom libraries and bespoke softwares with a custom scripting layer for programming flexible models.

Each case-study becomes a subcategory of one of the two general strategies outlined above by using the standard-bespoke continuum previously described as the categorisation mechanism to locate each case-study within the strategy framework. While I recognise that it is difficult to make generalisations from a single case study, the strategies which will follow respond to both considerations from the eight selected case studies and considerations from the literature. The literature and action research complement the limitations of the case study.

My central proposition then aims at enhancing parametric design software to facilitate designers in developing flexible design environments. This involves the development of workable strategies to overcome the inflexibility of parametric design systems and the complex design problems which challenge both designers and the parametric design environment. This in turn, will allow the parametric design environment to provide an adequate level of flexibility during the design process.

1.2 RESEARCH MOTIVATION

I commenced my doctoral research in January 2010, in response to a recurring problem I faced during my experience as a senior research and development design consultant at the design consultancy firm Gehry Technologies Inc. During this experience, I was responsible for the development of many parametric
design models, these models were built specifically for coordinating complex design work. Through the many projects I mapped using parametric design models, I realised these modelling systems had a significant limitation. As parametric models exceed levels of complexity that go from toy systems to the real world practice problems, they become increasingly inflexible. While only through its use can designers map complex design projects when the levels of complexity of a project are beyond ordinary the parametric design models seems to fail at addressing the essential needs of flexibility required to maintain the model. For example, while the parametric modelling paradigm can handle models with 200 components with relative ease, once a project reaches a size of 250,000 components, the paradigm looses its effectiveness at providing flexibility in design decision making.

The motivation for starting my doctoral research was to explore various experimental strategies. These strategies would enable me to improve the flexibility I was not able to achieve during my early experiments in practice. Can parametric design systems overcome these known limitations? Could parametric design models maintain the same levels of flexibility regardless of the model complexity? How scalable are these strategies?

These formative experiments helped me frame the research from the outset. This experience and the lack of having readily available strategies as a senior R&D design consultant motivated me to direct my research to find approaches to flexibility in parametric design modelling; particularly how challenging geometrical projects actually challenge the flexibility of the parametric design paradigm.

Because of the constraints imposed by the technology over the designer, I set out to find methods to overcome these limitations and, with this, enhance flexi-
bility in design modelling. The difficulty these limitations impose over the process of design, challenges the adequacy of the parametric modelling paradigm as an appropriate design environment for delivering rarefied design problems which resist rationalisation. This frustration with the technology led me to develop research for alternative methods to the traditional parametric modelling paradigm thus inspiring me to research bespoke strategies. For example, Dr. Axel Kilian developed spring-particle based bespoke software systems using the readily available Java based processing library to emulate catenary\(^8\) hanging models (see Figure 1.1). Kilian’s decision to make this bespoke software prototype stemmed from the shortcomings of the parametric design systems available to him at that time, in particular the shortcomings these systems had in enabling designers to represent bidirectional constraint systems adequately [Kilian, 2004]. His decision to search for bespoke software prototypes which emulate\(^9\) the behaviour of parametric design systems led me to the question of whether there were lighter alternatives to the parametric design paradigm. Lighter alternatives would be those that enable flexibility and yet do not require the user to create and edit a parametric schema as part of the design process. I then became interested in methods which went beyond the simple use of parametric design models. Meth-

\(^8\)The ideal shape of the curve produced by a chain or string-like material hanging from its own weight.

\(^9\)For example, the CADenary software tool provided the user with enough design modelling flexibility and yet was not modelled using a parametric modelling schema. Instead, the tool was developed as a custom software tool with a custom end-user interface which facilitated certain end-user interactive changes which would have been difficult to map using a parametric modeller. Thus the tool emulates a parametric model, because it enabled the production of design variants without a parametric model.
ods which enabled the development of custom tools going beyond the existing commercial parametric design tools. These strategies eventually became key aspects of the delivery of many of the projects I developed during this experience and subsequently documented here in depth within the case-study narratives.

Figure 1.1: Cadenary Tool, developed by Axel Kilian. This custom bespoke tool was developed using the Processing language for the design of funicular structures inspired by the inverted hanging models of Antoní Gaudí.

1.3 DESIGN MODELING INFLEXIBILITY

While the research community widely acknowledges the use of parametric design models in contemporary design projects; parametric design technologies also present barriers to flexibility in design modelling, these barriers restrict
designers from achieving a desirable enhanced flexible design modelling environment. However, it could be argued that most projects using parametric design technologies could not have been physically realised otherwise. A review of the literature indicates that several researchers acknowledge the advantages parametric design models bring to designers in enabling design flexibility, but equally the same researchers acknowledge that there is still significant room for improvement. Parametric design modelling alone does not guarantee a comprehensive enhanced flexible design modelling environment, as designers find it difficult to cope with ways of effectively integrating parametric models into normative practice. The literature suggests that there may be fewer examples of enhanced flexible design environments than there are examples of poorly implemented flexible design environments.

My research considers how these relatively poorly implemented flexible design models can be transformed by designers into enhanced flexible design models within their practice.

A framework for the development of strategies that induce flexibility in design modelling is built firstly through the analysis of the available literature on parametric modelling and literature about contemporary uses of parametric models in design practice. Secondly, from the use of a mixed-method research methodology involving both the use of the case-study method and action research methods to analyse the eight case-study projects presented and analysed in this thesis. The two general strategies formalise the descriptions, observations and considerations gathered from these projects. These strategies respond directly to the possibility of providing an enhanced flexible design environment through the use of parametric design technologies.
1.4 RESEARCH CONTEXT

My research explores how to develop flexible design environments, and how these flexible design environments facilitate the delivery of complex projects in contemporary design practice. The research draws from my experience as a key member of the team working towards the finished construction of two complex projects at the design consultancy firm Gehry Technologies Inc, my participation on isolated problem-solving tasks developed for the Sagrada Família Basilica in Barcelona, and my participation on four high level research workshops including SmartGeometry\textsuperscript{10} (SG) and two prototype driven workshops developed between the Spatial Information Architecture Laboratory (SIAL\textsuperscript{11}) in Melbourne, and Centre For Information Technology and Architecture (CITA\textsuperscript{12}) in Copenhagen, as well various workshops initiated at RMIT to introduce bespoke softwares written in C++ to designers. This research context is the setting for my doctoral research reported and discussed in this thesis.

The funding for much of my research has come from the Australian Research Council (ARC) research grant titled “Challenging the inflexibility of the flexible model”\textsuperscript{13} hosted at SIAL with the aims to explore improving parametric modelling to tackle rework and inefficiencies in the AEC industry and the case-study

\textsuperscript{10}SmartGeometry (SG) is a non-profit organization focusing on the use of the computer as an intelligent design aid in Architecture, Engineering and Construction (AEC). It encourages collaboration between practicing AEC professionals, academics and students using computational and parametric software tools. [Group, 2014]

\textsuperscript{11}http://www.sial.rmit.edu.au/

\textsuperscript{12}http://cita.karch.dk/

\textsuperscript{13}http://architecture.rmit.edu.au/Projects/arc_funded_projects.php
including the review of various projects selected by the chief Investigators as the initial context to this research. During the initial stages of this research, I was responsible for exploring various approaches to flexibility which address the aims of the ARC funded project, and within this context I developed a niche topic that became the core topic of my research and the central topic of this dissertation.

In summary, my research and implementation of the parametric design strategies outlined in this thesis form a principal basis of the activities in my research and these in turn provide a context to my practice.

The observations, the gap identified in the literature and opportunities for action developed during these four years of my doctorate, set the scene for my position as key researcher within this research context.

1.5 RESEARCH STRUCTURE

In order to test my central research proposition of whether the limitations of flexibility found in the parametric design paradigm could be enhanced by strategically manipulating it. I will first require the analysis of the core relevant research on the topic of parametric design and modelling, and second on the subtopic of inflexibility.

Through the various sources available for consultation, I aimed at arriving at a rigorous confirmation of the literature through triangulation of sources and the confirmation of considerations or agreements between the researchers.

Secondly I set out to develop a strategy based framework from a secondary analysis of the considerations of both research and practice. Chapter 3 and Chapter 4 outline these strategies.
Lastly, by testing these strategies with research from my practice I developed and evaluated the whole research thematic. This research thematic answers the questions outlined from the outset on the possibility of enhancing flexibility within parametric design environments from a strategic use of this technology.

In chapter two, I document the state of play by documenting the current research developed on parametric design research in the context of architecture design research and the various sources of descriptive literature on successful projects completed from the use of parametric design technology in the materialisation of these projects. This chapter introduces the background to the problem of design modelling inflexibility within parametric design.

In chapter three I document the Standard Strategy and all of its sub graduations, this chapter addresses the opportunities for designers to enhance flexibility in design modelling by using strategies which depend on the use of standard off-the-shelf software.

In chapter four I document the Bespoke Strategy and all of its sub graduations. This chapter discusses strategies which depend on the development of custom bespoke software which address problems of design practice.

In chapter five I discuss the thesis argument and support for the conclusions presented in chapter six. This discussion evaluates the findings of chapter two, three and four tying the literature, the case-studies and the strategies together and evaluating the plausibility for the approaches undertaken to enhance flexibility in design modelling.

In chapter six, I present the final points to the thesis, these conclusions will be drawn from the discussion presented in chapter five.
Chapter 2

STATE OF PLAY

2.1 BACKGROUND

2.1.1 THE INDUSTRY

Beck\(^1\) indicates how “Over the past three decades, most industries have undergone significant transformations resulting in substantial improvements in the value of their products and services”. [Beck, 2001]

On the contrary, the AEC (Architecture, Engineering and construction) industry has not been as receptive to these technological developments. The industry is famous for the reluctance of its stakeholders in adopting significant improvements in the way products are idealised and produced.

Beck also states that “Engineering News Record, the Construction Industry Institute, the Lean Construction Institute, and a variety of other industry publications and associations have documented much data substantiating the magnitude

\(^1\)Peter Beck is Chief Executive Officer and Managing director of The Beck Group.
of waste inherent in the traditional AEC delivery process”. [Beck, 2005]

Within architecture, Meredith posits: “Architectural production has been rather unsuccessful at keeping up with technological advances.” [Meredith and Sasaki, 2008, p2] He furthers the point that the current sluggishness found in the so-called “Real State Industries” when compared to the progress made in other more innovative industries, could be attributed to the AEC industry employing century-old-technologies and the risk averse culture absent in the other design-related industries such as the automobile and aeronautical industries [Meredith and Sasaki, 2008]. This lag or sluggishness as compared to other industries has also been noted by other researchers [Beck, 2001, 2005, Woudhuysen and Abley, 2004].

As indicated by [Beck, 2001] the conventional processes used in construction today prevent us from achieving significant progress. Relying solely on technology is not the solution to this problem. This technology will only enable us to enhance performance and reduce the latencies found in the way the disciplinary silos deal with the issues of coordination and integration, but the top-management of the disciplines will have to buy-in if we are to see significant progress in the industry as a whole.

Early attempts in technology transfer from the shipbuilding industries to the AEC industry were optimistic about how these technologies could radically transform the AEC industry. Ten years later many researchers have reevaluated their position in regards to the technology. Shelden indicates in [Kedan, 2010, p182–189] that the kinds of change brought from the adoption of these technologies, will be more a “revolution through evolution” [Kedan, 2010, p188] rather than the abrupt changes that were envisioned earlier by early adopters.
The reviewed literature suggest these technologies will lead to a more subtle change as the adoption of these tools transforms the industry. Shelden indicates a change in attitude as he suggests this transformation is happening at a “slower pace than some of us would like, but arguably as fast as the industry can consume it.” [Kedan, 2010, p188] Shelden’s statement refers to the AEC industry resistance to the adoption of technology and to the changes it could bring to the conventional process of design and fabrication delivery.

Many researchers have discussed the array of features in Computer Aided Design and Manufacturing (CAD/CAM) software developed from other industries from which architects would benefit the most. Chaszar [Chaszar, 2006] states the following features at the operational level:

“Computation, as the ability to perform numerical operations at high speed[…], geometric manipulation - the ability to deal with forms of great complexity[…], standardization - the ability to allow repetition [… or …] recurring design situations, rationalization - the ability to make explicit (and so editable) the decisions leading to a particular design solution[…].” [Chaszar, 2006]²

He also presents the following features at the level of practice:

"Project administration, procurement, documentation, collaborative communication, managing file sizes, detail resolution, protocols for information exchange and a host of other minutiae” [Chaszar, 2006]

These features presented by Chaszar, Suggest an increase in capability, and

²Andre Chaszar is a member of the editorial board of Architectural Design (AD) and is the editor/co-author of Blurring the Lines. He is also a consultant for Bollinger+Grohmann, a visiting professor at the Staedelschule and co-leader of the inter-university research group OSM (Open Systems & Methods for Built Environment Modeling)
the possibility of these tools to enhance the integration between designing and manufacturing.

As suggested by [Mitchell and McCullough, 1995] “Integrating computer-aided design with computer-aided fabrication and construction […] fundamentally redefines the relationship between designing and producing. […]”

The integration of designing with manufacturing eludes the need for the mediation produced by drawings. With the integration of CAD with CAM we can achieve higher levels of systematised customisation (mass customisation) “bringing the benefits of factory production to the creation of a unique component or series of similar elements differentiated through digitally controlled variation” [Kvan and Kolarevic, 2002]. In addition, this digitally controlled variation can be achieved with levels of accuracy and precision not attainable through traditional means of construction or fabrication.

There are intrinsic properties to 3D model-based methods that if correctly employed, have a greater chance of guaranteeing coherence of information when compared to traditional drawing media [Kolarevic, 2001, Maher and Burry, 2006, Shelden, 2002].

The revisiting of the design for manufacturing and assembly of products by the automotive, aerospace and shipbuilding industries during the 1980’s and 1990’s, demonstrated the use of integration strategies in combination with CAD/CAM technologies, and how their synergy enhances the effectiveness of the planning and manufacturing process [Kieran and Timberlake, 2004].

While the AEC industry has widely acknowledged the benefits of integration mediated through model-based technology, the process of adoption has been slow. In an industry built on the separation of design, construction and project
phases, many integrators have a hard time implementing collaboration and concurrence [Elvin, 2007].

As noted by [Hartmann et al., 2008] professionals find it difficult to integrate model-based strategies effectively into normative practice. This compounds with a lack of understanding of the digital modelling process and the digital fabrication process. [Chaszar and Glymph, 2003, Hartmann et al., 2008]

These researchers also dealt with issues of how the technology would support the design stages and the materialisation stages of advanced or complex geometry projects. [Burry, 2003] raises the concern that researchers who optimistically had adopted mechanical modelling tools had not yet paid close attention to the consequences of adopting these tools for designing. When looking closely at the implications from its use, Burry suggests the implied design process of these technologies "appears to be the enemy of intuition" [Burry, 2003].

Chaszar and Burry raised concerns of model ownership, digitally mediated collaboration and the limitations modelling tools bring to design flexibility within the conventional design process [Burry, 2003, Chaszar, 2006].

While Chaszar and Burry have a valid point to be concerned as to how this technology transfer will affect collaboration, ownership and design flexibility within the AEC industry, the challenges the industry faces today on “poor coordination and integration” have roots which predate the adoption of these mechanical modelling technologies. For example, researchers studying the practice of architecture in America circa 1990 [Cuff, 1992] show that these issues were prominent back then, when computer modelling technologies were not as pervasive as they are today.

While these appropriated technologies factor in the challenges we face with
“collaboration” and “integration”, the part they play in this issue is more the result of “how they work internally” and not the result of the unwanted side effects which result from their employment.

The discussion on collaboration, ownership and design flexibility has been reiterated by [Burry, 2003] and also by [Shelden, 2002] as the conclusion to his thesis. Shelden points that because of the way these technologies work internally they force the designer to make explicit their assumptions of how to make their designs constructible [Shelden, 2002]. It is this form of “explicit representation” which has been the most problematic to the industry. The “Explicit Representation” coerces the stakeholders to make explicit their tacit knowledge represented as intelligent computer models. Prior to representing tacit knowledge explicitly, project stakeholders would have to use a great deal of “interpretation” to transfer knowledge across disciplinary boundaries and make sense of it.

Making explicit tacit knowledge exposes an unprecedented level of accountability. This level of accountability and transparency makes the AEC stakeholders uneasy. For example; while using digital models to describe their intentions, some architects might not share the digital model as a contractual document and even if they do share the 3D model, this model might be used only as reference and a paper-bound document might still be used as the contractual document. Some architects such as the case of Foster+Partners countered this practice through the use of a verbatim “Geometry Statement” [De Kestelier, 2006] where they make explicit basic geometrical operations required to reproduce the design intentions of their complex geometrical configurations. The fabrication team or the engineering team working over-the-fence can read the “Geometry Statement” and arrive to the exact geometrical arrangement intended by the architect,
without compromising the legal risk compartmentalised within each disciplinary silo.

While an efficient strategy, there are limitations to the use of a “Geometry Statement” to communicate geometry-control across disciplinary boundaries. Certain kinds of complex geometries are difficult to make explicit in the form of a well-formed rational logical verbatim “Geometry Statement”, as there is the possibility of a future misunderstanding in interpreting the description of shape and form from a textual description. For example, the geometries of Frank Gehry resist these kinds of descriptions, in contrast to the geometries of Foster+Partners whose pre-rational forms enable the taming of the process of shape control. For this reason, Gehry uses physical models to persist the design intentions. The core of [Shelden, 2002] PhD thesis was in representing the material properties of paper like surfaces in an elegant and coherent computational codex. This codex helps constrain the design space yielding only feasible design solutions. Most fabrication materials used in cladding exhibit paper like material properties (almost negligible curvature, low ductility).

For the Guggenheim in Bilbao, Gehry contracted Rick Smith’s CATIA (Computer Aided Three Dimensional Interactive Application) based consulting firm, C-Cubed\(^3\) as a management strategy. This strategy allowed Rick Smith to assist the other stakeholders in the construction process without affecting their autonomy. Using a third-party modelling consultant, also provided important liability advantages to Gehry and others. The third-party consultant position provided a digital continuity from the design phases to the construction phases that would not be possible had Gehry interacted directly with the fabrication

\(^3\)C-cubed is now Virtual Build Technologies.
consultants [Lindsey and Gehry, 2001](p87).

The use of parametric design tools, particularly the ones that enable collaboration such as CATIA™, reconstitute the role of the architect into “a central role in the process of construction” [Lindsey and Gehry, 2001, p80] by directing the actual construction of the building components. Parametric design tools enable continuity from the design process to the manufacturing process crossing disciplinary boundaries and practices. While some architectural firms have struggled more than others in bridging this gap without discontinuities, the parametric design process has become invaluable in the materialisation of hard to conceive and difficult to fabricate architectural projects. However negative issues arising from the use of parametric design software also deserve attention. Acknowledging these issues does not undermine the relevance of the other challenges we are facing with “poor coordination/integration” and with providing a digital continuum among a fragmented process and discontinuous industry.

I am aware of the possible problems which could arise from computer modelling use, for example, issues with the legality of sharing information models, information model ownership, the poorly integrated disciplinary silos of the AEC industry and the cannibalising\(^4\) practices among these competing disciplines. These issues will not be provided coverage in this thesis. The issues described above, have been narrowly discussed only to provide an external context to the main topic of parametric design use and the core subject of this thesis—strategies to overcome the inflexibility of parametric design software— which

\(^4\)A form of self infringement between the stakeholders of an industry. For example, when Engineers provide services which compete with the services of architects, or vice-versa when architects offer services which compete with the services of engineers.
will be discussed further in the “phenomenon” section of this chapter as well as Chapter 3 and Chapter 4.

For example, during the delivery of the Yas Island Marina Race Track Hotel (YAS) project, Gehry Technologies Inc used an Integrated project Delivery (IPD) process. The issues exposed in the scholarship on poor “coordination”, poor “integration” and problems of “model ownership” were nearly absent in this project. In the Museo Soumaya (MUS), Gehry Technologies implemented a similar project-wide team structure and delivery process. The architect Fernando Romero and the fabrication team were tightly integrated early in the project. With the exception of Geometrica— the triodesic structure consultant— the client owned all the companies involved in the realisation of the project, enabling a vertical integration of the fabrication team. An early close collaboration helped transfer specialised fabrication knowledge from the fabrication consultants to the early stages of the design process. This rather unique project situation reduced the typical latencies and provided the framework for a near real-time feedback between project collaborators. This near real time feedback is absent in most conventional delivery processes. In typical delivery methods, each discipline

5“Integrated Project Delivery leverages early contributions of knowledge and expertise through the utilization of new technologies, allowing all team members to better realize their highest potentials while expanding the value they provide throughout the project lifecycle” [Eckblad et al., 2007]

6The Museo Soumaya client was Grupo Carso. The name Carso stands for Carlos Slim and Soumaya Domit de Slim, Carlos Slim’s late wife, after whom the museum was named.

7Design-Bid-Build or Design-Build projects are linear processes [Demkin, 2002], where by information is passed over-the-fence in a serial manner. With an integrated approach such as IPD, information is integrated using technology. This integration enables the design and fabrication phases to overlap, enabling concurrent engineering practices [Elvin, 2007]
works selfishly on its own bottom-line as there are no economical incentives for the stakeholders to minimise project wide costs and project-wide rework across disciplinary boundaries [Beck, 2001]. Therefore, in typical construction delivery methods the client pays for the consequences of inefficiencies. When inefficiencies are avoided through a more effective use of technology and better integration practices, clients avoid paying high premiums, which result from rework. Inadequate integration also affects the quality and time-to market of end products, as poor integration results in rework, resulting in unplanned work and construction delays.

Lastly, The FABPOD project, a prototype project developed during my research at SIAL, used an “internal” coordination strategy [Ku et al., 2008]. The design team and the fabrication team belonged within the same institutional boundary, sharing risk and rewards.

While acknowledging the importance of these issues crippling the industry, for reasons of scope, I will not be directly addressing the issues of “coordination” and “integration” or any of its subtopics such as interoperability and contractual arrangements that minimise these “problematic” teamwork situations which arise from the inefficient use of technology and from organisational science challenges.

Paul Nicholas PhD thesis “Approaches to Interdependency: early design exploration across architectural and engineering domains” [Nicholas, 2008] and Dominik Holzer “Sense-making across collaborating disciplines in the early stages of architectural design” [Holzer, 2009] discuss poor “coordination” and “integration” thoroughly and how information models can provide a solution to these issues.
This thesis will deal with the operational limits and opportunities of enhancing flexibility within Parametric Design technologies. It will also address the representations designers use to translate their design intentions into accurate fabrication intentions. This translation process enables designers to satisfice advanced constructibility constraints [Shelden, 2002]. The parametric design technology supports this process of translation. This thesis will present flexible approaches to translate design intentions using parametric design models.

There is a continuum of research opportunities which range from the scale of the industry, to the scale of the discipline, and finally to the scale of design operations.

While it is tempting to tackle all the research scales in a single research project, the endeavour is beyond the scope of a single researcher. There are limited pragmatic opportunities to be harnessed along all scales of this spectrum. This thesis will respond to the operational limits of parametric practice. The thesis will expose opportunities and limits found at the scale of tasks. These opportunities enhance flexibility within design environments.

The main objective of this thesis addresses issues designers face when they apply the parametric technology to enhance the quality of design outcomes. From the application of parametric technology to design problems, designers increase the opportunities for throughput in iteration and continuous refinement of design solutions. The secondary objective of the thesis will also address the model complexity issues that emerge as a result of applying parametric modeling technologies to complex design problems. This thesis will address the problem which emerges when these tools are used to solve the design development of challenging design problems. While most complex geometrical design solutions
could not be delivered without these tools. Some rarefied design problems and
design solutions seem to resist the rationalisation process supported by paramet-
ric technology. These rarefied design problems expose limits in the usability of
the tools.

The thesis will address a new paradigm for enhancing flexibility within para-
metric design systems. This new paradigm enables the development of a new
role. This role provides knowledge and expertise in the development of spe-
cialised design tools. For example, Foster + Partners has the specialist mod-
elling group [De Kestelier, 2006] and Arup has the Advanced Geometry Unit
AGU [Meredith and Sasaki, 2008, p34–60]. These in-house groups of design
specialists provide design generalists with a diverse set of skills ranging from
the expertise in complex geometry, environmental simulation, parametric design,
computer programming, and rapid prototyping. Within these groups, the design
specialist develops targeted tools to solve complex design problems not possible
through the traditional interfaces in existing off-the-shelf software. For design
practices without the in-house knowledge and expertise, a network of highly
skilled consultants provide specialised design services such as Gehry Technolo-
gies Inc, Evolute, Design to Production, FRONT Inc. For example, Gehry Tech-
nologies Inc develops specialised design services which encapsulate knowledge
and expertise in solving complex design problems [Technologies, 2014].

Lastly this thesis will expose specialised design software for complex design
problems extracted from the Sagrada Família Basilica in Barcelona. These spe-
cialised design tools to solve complex design problems, present a new form of
knowledge and expertise in the resolution of rare design problems, addressable
only through an algorithmic intervention to parametric design software.
The reasons to extend the capabilities of parametric design systems provide the basis for the thesis. As parametric design systems currently stand in the years 2010–2013, they do not present a comprehensive solution to geometrically complex design problems. Because of the shortcomings of the software style in its off-the-shelf state, various design practices worldwide have had to develop the role of the design modelling specialist. This role extends the capabilities of parametric design software in solving rare design problems. These design specialists, are also facing challenges when dealing with the limitations of the software.

Researchers have documented “scripting” as an emerging capability of design culture [Aranda and Lasch, 2005, Burry, 2013]. The specialisation of scripting referenced in this thesis, is a form of scripting for controlling parametric design systems. These scripts, developed on top of parametric design systems, support design specialists in rationalising emerging forms and emerging construction systems.

In this new paradigm, design specialists build bespoke tools on top of existing parametric software. These bespoke overlays enable design specialists to “script”— and increasingly to “program”— the creation and modification of complex associative models. These overlays require “interpretation” of designer input. The algorithmic control\textsuperscript{8} enables the designer to produce parametric design models with associative networks interconnected in ways not possible when

\textsuperscript{8}These are not traditional algorithms. They are algorithms which operate above the editing features of a parametric modelling application.
modelling the associations using the keyboard/mouse\textsuperscript{9} interface.

Designers can opt-out in having to define the formal properties of their design intentions by using algorithms in and out of parametric design models. The task of formalising design intentions has made parametric systems seem germane to conventional ways of thinking within design practice [Lawson, 2006]. Ironically, by deferring the formal definition of parametric design models to an algorithmic layer—a less implicit formal representation system—the designer is free to develop parametric variants and parametric combinations in ways not possible through the traditional interface. This way of working responds to Burry and Burry’s “Parametric Design Inflexibility” [Burry and Burry, 2008]. The deferring of the explicit definition of formal intentions to an algorithm does not preclude having to define an explicit formal definition, instead it enables the designer to detach formal definitions from design variants and embed them in computer code supported by the role of the design specialist. The design specialist must interpret the design intentions from designers and make them explicit through reusable computer code.

[Hudson, 2008, Shelden, 2002] indicate how the use of parametric design software makes explicit the constructibility of design intentions. The design modelling specialist makes explicit the assumptions of design constraints and

\textsuperscript{9}This interface is more technically referred as the windows, icons, menus, pointer interface (WIMP). The WIMP interaction paradigm is the dominant human-computer interaction used on most modern computing systems today [Hinckley, 1996]. WIMP interfaces were first developed by Xerox PARC for the Xerox Alto in 1973. The inspiration for the Alto’s graphical user interface paradigm came from the On-Line System (NLS) developed by Douglas Engelbart in 1968 at the Stanford Research Institute, the precursor to the graphical user interface as we know it today. For a demonstration of this system in action, see “The Mother of All Demos”.

30
fabrication rules assumed by both the design and fabrication team respectively.

Parametric design systems do not have the capacity to emulate design synthesis. Parametric design systems rely on the design team’s design synthesis capability. This knowledge is dependent upon the design team’s experience in fabrication and previous design experience of similar complex design solutions. Donald Schon refers to the designers dependence on experience, as “the designers repertoire of designs” [Schön and DeSanctis, 1986].

Parametric design systems only augment existing capabilities. Modelling design problems through parameters and relations, require extensive use of interpretation and experience. This interpretive knowledge and experience, enables the designer to understand how to make appropriate trade-offs, due to the representational limitations of the software, in making explicit physical phenomena accurately. For this reason, design teams delivering complex geometrical design solutions require knowledge in advanced mathematics, computer science and complex architectural detailing. The design modelling specialist encapsulates this knowledge and expertise. Although the majority of design modelling specialists are architects, these specialists do not acquire their knowledge from the body of knowledge of degree granting architectural programs. Instead, design modelling specialists use knowledge from various technical domains and are mostly autodidacts [Burry, 2013].

This new paradigm extends the parametric design software style and with it, promises to address many of the limitations found in off-the-shelf parametric software. Designers should be aware of the limitations of this paradigm before its uptake. This thesis will present both a solution to flexibility and will also discuss its limitations.
This thesis considers approaches to flexibility within the confines of existing parametric technology and design practice.

2.1.2 RATIONALISATION

As indicated by [Kaijima and Michalatos, 2007, 901] “Digital design tools have increased the architects’ capabilities to create complex forms. When attempting to realize such designs one is challenged to rethink the processes that generate such forms as well as the established ways of materializing them” [Kaijima and Michalatos, 2007, 901].

During the last decade we have seen an expanded use of new tools in proposing architectural outcomes, this plurality of new forms and methods of shaping comes as a result of progressive integration of computation into the architectural practice. The adoption of complex geometries has broadened through computation the domain of design expression [Dritsas, 2005, Kaijima and Michalatos, 2007]

The rationalisation process is an attempt to legitimise new forms emerging from this new way of practicing [Glyph et al., 2002, Shelden, 2002]. The rationalisation process enables to materialise the complex forms which have emerged from this increase in the capabilities of the architect. To materialise these forms, architects need to discretise the forms into constructible components. As [Kaijima and Michalatos, 2007, p901] point out “A problem that persists in the interface or gap between architecture and engineering practice is discretization. By this we mean the necessity to decompose continuous geometric objects into discrete elements”.

A rationalisation strategy is a heuristic device which enables a designer to de-
velop a trade-off between constructibility and design expression. Many factors can determine how this trade-off is weighted. Issues such as assembly scheduling, manufacturing cost, design cost or the availability of resources and labour affect the decision over how to privilege either constructibility or design expression during the rationalisation process. The rationalisation process is an experimental process, not in the sense of a controlled experiment but rather as a what-if experiment. An example of the kind of experiment which is conducted during a rationalisation process is the panelling of free-form surfaces [Eigensatz et al., 2010, Schiftner et al., 2009]. In this popular method, the designer tries to manage the trade-off of expressing the continuity of the surface and the formal qualities of the surface, with constructibility concerns, reducing either the design complexity or the way in which the design is manufactured and assembled. In this method, the parametric design software is used to host a panelling algorithm and the data it produces. The panelling algorithm is custom tailored to the panelling problem. No two panelling algorithms are alike. The algorithm extracts the panel construction data from a single large free-form surface. This method takes a single large free-form surface as input, resulting in multiple smaller constructible components as output. The rationalisation strategy enables the satisficing of desirable fabrication constraints in the extraction of this data. For example, by constraining the data extraction algorithms to a restricted solution space, “filtering” only desirable fabrication properties, such as for example, “planarity” in a panel, would enable the fabrication of the panel through a 2D CNC (Computer Numerical Control) process— a low cost solution per panel— rather than requiring a mold— an expensive solution per panel. Where a mold is inevitable, the designer can use a statistical process such as cluster analysis.
to determine the trade-off between unique molds and the frequency of use of these molds on a large number of panels [Peña de Leon, 2012]. A large number of unique panels can be fabricated using a low number of molds by using this technique [Peña de Leon, 2012].

The introduction of a rationalisation strategy early in the planning for fabrication and assembly phases has a dramatic impact in the reduction of the increased cost and risks of materialising unconventional geometry [Shelden, 2002]. Determining where to introduce the rationalisation strategy within the design process is crucial in exploiting the use of this heuristic device. In the best-case scenario, the design logic uses a pre-rational strategy from day one. A rationalisation strategy can also be introduced late in the design process (Post-Rationalisation) [Maher and Burry, 2003]. This allows for an appropriate handling or management of the design consequences of an earlier decision making. Lastly parallel decisions affecting the rationalisation of form can be made alongside of the design process (Co-Rationalisation coined by Hugh Whitehead former Director of the Specialist Modelling Group at Foster + Partners) [Fischer, 2007].

As exemplified by the Miran Galerie of dECOi Architects [Dritsas, 2005], in the post-rational method, geometry must be reconstructed through the use of a degenerative geometric transformation process. In this process, the complexities of the design become abstracted by simpler fabrication constructions. For example, in the Miran Galerie, a process of contouring reconstructs the geometry [Dritsas, 2005]. This process reduces the original geometry into simpler fabrication constructions.

This abstraction process is convenient for decoupling design intent from the final methods of construction. [Dritsas, 2005].
In the Bishops Gate Tower, [Hesselgren et al., 2007] suggest a required shift in thinking from designing objects to the design of the systems that generate designed objects.

The built-in guarantee of constructibility in a model is dependent on the coherence of the geometric schema. This simple principle, will be a running theme of the strategies presented in this thesis. It follows, that simple geometry should provide a simple construction method.

“Parametric technologies allow detailed logic of system component organisations to be encoded into generative approaches, so that this level of project understanding can be applied as a part of formal generative techniques.” [Shelden, 2009]. Parametric design models enable the representation of these broader definitions of form, and the devising of strategies to rationalise them generatively into constructible forms. A concept [Glymph et al., 2002] refers to as a “Geometric Strategy” defining a design constraint, enabling the embedding of fabrication rules through the use of a restricted geometric solution space.

Parametric modelling approaches which require the development of scripts to execute complex work-flows, can be developed incrementally in such a way that the same scripts used during the earlier stages of design can be used during the later stages of design [Shelden, 2009]. This implies the scripts are developed and used when design models are general and underdeveloped. These models convey general formal properties. The scripts are then re-used once the design has sufficiently been developed, where models have an increased level of reso-

---

10Generative rationalisation refers to a rationalisation process that is executed within the parametric model as an algorithmic process, as opposed to a more traditional rationalisation process which occurs outside the computer during a value engineering exercise.
olution and address fabrication conformance [Shelden, 2009]. In the majority of cases the increase in resolution of the model does not require a major rewrite of the original script, if the script warrants a rewrite, the change is minor.

The use of generic design software presents difficulties to the designer in re-solving awkward design problems. Shelden suggests that our demands for software to become operative across different disciplinary domains and platforms, has produced a trade-off between supporting generic design intentions or unique design intentions [Shelden, 2009]. Shelden indicates “in some ways this aspiration inherently requires us to restrict the set of potential intentions to the greatest common denominator” [Shelden, 2009] as this aspiration comes with an “implicit trade-off between the support of the general and the unique” [Shelden, 2009]

This implicit trade-off between the general and the unique, restricts the ways in which designers model specific design problems. The more generic the technology, the more distant it is in representing the details of the design problem. This trade-off between supporting generic or specific domains of practice with parametric design software will be addressed throughout the projects which will be documented in Chapter 3 and Chapter 4.

2.1.3 SUMMARY OF BACKGROUND

Both the industry and the rationalisation sections above introduce the background for the central topic of this thesis, “The Inflexibility of Parametric Design Systems”. The industry section exposes the current issues of the AEC industry. The industry literature was presented to provide a background to the externals of the core research. This external world is traditionally placed at the fore of
most research on parametric design and information modelling. In this thesis I foreground the internal aspects of parametric design instead.

The use of this technology suggests an evolved and improved way of working, for the designers of today and the future. The ability for these technologies to legitimise new design geometries, new materials and emerging construction methods have proved to be one of the most significant features of the technology in the AEC industry.

During the process of value engineering complex designs, the rationalisation process allows the architect to legitimise new forms. The rationalisation process allows designers to take advantage of new materials and methods of construction. This process allows the architect to experiment with shapes, forms, methods and material systems, which have not been previously tested. This process therefore pushes the boundaries of the profession.

The central environment where this process of legitimisation occurs is in the parametric design software [Lindsey and Gehry, 2001, p69–75]. The parametric design software allows the designer to experiment with new forms virtually. While the parametric design model has enabled a greater opportunity for experimentation and hence discovery, one notable area handicapped the model: the limited interaction between the designer and the parametric design model with which they are working. The process by which the designer interacts with the parametric design is limited.

In the section that follows, I will discuss in depth the literature addressing the limitation of the parametric design environment. The literature review which follows suggests designers are finding it difficult to use the parametric design environment to represent their design intentions.
2.2 THE PHENOMENON

This chapter documents the literature addressing the topic of inflexibility within parametric design. The strategies presented in chapters three and four have been constructed responding in part to the observations made in this literature review.

As indicated by Burry and Burry, “there is value in focusing on the ways in which the flexibility of our more sophisticated modelling tools can also be paradoxically constraining” [Burry and Burry, 2008]

While the inflexibility of flexible models may sound at first a logical contradiction, today designers are struggling with finding effective ways to produce flexible models despite having adopted the parametric design paradigm. This modelling paradigm had promised from the outset a greater flexibility in design modelling than its precursor— the explicit modeller. Paradoxically the same tool which promises to increase flexibility in design, has now become an obstacle to the very flexibility designers sought from its use.

Parametric models, associative geometries or relational digital models are used interchangeably to refer to the same design technology and its inherent design paradigm. In “relational digital models all the geometrical elements are associated to one another or a higher order schematic geometry linking the parts” [Burry and Burry, 2008] these higher order networks of links or associations form what is by now termed a relational ‘schema’.

As indicated by Burry “parametric modelling software is invaluable for both preliminary and developed design where there is a need for the definition, manipulation and visualisation of complex geometry” [Burry and Murray, 1997]. The parametric modelling software has clearly demonstrated in the last 13 years
its ability to enable design practitioners to conceptualise, define, manipulate and visualise complex design problems at the highest level [Kolarevic, 2001].

Having improved on its predecessor (the explicit modeller\textsuperscript{11}), the parametric modeller has enabled a completely different and improved philosophy of design compared to the explicit modeller. In explicit design “any dimensioned parameter can be changed, only through a sequence of erasure and redrawing” [Burry and Murray, 1997], in contrast to a parametric model whereby “the generated model can be ‘updated’ by identifying one or more of the parameters and changing their values” [Burry and Murray, 1997]. The contrast between these two different philosophies of working demonstrates “the difference in philosophy between explicit and associative geometry” [Burry and Murray, 1997].

As Burry suggests, parametric design with its attendant model development

\textsuperscript{11}Explicit modelling refers to software where the designer cannot associate elements of the design. They are also termed object based CAD technologies as these CAD softwares where based on “object-orientation” a computer programming paradigm. Not all explicit modellers are based solely on objects, some explicit modellers are also based on commands which modify objects. Examples of explicit modellers are Autocad without DesignScript, Rhino without Grasshopper, Microstation without Generative Components. Other examples are animation software packages such as Maya and 3ds Max. While the animation software packages provide a limited form of associativity through a history tree, parameter wiring (a process termed rigging within the computer animation industry), node graph editing, and scripting; associative design is not a central activity to these animation software packages. For example, softwares which provide associative design off-the-shelf and where associativity is central to the purpose of the software, are CATIA, Solidworks, Pro/engineer, NX. Other softwares such as Microstation and Rhino do not provide associative design off-the-shelf, however, they have developed in-house plugin systems which extend the base software providing associativity. For example Microstation provides GenerativeComponents and Rhino provides Grasshopper.
process is “[...] becoming a slave to its own process” [Burry, 2003]. The process of developing a parametric model is hardly intuitive, and its reliance on the declaration of a schema ahead of the modelling presents major obstacles to designers who have not yet conformed to this highly structural and restrictive way of inscribing their thoughts. Moreover, as suggested by Burry, this design process carries implications to the designer which can prove restrictive if they use a loose planning process. The parametric model requires a structured planning process, which is for the most part absent in traditional media [Burry and Murray, 1997].

Similar to Burry, Kilian also points to the limitations in having to “declare a schema ahead of the modelling” as well as the “structural and restrictive” ways of representing the design intent. Kilian suggest that our current parametric design tools are limited by the way they define design, as well as by the restrictions of its hierarchical dependency chain structure over the designer and the design process [Kilian, 2006, p54]. Moreover, Kilian points to how the hierarchical chain of dependency may force designers to shift structuring activities to the early stages of design, and that once these structures are in place, the designer gains little flexibility from their use [Kilian, 2006, p54].

Equally, the parametric model lacks the necessary artificial intelligence to predict future design moves. It is easy for a designer to break a model which had been previously working with a great degree of variability. The development of flexible parametric models which anticipate possible areas of conflict, require a great degree of forward planning. Burry suggest “The computer does not have the necessary artificial intelligence with which it can predict what effects the change will have on later decisions [Burry and Murray, 1997].”
With this in mind, Burry indicates "as the architectural design community becomes more familiar with parametric design software, there needs to be an awareness of the implications in its use". [Burry and Burry, 2008].

Surely, there are aspects of the design process which can benefit from having a highly structural and imperative process. For example, once past the conceptual stages of design and during the later stages of “production description”.

As Burry indicates:

"It is a common catch cry of those working in architectural practice who have experience of constructing relational digital models that this is a good workflow for modelling a project once past the very volatile conceptual stage but where the design is still undergoing significant refinement and iteration. It is not only too slow and cumbersome for the very early stages of design, especially at the stages of ideation or conceptualization, but the level of change in conceptual design is so fundamental that it can never conform to a relational schema". [Burry, 2003, p303]

While narrow, the use for the parametric modelling paradigm in design can be most effective when the paradigm is applied past the conceptual stages and for supporting designers during the detailed iteration and refinement aspects of design.

Shelden presents a similar observation in response to the question of whether the computer might be given a broader use during the design process. For example, by using generative processes as generators of the design intent, Shelden indicates how “The complexity that can be independently generated in terms
of design intent is[...] still relatively limited.” [Kedan, 2010, p184] Shelden supports this argument with the premise that “architectural intent is generally beyond [...]” a scale appropriate for the use of computers in design. Shelden suggests that rather than focus on the use of computers as the generators of design intent, architects can obtain a far more effective use of the computer in design, if we restrict their use to the stages which succeed the conceptualisation of design and preferably to “[...] localized, well-defined problems [...]” as suggested by Shelden [Kedan, 2010]. However, while the use of a generative process for the synthesis of design intent is not yet viable with our current parametric design technology, this does not necessarily mean it is not a desirable feature or use of the parametric design technology.

The observations made by Shelden indicate that the parametric modelling paradigm is not alone in falling short of addressing the wider process of design. It appears that in general, all use of computers in design is still limited, and that as yet, designers are better-off applying the computer to localised well-formed design problems instead of applying computers holistically to the larger and messy design process. This position is shared by Prof. Burry as he limits the scope of his research on parametric design in the Sagrada Família Basilica to “the more localised issue of design development” rather “[...]than focus on design as a holistic process of formal synthesis” [Burry, 1996].

From the observations made by Burry & Burry, Kilian and Shelden, we can derive the following considerations. As the technology currently stands, designers will gain more from the deliberate application of parametric modelling software towards the resolution of complex geometries during the later stages of the design process and to a restricted and localised well-formed design problem.
Equally, parametric design is more effective when used to address the analytical issues of design development rather than addressing the issues of formal synthesis.

As the design protocol of parametric modelling currently stands, it requires premeditation supported by a deliberate instruction of the parametric modelling software towards a desired design goal. There is no room for ambiguity within the parametric modelling paradigm. For this reason, the parametric design modelling protocol is best applied towards restricted and localised well-formed design problems. These types of problems are more likely to emerge during the later stages of design as more information is at hand. Because designers work with incomplete information during the earlier stages of design, the later stages of design are prone to be incomplete representations of what is going to be built. As more information becomes available to the design team, the designers need to respond aptly to amend their design “production descriptions” without disrupting the earlier design decisions. While the resolution of complex geometries might seem to be a well-formed problem aptly addressable during the later stages of design, apparently architects are struggling with legitimising the kinds of forms which have emerged with the uptake of the computer as a design tool. Parametric Design enables architects to legitimise challenging geometries under the current rules of constructibility within a reasonable threshold [R Shelden, 2006]. This process of legitimisation is anything but easy or seamless; it requires grit and a creative manipulation of the technology aimed at the goals of achieving the constructibility of the desired forms and properties.

The design process remains largely unchanged with the use of parametric design software. However, as Shelden indicates “[…] all media carry certain
unique affordances, and computation has of course unique characteristics that are different from traditional physical or worldly media.” [Shelden, 2009]. The most apparent characteristic of digital models, is that in order for the models to be computationally operative, “they must be built on constructs that are explicit, specific and consistent” [Shelden, 2009]. However, these limitations are not required of physical design media. Shelden also suggests that while geometry and information models enable a high-order of operability, these models restrict the flexibility of what can be represented with them, as there seems to be a trade-off between the effectiveness of computational approaches and their flexibility. As Shelden indicates “There is something of an inverse relationship between the efficacy or power of a computational approach and its flexibility.” [Shelden, 2009]

In response to this trade-off, Shelden indicates, one must ask “given the computational approach, what is required for a computer or human to translate the model to other sets of intentions?” [Shelden, 2009]. This question, requires us to devise representations which are both legible by the computer and by humans, but somewhere along those lines a compromise will emerge, between developing representations which favour legibility by computers or legibility by humans. The parametric modeller, is one such representation which meets both computers and humans in the middle. It enables the designer to translate design intentions in ways which both the computer and the human operator can understand; for the most part this representation has enabled us to manage ever more complex design problems. As design problems become more complex, this design representation needs to be extended.

Similar to Burry’s and Shelden’s observations on the limits of scope in the
applicability of the parametric modelling paradigm within the larger process of
design, other researchers have noted that when designers expand the scope of
these technologies towards the greater process of design with an uncritical dis-
position, the technology affects the process of design negatively as the buildings
which result from this uncritical process may simplify what is otherwise a crit-
ical and complex process. For example, [Gengnagel et al., 2011] indicate “In
many cases the relationship between design idea and computational tool seems
reversed. The resulting buildings appear as reductionist materialization of the
possibilities of the software that shaped them”. Admittedly, what constitutes a
“reductionist materialization” is subjective, but nonetheless [Gengnagel et al.,
2011] brings to fore an issue of the flexibility of our design tools in accommodat-
ing a greater variety in the range of expressions permitted by our design intent,
as opposed to restricting the range of expressions by their use.

Burry and Burry provide a more critical stance towards the parametric de-
design software, by questioning the flexibility afforded its design paradigm. They
suggest that the parametric design paradigm may “delude us into painting our-
selves into a corner” by providing “infinite variety within a much reduced palette
of opportunities” [Burry and Burry, 2008]. As these technologies in their off-
the-shelf modalities provide little more than the opportunity to vary numerical
parameter values, add, edit or remove relationships, the software does little to
support the process of building the associative network of relationships, a pro-
cess Burry and Maher term “Designing the Design” or “meta-design” [Maher
and Burry, 2003].

After examining the corresponding literature, it appears there is a gap in this
literature addressing solutions to inflexibility. While the problem of parametric
modelling inflexibility has been highlighted throughout the literature whether explicitly or implicitly; there remains a gap in the literature indicating how to reduce or improve this inflexibility. Hints have been provided by many researchers, but context-based practical solutions to this problem remain to this day an open question. Specifically, how can we overcome the limitations found with the default behaviour of off-the-shelf parametric design software. As the process currently stands, the designer has little support in the development process of building the schema. There are no technological features within the available software packages which assist designers in the development process of creating the declarative schema.

Burry and Burry suggest:

“The schema as a design construct paradoxically needs to loosen-up. Some of the packages available offer such opportunities through having a scripting interface. If not actually shifting the goalposts at least we can widen them; we can mitigate the domination of the highly structured schema that orders the design to a more benign and flexible scripted narrative”. [Burry and Burry, 2008, p305]

The scripting interfaces available within some parametric packages might be able to provide the necessary means by which to relax the strictures of the schema. while there are many examples of practices extending the parametric modelling paradigm through scripting in the service of delivering challenging geometries. For example, the work of Dritsas on the Pinnacle with KPF (Kohn Pedersen Fox) [Hesselgren et al., 2007], the work of Shelden on the Music Experience Project with Gehry Partners [Shelden, 2002], the work of Cecatto
on the Beijing National Stadium with Herzog and Demeuron [Fischer, 2007], the work of Pisca from Gehry Technologies for the Beekman Tower by Gehry Partners [Burry, 2013] to name a few. There are as of yet limited examples of literature on strategies which support designers in the process of implementing “scripting overlays” over parametric modelling software or alternative strategies for dealing with the limits designers experience with the use of the highly restrictive relational schemas.

Scripting overlays over the relational modelling paradigm may provide an alternative method of interaction with the parametric model and with this enabling us to escape the restrictions imposed by the highly structural “schemas” of parametric modelling software. Yet, there are limited examples of literature addressing how to implement scripting overlays over relational schemas and how these are used to resolve complex project work with challenging geometries in design practice.

As pointed out by Burry, the “schema” of a parametric model enables designers to defer design decision-making to the later stages of the design process since with the use of a parametric model the decisions which have a direct impact over the design “can be revisited and reworked accordingly” [Burry, 2003]. The ability to defer decision-making is based on the premise that when both minor and major changes occur during the evolution of the design intent because of the use of the parametric model designers do not have to resort to “techniques of erasure and remodelling” [Burry, 2003].

While the “schema” frees the designer from having to resort to erasure-redraw techniques, the highly structured nature of a “schema” forces designers to have a great sense of premeditation over how they translate their design intentions in
a parametric model.

While arguably, the structural demands the software places on design thinking is one of the reasons why the software has not been widely accepted. The intractability of the design process might also be responsible for the difficulty designers face with representing design problems with parametric software [Lawson, 2006], and this intractability henceforth might explain its limited acceptance by the wider community.

[Aish, 2005] highlights the dichotomy between intuition and deliberate action, which seems to be compulsory in the use of parametric models. [Aish, 2005] suggests “design necessarily has to be predictive in order to anticipate what the consequence of the ‘making’ or ‘doing’ will be. Therefore we inevitably have to counter balance our intuition with a well developed sense of premeditation” [Aish, 2005]. This also proposes that intuition and deliberate action need not be placed in rivalry, but rather as complementary to one another. Aish does this to illustrate the similarities between the traditional process of design (independent of which tools are used) and the process inherent in parametric design (a process specific and unique to this tool). The analogy implies that while the “schemas” found in parametric design softwares may seem restrictive to designers, design must intrinsically also be predictive and therefore designers must balance both intuition and a well developed sense of premeditation when they design.

This well developed sense of premeditation, may not always be common to all designers. Having to build a “schema” ahead of time, assumes all designers have a well developed sense of premeditation, it is suggested by Burry and Burry that this may not always be the case [Burry and Burry, 2008].
Assuming all designers have a well developed sense of premeditation, the restrictions of having to develop a “schema” ahead of the conceptualisation is a limitation and restriction imposed on the design process, which designers may not be aware of when adopting the parametric modelling paradigm.

When designers have to anticipate the consequences of the ‘making’ in design, “schemas” become useful constructs to formalise both the “pre-geometric” aspects of design, and the “post-geometric” aspects of design. “Pre-geometric” aspects are those issues “independent of any specific configuration” while “post-geometric” aspects are aspects of the design where “once a particular configuration has been selected[...] there may be many material interpretations of the same geometry” [Aish, 2005].

Parametric models serve a twofold purpose by enabling designers to defer decision-making. Firstly they enable them to store early “pre-geometric” decisions (Design Parameters or Skeletal Geometry) when the designer has the least amount of information at hand and second to make late “post-geometric” design decisions (Fabrication Methods and Material Properties) when the designer has the most amount of information at hand, as new design criteria becomes available [Aish, 2005].

As indicated by Burry and Burry “the whole model has the potential to respond by updating in response to the new criteria or values while maintaining the relationships, as long as this is geometrically possible” [Burry and Burry, 2008].

The ability of parametric models to maintain relationships unless otherwise instructed by the designer makes the parametric model topologically stable. The
topological\textsuperscript{12} stability of the parametric model is key in enabling designers to persist early design decisions in a “schema” while also enabling late design decisions both in values and in changes to the structure of the relationships between objects.

By enabling designers to coexist early and late design decisions, parametric models enable designers to make decisions when they are most informed and when the design problem has been transformed into a localised and well-formed design problem. These features of parametric models enable designers to maintain design models whereby early design decisions and late design decisions can coexist in a single model without having to resort to blank-slate models at major design revisions. Parametric models enable designers to accommodate both incremental and major design changes in a single design model. Exactly how the parametric model does this, is the issue. Major changes require extensive manual editing. How manual the process actually is, is a part of the story of parametric modelling which is for the most part untold.

These features of parametric modelling make them better candidates for the modelling of design intent than their explicit modelling counterparts. In explicit modelling, designers cannot make associations and every change requires them either to resort to erasure-redraw techniques or to the production of a blank-slate model for both minor and major changes.

In response to the limitations the parametric modelling paradigm imposes over its users and the difficulties these users have with coping with its highly struc-

\textsuperscript{12}Topology in this sense is used here, to indicate the structure of the model, a parametric schema. This is not the same as the topology of the 3D model, represented in BREP (Boundary Representation) data, the underlying representation of 3D information in a parametric modeller.
tural schematic practices, [Woodbury et al., 2007] suggest the adoption of a conceptual tool: the “design pattern”, as an explicit element of learning.

“Patterns appear to have utility in learning. We have taught parametric modeling to several hundred professionals and graduate students. Over time we noticed that our instruction has increasingly focused on this tactical level”. [Woodbury et al., 2007]

The software engineering community has by now widely adopted the conceptual tool of the “design pattern”, Christopher Alexander [Alexander et al., 1977] coined this term in reference to well-formed design templates which can be readily deployed given the same context and problem requirements within the field of design. As indicated by Woodbury et al. “Patterns express design work at a tactical level, above simple editing and below overall conception. A pattern typically comprises a name, a problem description, an abstract solution, and a discussion of consequences.” [Woodbury et al., 2007]

Patterns enable the designer to catalogue well-formed or tamed-problems into a stock of ready-made deployable generalisable solutions for tackling same kind problems in the most general and universal way possible. Patterns can be applied to as many similar problems as possible, albeit only for tame-problems. Design patterns are not as useful in the resolution of ill-formed problems. Ill-formed problems are most likely to emerge within the context of real world problems in design practice.

The usefulness of design patterns in the education of the uninitiated is self-evident. However, the benefits of using this conceptual tool for assisting design professionals in the resolution of real world projects, which have challenging geometries within the typical constraints of practice (quality/time/cost) might
prove less useful. In design practice, problems and solutions are in a state of constant flux. By the time a designer adapts a pattern to the specific case, the context has changed, thus requiring the designer to adapt the parametric model to a new pattern. The ill-formed nature of challenging geometrical projects and the uniqueness of each architectural project, make it difficult to subject a project to a predefined design pattern.

As such, the simplicity and generality of the design pattern as a conceptual tool, may not be as effective in improving the flexibility in design modelling required in the resolution of complex problems in design practice.

As an alternative to the design pattern, Hudson proposes the use of the parametric model itself as a way to bootstrap ill-formed design problems as they develop. Hudson suggests using the parametric model as a tool for “acquiring, capturing and representing the problem description as it [is] developed.” [Hudson, 2008]

Hudson concludes “the process of developing a parametric model can begin with incomplete knowledge of the problem.” [Hudson, 2008] He positions his claim in rivalry to Burry’s and Maher’s claim “that everything needs to be considered or known at the outset of a parametric design process.” [Maher and Burry, 2003] Further, Hudson remarks “Parametric tools can provide an explicit means of conducting reflective tests that enable knowledge acquisition in order to develop and structure problem descriptions.” [Hudson, 2008]

Hudson supports his argument by the premise that “parametric tools can provide a representation for capturing existing knowledge and acquiring new knowledge.” [Hudson, 2008]

However it may appear that the differences between Hudson’s claim and Ma-
her’s & Burry’s claim might be more subtle than they first appear when taken at face value. For example, Maher and Burry purport that the parametric modelling paradigm “requires at the outset the development of an initial declarative schema, which then drives the design” [Maher and Burry, 2003] where as Hudson contends “the process of developing a parametric model can begin with incomplete knowledge of the problem.” [Hudson, 2008]

It is not clear if Hudson is assuming that, by Maher and Burry suggesting the need to develop an initial declarative schema; they are also therefore implying that it is compulsory for the designer to have completely finished the problem description and problem structure ahead of the declaration of the schema. Further, while arguably using Hudson’s theoretical framework one could conceive of a process by which the parametric model is used to acquire knowledge of the design problem as the design problem is evolving and being structured by the designer, it is not always the case that the designer is in control of the problem definition and structuring. For example, external factors such as client-driven changes, a change in the selection of fabrication contractor or a change in the materials due to fluctuations in the market price, etc, might have major unforeseen structural consequences over the design structure and hence over the parametric model’s declarative schema. Most of these external factors would have been difficult to anticipate during the early stages of design and are outside the designer’s control. While the model could be used to record changes as they accrue, the contingency of these external factors might induce major disruptive changes in the schema from which it would become difficult to recover, hence Burry’s paradox that the designer under these extreme circumstances would have no other recourse than returning to techniques of “erasure and remodelling”, a
practice which is the antithesis to the use of a parametric model.

From second inspection, it may seem that Maher and Burry made the observation that “parametric models require an initial schema from the outset” as a response to the limitations they found with translating their explicit design to a parametric representation. As they both indicate that they built their hierarchical parametric models by “first interrogating the explicit model, […] from Rhino NURBS modelling software, to gain an understanding of the geometry.” While the practice of building an explicit model first to gain an understanding of the relationships between the parts may seem contradictory to the use of the parametric modelling paradigm, there are design problems which elude understanding and which require a great deal of interpretation from the designers to understand the internal relationships of the parts to the whole. When designers are faced with these kinds of problems, the parametric modelling paradigm offers little help to the structuring of the parametric model. For these kinds of problems, explicit 3D sketches (in non-parametric software or animation packages) and physical prototyping are better vehicles to arrive to the resolution of the design schema.

Hudson, Maher and Burry are making reference to a similar but distinct issue in the use of parametric models for design modelling. Hudson is referring to how the parametric model can be used to both document the problem development process and support the problem structuring process by using the parametric model as a knowledge acquisition and knowledge representation tool. By contrast Maher and Burry are suggesting that having to declare an initial schema from the outset, presupposes having resolved all the geometrical relationships between the elements of a design.

Maher and Burry touch upon an important aspect of the intersection between
“designing the design” and the character of architectural design. Burry suggest that the schema presupposes an intentional clarity in how a designer envisions their design intent as “the messiness of design or its inherent wickedness as a problem space by definition thwarts such otherwise intentional clarity [Rittel and Webber, 1973]” [Burry, 2003]. This for the most part has been absent in the traditional process of design. As Lawson indicates, “Design problems cannot be comprehensively stated” at any stage of the design process and therefore, “Design problems require subjective interpretation” [Lawson, 2006].

For this reason, Burry & Burry conclude that the use of a schema as a design representation tool is “not only too slow and cumbersome for the very early stages of design, especially at the stages of ideation or conceptualisation, but the level of change in conceptual design is so fundamental that it can never conform to a relational schema” [Burry and Burry, 2008].

Once the fundamental constraints of the design process are taken into consideration from the outset during the development of parametric models, it becomes difficult not to sympathise with Maher and Burry. The role of the parametric model becomes easier to implement in design practice when the declaration of the schema is placed as an activity which succeeds the design conceptualisation.

The role of the parametric model during the wider process of design becomes more evident if the design process is characterised and framed as an activity which precedes the declaration of the schema and not as an activity which occurs after or during the declaration of the schema. Any tool used in the modelling of design will therefore have to be aware of these fundamental constraints of the design process.

As it has been suggested by the commentary of both Burry and Shelden while
the architectural design process in its totality cannot be comprehensively stated
and addressed through the use of computers in design [Lawson, 2006], arguably
elements of the architectural design process can be partially stated. For those
partial, localised and well-formed design problems that can be stated and there-
fore, declared, the parametric modelling paradigm can prove to be most useful.

After revising the perspective of the leading researchers on this subject, it
appears there is a gap in the literature addressing pragmatically how designers
improve the parametric modelling paradigm. The research begs the question
of how compulsory is the need to use a highly restrictive schema in the devel-
opment process of parametric models and if there are strategies which could
be devised for alternative representations of complex design problems by other
more flexible means than with the use of schemas. Hudson’s focus on the para-
metric problem description and parametric problem structuring components of
the design process is a theoretical tool for making sense of design theory from
the perspective of the parametric modelling paradigm. While his tool addresses
the issue of how designers acquire, represent and modify information by using
parametric models in their off-the-shelf modality and how designers structure
problems from this information; Hudson’s theoretical tool does not resolve the
dependence of parametric models on declarative schemas which restrict an other-
wise fluent design process. It seems that the answer to the resolution of whether
parametric models could be devised without the use of the restrictive schema
might require not a theoretical response, but rather a technological response, as
the limitations found in the current implementations of parametric models are
technological and not necessarily in our understanding of the issues with the use
of the technology.
2.3 THE GRAPH

As Burry and Burry, point out “It is a rude awakening when our mutable, morphing digital model falls foul of the constraints of its representation” [Burry and Burry, 2008, p307]. The hyperspace produced between the networks of relations in parametric graphs, both facilitates and hinders “flexibility” in design modelling [Burry and Burry, 2008, p307]. Burry and Burry indicate that there is a naive expectation that by linking anything to anything, the mutual impact of the constraints represented in our models will somehow find resolve on their own [Burry and Burry, 2008, p307].

However, as Burry and Burry suggest, the difficulty to “predict model performance or which parameter value combinations result in viable model manifestations” [Burry and Burry, 2008, p307] is a major hindrance of the hypergraph\(^\text{13}\) produced by the tangled networks of relations within parametric models [Burry and Burry, 2008, p307]. As it is being alluded by the attention Burry and Burry give to the hypergraph on their writing of parametric modelling “inflexibility”, it appears the brittleness and the constrained variability designers are facing when using parametric models, could be attributed to the inherent characteristics of adjacency graphs or complex networks used internally by the parametric design software as a design representation system. Adjacency graphs in the form of directed or undirected acyclical graphs have advantages which are favourable to the representation of complex design with computers. As pointed out by [Aish and Woodbury, 2005, p4] graphs are useful data structures which have been employed successfully in the implementation of spreadsheet softwares, project

\(^{13}\)In mathematics a hypergraph is a generalisation of an adjacency graph in which an edge can connect to any number of vertices. [Illingworth, 2001]
management tools and dataflow programming languages [Aish and Woodbury, 2005, p4]. [Aish and Woodbury, 2005, p4] present some of the advantages this representation bring in computational terms:

- The use of adjacency graphs enable designers to effectively implement a propagation-based constraint system which is comprised of two algorithms, one for ordering the graph and one for propagating values through the graph.

- The propagation-based structure enabled by adjacency graphs, enables the use of an *update algorithm* for updating specific values across the network, and a *display algorithm* for displaying the node symbolically\(^\text{14}\) and in the 3D viewport.

- Graphs can be used to embody decisions about chosen relationships in a model. The use of graphs enables the parametric design software to compute values induced from the structure of a relationship network. By deferring decision making, graphs enable a higher level of abstraction in work.

As indicated by [Aish and Woodbury, 2005, p4] above, the deferring of decision making to the later stages of a design and the higher-levels of abstraction afforded by the graph, presents advantageous properties to designers who do not wish to over-commit to their designs early in the conceptualisation phases, particularly when these models are undergoing iteration and refinement. However

\(^{14}\)The two dimensional diagram used in Generative Components and used by default in Grasshopper as the representation of this complex network of nodes and relations.
as indicated by Burry and Burry while “This mode of using the computer to record relations ahead of dimensions aims to reduce the severity of impact for each constraint, to defer precise form and space making through prioritising the formulation of a schematic graph of relations. […] the graph of relations […] takes a form of its own and the mutual impact of the constraints can become a complex system in which the number of variables defies mathematical modelling.” [Burry and Burry, 2008, p307]

As pointed out by Burry and Burry above, by prioritising the formulation of a schematic graph of relations ahead of specifying dimensions to the model, the parametric modeller aims to reduce the severity of the mutual impact of the constraints represented in the model. However this front-loading of model structuring considerations ahead of design decision making, impacts substantially the design modelling process. [Aish and Woodbury, 2005, p11] warns that “We should neither under-estimate nor under-value the change to the structure of work and design process[…]” [Aish and Woodbury, 2005, p11] which the inclusion of the parametric design software has over the conventional design process.

This complex system which is embodied by the hypergraph of relations in a parametric model, and which defies mathematical modelling as indicated by Burry and Burry is what mathematicians and computer scientist define as a complex network [Boccaletti et al., 2006, Strogatz, 2001]. Complex Networks are networks whose structure is irregular, complex and dynamically evolving in time [Boccaletti et al., 2006]. As indicated by [Strogatz, 2001] “researchers are only now beginning to unravel the structure and dynamics of complex networks” [Strogatz, 2001]. In his account of complex networks [Strogatz, 2001]
indicates, the importance of characterising network anatomy, given the anatomy of a network has strong influences over the network’s function. In response to this, [Strogatz, 2001] presents the following characterisations of network anatomy which arise from the use of graphs or complex networks as representation systems:

- Graphs are *inherently difficult to understand*.
- Graphs have *Structural Complexity*: their associations could be made up of an intricate tangle.
- Graphs have *Connection Diversity*: the links between nodes could have different weights, directions, signs and mathematical expressions.
- Graphs have *Node Diversity*: there could be many different kinds of nodes representing a system. For example, in parametric design software there are hundreds of kinds of objects which could be used as a node in the graph.
- Graphs have *Dynamical Complexity*: the state of each node could vary in time in complicated ways.
- Graphs suffer from *Meta-complication*: the various complications can influence each other.

As it can be seen from the broad list presented above by [Strogatz, 2001], graphs are borne with inherent complications out-the-box. These complications are endemic of the graph. As [Aish and Woodbury, 2005, p4] presented above, graphs are convenient for implementing parametric design software, however
this convenience of computing due to its effectiveness at making design decision-making explicit, is taxing the design process significantly. For example, many design problems are best represented through chicken-egg cycles, bidirectional constraint modelling provides this ability by making it possible to swap driver with driven or independent variables with dependent ones [Kilian, 2006]. However the acyclic directed graph of parametric design software does not enable parametric design software to comprehensively represent all kinds of design problems. For example, parametric design software cannot represent bidirectional constraints\(^\text{15}\) in a model [Kilian, 2006]. This does not mean that design cannot be represented through bidirectional constraints, but rather, that the selection of the acyclic graph as a design representation tool has restricted our palette of design solutions to only those that are easily accommodated by this form of representation. In a sense, the acyclic graph has become like Maslow’s hammer who posits that “[…]it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail” [Maslow, 2004]. The restrictions of the acyclic graph, have somehow forced the parametric design process into framing all design problems and solutions as if they were graph-like problems, ordered and imbued with hierarchy traits which are not necessarily borne out of the design problem at hand. However, while these ordering problems and hierarchical restrictions are necessary for the computational model to work properly, they are not an indispensable element of the conventional design process (independent

\(^{15}\)In a limited sense, CATIA and Solidworks enable the representation of bidirectional constraints in two-dimensions through sketch based modelling. CATIA also provides assembly level constraints which do enable bidirectional constraints modelling, however the parts must be fixed for the constraints to work properly.
from the parametric design process).

In feature based PLM CAD software such as CATIA and SolidsWorks, nodes and graphs are represented by features and assembly hierarchies respectively. While these softwares do not refer to graphs directly and instead prefer to call the node a “Feature”, the underlying structure of the model is an acyclic graph persisting associativity. Due to the use of acyclic graphs as a design representation, “feature” based software also inherits the same negative traits of graphs as characterised by [Strogatz, 2001].

For example, the burden or encumbrance the acyclic graph places over the designer and his design flexibility is succinctly pointed-out by the following SolidWorks patent excerpt [Rothstein et al., 2009, BACKGROUND OF THE INVENTION]:

Often the design engineer discovers that the feature order results in the generation of a physically incorrect part. The design engineer is then burdened with deleting and re-creating portions of the part or the entire part, re-ordering the features that constitute the part by manipulating one or more feature locations in the overall historical order of features, or in some other manner, which may be tedious, correcting the inaccurate geometry. The design engineer may be required to spend an enormous amount of time and effort during the 3D modelling process controlling the feature order and the feature order’s affect on the final geometric representation of a part. Moreover, while building a part, the order in which a design engineer should introduce features and direct the system to perform operations is not always intuitive. Many times the design engineer
has invested a great deal of time designing a part before discovering that the features should be introduced in a different order. When the design engineer realizes that the feature ordering did not achieve the desired result (e.g., the desired geometric result), he or she must modify the definition of the part, for example, by rearranging the hierarchical structure of the part. Due to the problem of introducing features in a particular order, modelling a part may require a great deal of planning and expertise. The design engineer must determine the correct ordering of features before creating the features to obtain the desired geometric result. The ordering problem is present throughout the modelling process. For example, features introduced latter in the design process may affect features that do not share common boundaries, due to parametric relationships or other inter-relationships. The difficulty of the ordering problem may increase as the modelling process progresses because as a part becomes more complex, the design engineer has more difficulty determining the correct feature order. [Rothstein et al., 2009, BACKGROUND OF THE INVENTION]

In this excerpt, the authors refer to the pitfalls of the dynamically evolving structure of parametric models over time and its inherent feature ordering problem. Feature ordering in feature-based parametric design software is the equivalent of the Structural Complexity graph trait described by [Strogatz, 2001]. The dynamically evolving structure of the model is simply another way of describing the Dynamical Complexity described by [Strogatz, 2001], this Dynamical Complexity emerges from the kinds of changes which the graph undergoes as the
designer makes changes to his design intent over time. From the patent excerpt in [Rothstein et al., 2009] we can gather the following “inflexibilities” which hinder the parametric design process due to its graph based representation:

1. The feature ordering problem of parametric models make it difficult to determine if the model results in physically incorrect geometry—this implies it is difficult to determine if the model produces a coherent result by interrogating the graph.

2. When the designer identifies that the model is not in agreement with the designers design intent, the designer must undergo the following manual tedious activities to correct the inaccurate geometry:
   - delete and re-create portions of the model or the entire model.
   - re-order the features that make up the model by manipulating one or more feature locations in the overall historical order of features.

3. Controlling the feature order and the impact this order has over the design intent is time onerous and effort intensive.

4. The order in which a designer should introduce features and direct the system to perform operations is not always intuitive.

5. It is possible for a designer to invest a great deal of time designing a model before discovering that the features should be introduced in a different order.

6. If the designer realises that the feature ordering does not achieve the desired geometric result, the designers must modify the definition of the model, by rearranging the hierarchical structure of the model.
7. Modelling a part may require a great deal of planning and expertise, due to the feature ordering problem.

8. The feature ordering problem is present throughout the modelling process.

9. The difficulty of the ordering problem may increase as the modelling process progresses because as a part becomes more complex, the designer has more difficulty determining the correct feature order.

Moreover, [Baran et al., 2014, p3] points out that “Although, a CAD system may provide a feature management tool to help a design engineer rearrange the history of features included in a part, the design engineer is encumbered with analyzing the feature history and re-ordering the features in the part hierarchy as necessary to ensure that the part is geometrically correct” [Baran et al., 2014, p3].

To summarise, the entangled hypergraph of relations which emerges during the parametric design process, emerges not from the design problem at hand but rather from the selection of the graph as a computational representation system. Graphs inherently are difficult to understand, present structural complexities, have dynamical complexities over time, have combinatorial complexities (which emerge from its node diversity and connection diversity) and suffer from metacomplications which have taxed mathematicians and other professionals trying to use graphs as representational data-structures to manipulate computationally complex problems [Boccaletti et al., 2006, Strogatz, 2001]. Despite graphs having many computational advantages which make them effective for implementing propagation-based or constraint-based modelling systems; using the graph as a “User Interface” leads to a confusion between issues which emerge from
the use of graphs (with their negative traits) and issues which emerge from the design problem at hand.

As was presented by [Rothstein et al., 2009, BACKGROUND OF THE INVENTION] in the SolidWorks patent excerpt, the dynamic complexities of structuring feature graphs and their difficulty in representing the design intent as well as the sluggishness of the feature graph in mapping the evolution of the design intent are issues which hinder flexibility in design modelling today.

The examination of this literature review suggests that there is value in the search for alternative representations to the graph for mapping complex design problems, as well as methods to avoid the pitfalls introduced by graphs into the design modelling process where the use of graphs is inevitable.

From this literature review on parametric modelling use in design modelling, three major themes emerged as the key issues with regards to the use of parametric modelling technology in design practice. The first theme deals with the scope of applicability of the parametric modelling paradigm within the larger design process, the second theme deals with the ordering of the problem structuring phase and the schema building phase within the process of representing design intent with parametric models, and lastly the restrictions to design practice from the use of the schema as a design modelling interface in the development of parametric models. Of these three themes this thesis will focus only on the latter: “the restrictions to design practice from the use of the schema as a design modelling interface in the development of parametric models”.

From the theme selected above, the literature points to a gap in knowledge that could be addressed through the development of technological strategies which enable designers to respond to the limits of flexibility found with the parametric
modelling paradigm. Such strategies might enable designers to overcome the restrictions to design practice from the use of the schema as a design modelling interface.
Chapter 3

STANDARD APPROACH

3.1 INTRODUCTION

This chapter will address approaches to flexibility in parametric modelling through the use of standard off-the-shelf software.

This chapter will expose two sub-strategies. The first sub-strategy will address the use of off-the-shelf parametric software with internal code. The second sub-strategy will address the use of off-the-shelf parametric software with external code.

In the first sub-strategy, the parametric model includes computer code as part of the model. In the second sub-strategy, the parametric model is controlled from external computer code which belongs to a larger automation workflow.

The “Standard” software satisfies the needs of a large common user base. It is difficult for standard software to fit the needs of bespoke design problems.

To solve bespoke design problems, end-users need to extend the standard softwares. Most standard software provide at least two internal ways for the end-
users to extend them. One way is by enabling the software to co-execute custom computer code inside the main software. This software executes in parallel with the main application. The software can execute as part of a plug-in system or as a script interpreted by a script interpreter. Most standard softwares provide a facility for scripting or a plug-in system for extensibility. The second way is by enabling the application to be automated from an external running process or application. This capability enables the application to participate as subprocess of a running workflow.

In the following sections, the “internal code” method is the internal way of enabling the end-user to extend the application functionality.

The “external code” method is the external way of enabling the end-user to extend the application functionality.

Most parametric modellers have a facility which enables the parametric modelling system to interoperate with external software directly. This facility enables external software to automate the production of parametric models. Similarly, most parametric modelling software can enable the execution of computer code embedded in the model itself. Through this facility, the parametric model can include computer code as an element of the model.

The external code and the embedded code enable the parametric modeller to escape certain limitations and restrictions, such as the current limitation parametric modellers face when dealing with restrictive schemas.

Figure 3.1 shows different approaches to flexibility from code use in a parametric model. The “hybrid” approach combines computer code with traditional parametric models. The “pure code” approach relies exclusively on computer code as the method of representing the design intent. Lastly the “pure paramet-
ric” approach relies exclusively on parametric models for representing the design intent through graphical geometrical and graph like representations in the user interface.

The “hybrid” approach enables the designer to decouple architectural geometry problems (tectonics) from computer programming problems (logics). With this method, professionals working on complex shaped and articulated projects can effectively solve geometrical issues in 3D using visual methods which are natural to the process of design while the computer programming component that instantiates the three dimensional features can be debugged using traditional text-based debugging methods.

Users with sufficient parametric model-based knowledge can contribute in the building of the smart features. While one user builds the smart features, another user can build the representation of the design intent logic through computer code.

This thesis will not be dealing with the “pure parametric” approach, or the “pure code” approach as both of these approaches are equally restrictive to users of parametric modelling; as their approach to representing the design intent, may be too restrictive for designers. This thesis is working against these two modes. The “pure parametric” approach introduced by Burry and Maher as “designing the design”\(^1\), and the “pure code” method. The “pure code” method requires the designer to work in a text-based computer code representation. To work

\(^1\)While “designing the design” could also refer to the meta-design process of designing the structure of a model either through a parametric model schema or developing computer code to represent the design intent. “designing the design” or “meta-design” is taken here to signify modelling only through the keyboard/mouse interface of standard parametric modelling software.
Figure 3.1: Three approaches to parametric modelling. Indicating which areas of the design-logic are maintained by either a scripter or an end-user, as well as who maintains topological variations in the parametric model.
This chapter will examine in detail two sub-strategies within the “hybrid” approach.

The first sub strategy uses computer code within the parametric model “internal code”. The second sub strategy uses external applications to control the parametric model “external code”.

This chapter will address in further detail both the “internal code” and “external code” method of approaching flexibility with the use of a standard off-the-shelf parametric software.

3.2 INTERNAL CODE STRATEGY

3.2.1 OVERVIEW OF INTERNAL CODE STRATEGY

This section will expose a sub-strategy for use with standard off-the-shelf parametric modelling software. The strategy will address the flexibility gained from the inclusion of user written computer code within a parametric model. This strategy takes advantage of the facility most off-the-shelf parametric modelling software have in enabling user written computer code to live inside the actual parametric model. This “internal code” is able to interact with the parametric geometry and parameters in the model. The code executes from the interaction between the objects and from changes in their values. Following this section, two case studies from the Sagrada Família Basilica will demonstrate the use of this strategy.

The main definition of this strategy is the use of “internal algorithms” written by the end-user embedded within the parametric model. These algorithms can
create, edit or remove parametric objects, relations and parameters. The goal of this strategy is the development of responsive parametric models. These more responsive parametric models can run and react to designer-driven input increasingly in real-time. With this instant feedback, designers can respond to design changes quickly. This responsiveness allows designers to use models actively during the earlier stages of design.

The use of this strategy has many advantages and disadvantages. First I will expose the advantages and then the disadvantages.

**ADVANTAGES OF INTERNAL CODE STRATEGY**

The “internal algorithmic” layer lives inside the parametric model. The author of the “algorithmic layer” and other members of the design team can re-use the knowledge represented in the model. Re-usable models can be shared among team members. The knowledge embedded in a parametric model is shareable if the model is shareable. Because algorithms become part of the model, they become shareable knowledge. As the model evolves, so does the internal algorithms within the model. In the “internal algorithm”, an algorithm is an object. This object is editable by the user as any other object in the model. There are differences between this object and a normal object. While normal objects can only make changes to their internal properties, the algorithmic object can only make changes to the properties of the model and to the properties of other objects. The algorithm reacts to a change in its parent object. The algorithm can react to changes in parameter values or to other objects. It can also react to changes in the content of the model. Adding or removing an object, can trigger the execution of an algorithm. The user then determines which action the algo-
algorithm reacts to. An algorithm can only react to one type of user action at a time. While the algorithm is triggered by only one user action assigned by the user; the algorithm has access to all the parameter values of the model. Because the algorithm has access to all parameter values; the algorithm can determine based on these values if the source action warrants a response action.

By using the “reactive” capacity of these “internal algorithms” the parametric models can now provide near real-time feedback to the end-users, with this increased response from the model, designers can now interact more fluently with the parametric model. Most of the issues of the difficulty designers face when “designing the design”, are addressed by the use of an “internal algorithmic” layer. The “internal algorithmic” layer takes care of the restrictive parametric schema, by enabling the end-user to modify the schema by other means other than the schema itself. For example, an “internal algorithm” can be linked to parameter changes, which trigger a series of complex schema editing functions, enabling the end-user to abstract the process of creating complex schemas by making changes to a single design parameter which triggers the complex process.

The use of the “internal algorithm” facilitates the process of knowledge capture, particularly procedural knowledge. Currently off-the-shelf parametric models only enable users to map linear chains of operations. This chains of operation may not be circular. For example, when two objects are both parent and child to each other. While circular relations are not permitted through off-the-shelf software, the complex management of circular parent-child relationships can be handled effectively through the use of an algorithm for its resolution.

At the moment, off-the-shelf parametric modellers do not provide effective
means to embed complex design rules and to check their consistency. Only simple rules can be embedded within a model. These rules can only map simple conditionals. By using the complexity that is permitted by an “algorithmic” layer, the designer embeds complex design rules within a parametric model.

LIMITATIONS OF INTERNAL CODE STRATEGY

While the use of “internal algorithms” enable the designer to map complex procedural knowledge, not all design problems are easy to map using “algorithmic” layers. Equally, the use of this strategy is experimental while the strategy works for some cases, it might not work for others.

One of the main drawbacks of this strategy is that it relies on the user having scripting knowledge. If scripting knowledge is not an issue, this strategy presents many advantages for flexibility in design modelling.

The code editing environment of the “internal code” is not as helpful as external code editing environments such as Microsoft Visual Studio 2008. For example, the knowledgware “reaction” system found in CATIA, has a very limited code editor (see Figure 3.2), which provides little help to the end-user in debugging the code or in writing the code.

The documentation for learning how to use the “internal code” scripting language is limited. For the “internal algorithm” there is limited documentation of the knowledge engineering language used. The obscure programming language used in CATIA’s “reaction” system, for example, means there are limited means for the end-user to learn on his own how to write algorithms using this system. At the moment, this knowledge is acquired from the exchanges between expert and novice.
The objects generated through an “internal algorithm” do not mix well with objects generated without it. When “pure parametric objects” depend on generated objects, there is a possibility that the update mechanism of the parametric model may fail, as pure parametric objects do not have the intelligence to adapt to algorithmic changes. While there are no issues with generated objects depending on “pure parametric objects”, it is advisable to combine only “algorithms” with “algorithms” and not “algorithms” with pure parametric objects. Therefore, only “pure parametric objects” may be the parents of “algorithmically generated
objects”. “Algorithmically generated objects” may not be the parents of “pure parametric objects”.

While this strategy enables the designer to map complex design problems more effectively than through the use of pure parametric models. This strategy is still more difficult to use and less effective when compared to the “parametric model with external code” strategy.

The following sections will report on projects which tested the strategy outlined in this section. The accounts of the Sagrada Família Hyperbolic Bridge and the Sagrada Família Passion Façade Colonnade, present with more detail the specific advantages and limitations which can be gained from implementing this strategy in practice.
3.2.2 SAGRADA FAMILIA HYPERBOLIC BRIDGE

The Sagrada Família Basilica Hyperbolic Bridge project, required the development of an end-user responsive parametric model to model the pre-design of Sagrada Família Pedestrian Bridge (Pont) for Antoni Gaudí’s Sagrada Família Basilica in Barcelona. The Sagrada Família Pedestrian Bridge Portal connects from the Tower of Jesus Christ (Main Central Tower) to the 4 Evangelist Towers (Mathew, Mark, Luke and John) that are located at intervals of 45 degrees of rotation. The Pedestrian Bridge floor is elevated at 85 Metres. The purpose of the bridge is to assist with the flow of traffic up to the observation deck at approximately 170 Metres.

The model had to enable the design team led from Melbourne, to develop

---

2The work presented here is Work In Progress, the project is currently ongoing
various parametric model prototypes to assist in the design development of the bridge and its panellisation.

The parametric modelling schema of the bridge had to allow disruptive changes that would not be possible through a pure-parametric approach. From the outset it was clear that the kinds of variants that would be required from the system would not be possible using the standard off the shelf parametric software as is. The parametric model was built having the requirements of extreme variability from the outset.

This project, raised the issue of shared tool-making between end-users and scripters. In this case because of the limitations of the standard interface in providing the end-user with a modelling environment in which they could represent the design intent. The situation created the opportunity for the development of an embedded software tool. This software tool had to be different from other tool making exercises, as the tool had to be embedded within the parametric model. It also had to provide a customised user experience which was transparent to the end-user. Manipulating the extended parametric model would have to be as natural as manipulating a pure parametric model, although this model would provide extra features which are not available in pure parametric models.

The project used the research lessons from hindsight of the Passion Façade retrospective analysis exercise. The first experiment developed with the use of CATIA knowledgeware “Reactions” on the Passion Façade facilitated parametric models which would reconfigure the internal parametric schema upon user driven parameter value changes.

The bridge model needed to accommodate the complex design intent requirement of developing the four quadrant cotangent hyperbolic surfaces. The surface
then had to be subdivided into panels following the directrixes of all four quadrants. The parametric model for the bridge required the panel geometry to vary. The team needed to be able to replace the panel model as the design evolved. For this reason the four-hyperbolic-quadrant geometry lived on a separate file from the panel geometry. The “reaction” glued the hyperbolic-quadrant system to the panelling system, replicating the panel geometry into the hyperbolic-quadrant support geometry. In a pure-parametric model the designer would have to manually replicate each panel for each design variant. This would render the model restrictive for even simple design variations.

Figure 3.5: Sagrada Família Basilica Bridge Variants. This figure indicates different variants obtained from simply changing parameters in the user-control inputs. Because of the use of “Reactions” the system can produce extreme topology variations.
Without these complex interfaces to parametric modelling, the parametric modelling paradigm would be useless in evaluating these kinds of complex design problems.

The parametric model for the Sagrada Família Basilica Bridge enabled the design team to make early design decisions, by enabling them to manipulate low level controls while creating a high-level of resolution variations. The model was built to evolve with the design solution. Flexibility was built-in for future changes in the panelling geometry and in the way the model would be positioned and configured. It enabled the use of the CATIA knowledgeware optimiser for optimising the placement and configuration of important driving parameters which where difficult to map directly in the model.
3.2.3 SAGRADA FAMILIA PASSION FAÇADE

PROJECT DESCRIPTION

As part of an ARC Discovery project titled “Challenging the inflexibility of flexible models” I revisited the model and complex context for the Passion Façade parametric model of Antoni Gaudí Sagrada Familia Basilica. This exercise sought improvement where possible of the parametric modelling protocol that could be used for resolving these kinds of challenging geometries and dynamic design representation models.

Similarly to the retrospective analysis of the Sydney Opera House developed by Burry and Murray [Burry and Murray, 1997], the exercise which will be described in this section, required me to devise a retrospective analysis to assess the impact CATIA’s Knowledgeware “reactions” would have had on the design process of the narthex, or upper colonnade, of the Passion Façade Flexible model. The original model was developed using CATIA without the use of scripting aids, through a pure parametric modelling approach by Burry and Burry [Burry and Burry, 2006]. The difficulties of the model had to do with how to maintain the schema identified by Burry and Burry coupled with the subtle and sensible refinements that the model had to accommodate in response to the requests from other members of the design team. As such, the models developed by Burry and Burry had to withstand the subtle “what if” variations required by the other members of the design team.

The expressions of Gaudí and the interpretations of the surviving original drawing photograph from his time and a 3d plaster model interpretation from 1980’s was also analysed. The parametric model was only a means to an end,
and not the end in itself. The challenge was in the structuring of a parametric model schema in such a way that these models could represent the relationships apparent in the photograph and allow for subtle variations induced by the sensibilities of a designer to conform to the site on the church.

While at first it may seem capricious to define models whose logic originates at the heart of a designer’s sensibility, it is not unsystematic to follow intuition during the design process. Architects have relied on their intuition for as long as the profession has existed. As parametric models currently stand they are difficult to operate in an intuitive way.

This project will serve as an implementation example of the Reaction Based Strategy documented in the preceding sections. The Passion Façade project was the first experiment of the thesis and the source for the Reaction Based Strategy. While this was the first project of the thesis using reactions, it was not the first project I had worked on where I had to apply the use of Reactions. The Reaction Based Strategy started with the Building Core Modeler\(^3\). During the building core modeller the strategy had been prototyped and made ready for use in a real-world scenario, the Passion Façade redux project extended the work in the areas of usability, interaction and automatic instantiation of user-defined features.

---

\(^3\)The Building Core Modeler, was a specialized tool commissioned by Skidmore, Owings & Merrill (SOM) New York, whereby Gehry Technologies Inc developed a tool for the preliminary design of building cores which would enable SOM to have a “working core” prior to the engagement of a vertical transportation consultant much later down the project lifecycle. My role within this project, enabled me to develop the CATIA\(^\text{TM}\) automation system as a knowledgeware “Reaction” system, combining prebuilt intelligent modules configured from an intelligent Excel\(^\text{TM}\) proforma.
FORENSIC PARAMETRIC MODEL

As part of the research project “Challenging the Inflexibility of the Flexible Model” I revised one of the case-studies on “flexible models”, with the purpose of revising the flexible parametric modelling process used in the creation of the models and also to devise a retrospective analysis on how to incorporate “subtle variation” in the model in a non-disruptive way. This enables the kinds of variations that the model was exposed to during its actual use, but where it had presented the highest level of difficulty in maintaining the design intent requests of the design team.

This retrospective analysis first required me to revise the work done by Burry and Burry on the models, and to understand the parametric schema of the project, the model relied heavily on the use of hyperbolic paraboloids and revolved surfaces, the geometry also relied heavily on booleans and intersections over complex arrangements.

The tackling of this model through parametric technology required me first to understand the geometric codex and then to understand how to use the codex parametrically within the model.

DIRECT MODELING

During the creation of the parametric model for the passion façade, Burry and Burry developed the “schema” and discovered the mathematical and geometrical logic behind the construction, from the surviving photograph [Burry and Burry, 2006] (see Figure 3.6).

The fundamental crux of the passion façade parametric model, lies at the reconciliation between mathematics, geometry and intuition.
Burry and Burry, had to reverse engineer the model as observed from the only surviving photograph of the drawing, the original drawing in the photograph had been destroyed by anarchists during the Spanish civil war, and there where no other sources of evidence beyond this single photograph of the drawing prior to the destruction of the originals. From this photograph Prof. Burry and Dr. Burry devised a schema to manually optimise the parametric model until it matched the geometry of the photograph through an extensive process of reverse engineering [Burry and Burry, 2006]. The model was used to compare and validate the multiple hypothesis testing, about how to structure the model to resemble as closely as possible the original photograph representing Gaudí’s original design intent.

Knowing Gaudí had used the hyperbolic paraboloid surface as the basis for most of the surfaces of the project, Burry and Burry predefined the surfaces and
the geometries of the model to be hyperbolic paraboloids and elliptical hyperboloïd surfaces.

Having defined the surface classes prior to the development of the model makes the project inherently parametric. Parametric objects representing the hyperbolic surface class can be developed as a template for all the desired properties in the hyperbolic surfaces. This kind of parametric template is referred to as a Paramorph by Burry [Burry, 1999]. This template can be used for the creation of all the hyperbolic surfaces of the model.

The issue then was about configuring these models in space and second, to relate them to one another mathematically adhering to the dimensional properties observed in the photograph. How does one respect geometrical consistency while adhering to a mathematical progression obtained from the measurements of a photograph? For this Jane Burry decided to identify the mathematical functional that defines the progression, instead of treating the progression of numbers as a list of random numbers, the source of the spatial pattern followed a geometrical rules, emerging from the trajectories of parabolic curves and elliptical curves by using a curve fitting software [Burry and Burry, 2006].

After confirming that the mathematical progressions of the spatial patterns in the stepping cornice were in fact the result of a parabolic distribution, it became easier to map parametrically this progression, thus enabling the team to induce variations outside the original progression obtained from the photograph.

The last component of the model is the implementation. While the “schema” and the context are in place, an implementation that embodies these two must be created in such a way that it respects the design-logic identified in the steps mentioned above while providing intuitive “user-controls” which enable the modeller
to tune the model by the design team.

After having developed the described model, it needs to be tweaked inside its context surrounded by the other geometrical components enabling it to snap into place, if modelled correctly.

This last tweaking phase had to withstand the manipulations of the “user-controls” developed to customise the model. Accommodating these “user-controls” would serve as the ultimate challenge to the parametric model, for all requests of variation, had to be accommodated by the model in an interactive manner.

**DISCUSSION OF SAGRADA FAMILIA PASSION FAÇADE**

During the production of the model Jane Burry had relied on the pure parametric modelling protocol with mathematical formulae for the design of the model and for executing and reviewing the subtle variations requested in on-site meetings.

It was immediately identified by (Burry and Burry), that parametric modellers in their off-the-shelf state are not able to easily handle parametric models that can persist a set of logical rules governed by either geometry or mathematics while simultaneously enabling a designer to satisfice the caprice of having an intuitive control over the spatial arrangement.

They also identified that in order to implement such a model, it would require to have a scripting overlay over the parametric model, enabling the designer to define algorithmically the logic of the model and therefore increasing the level of control over the subtle variations one can produce.

While it was clear the solution to having control over both the geometrical and mathematical requirements would be in the form of an algorithmic controller, the overlay would not enable the designers to “pull and push” through direct mod-
elling features governed by the hand-eye coordination of a designer operating the model while still satisfying all the geometrical constraints and relationships between parts.

I suggested the use of an underlay, the Reaction Based System as used previously for the Building Core Modeller, where the designer can “push and pull” the model components while the model would React and tweak the parameters governed by the geometrical and mathematical rules identified by Burry and Burry in the previous exercise.

I set-out to create a Reaction Based System for the Passion Façade, conforming to all the requirements defined in the earlier exercise and with the new additional requirements of the “instantaneous update” provided by the use of reactions.

The Reaction Based System would need to propagate, delete and edit objects automatically, depending on the changes induced to the project driving geometry.

This suggests that parametric modellers are limited even when having full access to the algorithmic processes within parametric models. The limits had to do with the speed of execution and the perception of flexibility that comes with models that do not respond instantaneously to the designers actions (slow updates).

While the models did react to designer input and enabled subtle variations to the models spatial and non-spatial configuration, the model developed did not satisfy in full the requirements of the designers.

The Reaction Based implementation of the façade project, demonstrated successfully the use of an algorithmic controller used as overlay to the parametric
Figure 3.7: The Reaction Based System for the Passion Façade. This figure shows in the CATIA specification tree the levels of control which were added to the model to provide “responsiveness”

model, the algorithmic controller enabled the design logic to persist under variations induced by the designer at will.

The evaluation of the strategy identified how the success of such a strategy is dependent upon issues of usability. The “slow updates” of the reaction based model while enabling the creation of a level of intelligence in models not possible through traditional parametric modeling, fell short in satisfying the users desire to flex the models at will and obtain expected outcomes.

This does not mean that the reaction based strategy failed completely, but rather that when examining issues with user control over algorithmic processes, we must stress the consideration of usability factors as a wild-card in the de-
sign of such a system. As parametric models currently stand, “direct modelling” features that enable the end-users to directly manipulate the model can only be applied to prepackaged components. There is not yet a mechanism that enables the end-user to associate reactions with “direct modelling” features enabling the designers to manually tweak an algorithmic controller directly in the 3D viewport.

While in the the passion façade experiment the reaction based strategy was not fully tested to its limit, the lessons learned from this small experiment crossed-over to the Sagrada Família Basilica Hyperbolic Bridge Experiment. The Passion façade experiment provided the platform for experimentation in the application of reactions for the control of subtle variations through design-led intuitive manipulation.

3.2.4 SUMMARY OF INTERNAL CODE STRATEGY

In this section I examined the Internal Code Strategy and two specific case studies which have used this strategy in practice. I will summarise in this section the key elements of the internal Code strategy section.

In the Sagrada Familia Hyperbolic Bridge project we examined the "Reaction" a CATIA/Digital Project specific technology which enables to customise the update mechanism of a parametric model. Through the use of "reactions"—embedded computer code inside parametric models— I was able to achieve a greater level of user responsiveness and interactivity than the current process afforded by parametric design software in its off-the-shelf state. The panelling process in this project demonstrated how processes which were typically executed by auxiliary external tools, can now also be executed concurrently within
the parametric model, avoiding interruptions in the parametric design workflow.

The Sagrada Familia Passion Facade project demonstrated a particular application of this strategy— to tweek/tune complex geometries in space— the "Reaction" system enabled in this case to manipulate complex parametric models through the more natural interface of hand-eye coordination.

This strategy demonstrated how parametric models can be made more responsive to user interaction and how this higher level of responsiveness increases the designers control over the design process. Equally this strategy highlighted that the use of "reactions" is limited to problems which can be made explicit as computer code and which rely on wireframes. This strategy can only resolve moderate levels of complexity. The strategies which will follow in the "External Code Strategy" section will present methods to approach problems with extreme levels of complexity.

3.3 EXTERNAL CODE STRATEGY

3.3.1 OVERVIEW OF EXTERNAL CODE STRATEGY

This section will expose a sub-strategy for use with standard off-the-shelf parametric modelling software. The strategy will address the flexibility gained from the manipulation of the parametric model as part of a larger automated workflow. Following this strategy overview, four case studies illustrating the use of this strategy will continue. This exposition will allow the understanding of the applicability of the strategy on a varied range of practice contexts and design problems. It will also focus on the applicability of the strategy and its capacity to enable flexibility within parametric modelling, responding directly to the
limits found with “Designing the design” or “meta-designing” — that is the manual development of parametric schemas— or what could be considered a pure parametric modelling approach where by the designer does not use any of the scripting facilities provided by most off-the-shelf parametric software and relies solely on the off-the-shelf capabilities provided by the original software developers.

By enabling an external computer software other than the parametric modelling software to manipulate the development of parametric schemas; the designer shifts the attention from the development of schemas to the development of domain specific automated workflows.

The main definition of this strategy is the use of an external application or applications with the purpose to exert control over the parametric modelling application, with the goal of modifying existing parametric models, thus creating new parametric models or a hybrid approach. In this hybrid approach new models are created through the control which the external application exerts over manually developed parametric schemas.

The use of this strategy has many advantages and disadvantages. First I will expose the advantages and then the disadvantages.

**ADVANTAGES OF EXTERNAL CODE STRATEGY**

Many of the issues encountered through the process of parametric modelling are not issues with the design problem definition or the design problem structuring, but rather with the limitations of the parametric modelling software in representing complex design problems. Through the use of this strategy, it becomes clear how to separate issues with the design problem structuring from problems with
the limitations of the parametric modelling software.

By using this strategy, domain specific design problem solving can be dealt with using a custom external application written specifically for this purpose. This domain specific higher-order external application can be written in a higher-order computer programming language such as Javascript, VisualBasic or Python.

This strategy enables a design practice to manage appropriately how a designer should respond to a certain design problem. For instance, there are design problems which require an algorithmic response, such as the panelling of a free-form surface; while other design problems require a more intimate and manual resolution of the detailing through a direct modelling approach. By using this hybrid approach designers can combine and manage the trade-off between algorithmic-driven responses or tacit-driven responses.

Equally, by separating both the domain specificity towards a higher-order automated workflow and the geometry specific aspects to the parametric modelling software, it becomes easier to understand the domain specific aspects of a particular design. The separation of computer code from the complex spatial geometrical issues of the design also enable to manage changes in the domain specific aspects of the design. For logic and semantic code errors, designers can use the traditional text-based code debugging facilities provided by most programming Integrated Development Environments (IDE), but for spatial problems designers can use the direct modelling 3D interfaces provided by most 3D modelling softwares. This strategy enables the designer to take advantage of the tacit knowledge most designers have accumulated in their design experience. Designers are therefore enabled to use their tacit knowledge in solving complex spatial assembly puzzles using the 3D viewports of standard off-the-shelf parametric
software. For procedural and logical design problems, for example issues with the “pre-geometric” and “post-geometric” aspects of design, designers can use the algorithmic capabilities of the external automated workflow.

Because of this hybrid approach, where the user combines manually constructed parametric models — built with the end-user’s design tacit knowledge— with a scripting overlay written as external controller, designers can generate complex parametric models that are not possible otherwise in relatively short sprints of time. A single author model through this approach can be generated in only a matter of days for a complex project with challenging geometries. If using only the keyboard/mouse interface of off-the-shelf parametric models, the same project could have required a much longer time. Also, there is the possibility of the design problem not being able to be represented through the use of a pure-parametric model. For example, problems which rely on iteration or dynamic procedures, for example the use of Dynamic Relaxation for minimising stress and torsion on steel gridshells or the use of sphere-packing for decomposing a doubly curved surface into flat regions. These are problems not easily mapped by using a parametric modeller as-is off-the-shelf. For these advanced design problems, designers need to develop a higher-order control over the design of parametric models which enable them to represent complex design problems.

LIMITATIONS OF EXTERNAL CODE STRATEGY

While there are many advantages to the use of this strategy, it is also important to understand the consequences of its adoption.

Using this hybrid approach of using a standard-off-shelf parametric modeller with an external automated workflow is not the most elegant and efficient method
of taking advantage of the computational resources available within a computer. Standard off-the-shelf software rely on large file sizes. They use a large amount of computer memory and it is not written as optimised softwares for executing at fast speeds on personal computers (high performance computing). But while standard software is sub-optimal in speed, memory use and the file sizes it uses; the cost of improving any of these features in the software is much larger than the benefits. For example: while it might take 6 hours for an automated workflow to complete the panelling of a free-form surface as a parametric design process, the time, cost and effort required to optimise the code for an improved time might be significantly larger than simply using the suboptimal code as-is (taking into consideration labour cost and opportunity cost).

Equally because the external application is written as a separate application, many considerations emerge about how this external application is deployed to other users within an organisation. For example the tool might require documentation for explaining to the other users how to use the software to meet its goal, and during this process there is much room for errors. Ideally the same designer who designed the external application is its only user, but increasingly as design practices rely on larger team structures for more complex projects, designers need to share tools between the members of a team or even across another team within the same organisation for providing organisation-wide knowledge re-use.

When the user of the tool is not the same designer who developed the application, and the user finds an error with the application, it becomes difficult for the user to solve the error. When this strategy is implemented in design practice, it is preferable that the design practice has developed distinct roles for a design modelling specialist with scripting capabilities and a generalist designer
who uses the tools. Even in this situation there are major issues which could emerge from the design modelling specialist not being able to communicate to the designer the intentions of the software and issues which could emerge from the designer not being able to communicate his design intentions to the design modelling specialist.

And lastly, the execution times of running these external automated workflows with the standard parametric modelling software may be many hours. In the examples which will follow, some of these processes took from 30 minutes, 6 hours, to 24 hours of execution time. Because of these long running times, it is important to segment the automated workflow. By segmenting the automated workflow, the same workflow can be executed on multiple computers at once and take advantage of the speed gains from the parallelisation of work. An example is a process executed for dressing-up the 10,000 line elements and point elements representing respectively the beams and nodes of a gridshell. If the workflow is developed with a segmentation capability in mind, and the designer has ten available computers, he can execute the workflow as only 1,000 beams and 1,000 nodes per machine. If all computers begin the workflow at the same time, they will all complete the work at the same time. Regardless of how powerful the ten machines are, together they will always outperform a more powerful single computer running the same workflow on all the 10,000 beams.

In the following sections, four case-studies will illustrate the strategy exposed in this section. Each of the four distinct complex geometry case-studies has been taken from my participation on each of these projects. The case-studies will only deal with the aspects of the projects for which I was directly responsible, which for the majority of them, comprised of the development of complex scripting
overlays for developing complex parametric models for design modelling.
3.3.2 YAS ISLAND:

EXAMPLE ONE OF EXTERNAL CODE STRATEGY


YAS ISLAND PROJECT DESCRIPTION

The Yas Island Marina Racetrack Hotel (YAS) designed by Asymptote Architects, New York commenced construction in 2007 and was completed in November 2009 in time for the opening of the Formula 1 Abu Dhabi Grand Prix [de Leon and Shelden, 2012].

In December 2007, Gehry Technologies Inc (New York) was contracted as the integrated delivery consultant for assisting with the delivery of the complex gridshell design, together with Schlaich Bergermann und Partner (SBP) and Waagner-Biro (WB) as the grid-shell engineers and Front Inc. as the façade consultant.

I will document in this case-study the development of the Yas Island Marina Hotel steel grid Shell and panel assembly parametric models. The external code strategy outlined in this section enabled the production of an increased resolu-
tion model, with levels of complexity, not possible through the keyboard/mouse interface. The steel gridshell and panel assembly Digital-Mockups (DMU) of the YAS were developed through the use of the “Parametric model with external code” strategy.

The following account will expose how the “Parametric model with external code” strategy enabled the development of the parametric model of this complex and challenging project in a relatively short time, by a small team of individuals.

Digital Project™ models enable the centralisation of project data. This centralisation permits the integration of multiple concurrent fabrication specifications. This model enables the design team to coordinate multiple conflicting constraints in the manufacturing of complex 3D assemblies.

Digital-Mockups (DMU) are models with an extreme level of detail, as compared with wireframe models (models with low levels of detail). The extreme level of detail required in the development of a DMU makes it almost impossible for designers to develop these models without the use of an automation strategy.

This section will describe the parametric model development process for one component within the YAS project rainscreen gridshell. The parametric model was developed through the use of the “Parametric model with external code” strategy described in the preceding section. The model was developed using Digital Project™ — a CATIA™ V5 PLM solution for the architecture industry— as the base parametric modelling software.

The parametric model of the YAS was developed as a hybrid combination of hand built parametric models combined with external automated workflows for manipulating Digital Project™. The hand-built areas of the model are outside the scope of this exposition. It is sufficient to say that these hand-built models
represented the master design surface and the low level wireframe curves and points used to represent the centreline of gridshell beams and nodes. These wireframe models did not represent solid geometry. The wireframe model required hand-tuning and optimisation. This process was almost craft-based and required a close collaboration with the architect and with the fabrication team.

The external automated workflow, automated the production of the massive product structures, managing the complexity of developing parametric schemas as well as handling the folder structure of the model. The external automation workflow linked Digital Project™ with Excel™. Most of the data exchange between Gehry Technologies Inc (GT) and SBP was submitted and received in the form of Excel™ spreadsheets, representing large quantities of data. For example, each of the ten thousand beams and nodes were managed in Excel™ with attributes such as beam type (primary, secondary, midbeam, ringbeam) as well as coordinate data for each beam and node, as well as cardinality (top of steel, centre of steel, bottom of steel). From these spreadsheets, it was possible to interrogate the model; for example, to know the beam types around a particular node. With this facility to interrogate the gridshell database, represented in spreadsheets in Excel for non-geometrical data and as Digital Project models for geometric data, it was straightforward to develop a rule-based algorithm. This algorithm was able to interrogate the Excel spreadsheet to build new parametric model information. This rule-based algorithm was hard coded into the external automated workflow. The external automated workflow was able to either mod-

---

4Excel™ is easily automated through its component object model (COM) interface. From this interface, other applications can request information located inside worksheets and cell ranges within Excel™ [Roman, 2002].

101
ify existing parametric models or create new models based on existing models.

**WIREFRAMES**

With wireframe models, designers can communicate a limited set of design intentions without having to invest upfront during the early design stages a high effort in the development of the parametric models. While low in geometric information content, these models communicate enough information about the design intent without requiring the design team to commit upfront too much time in its development. Because wire-frame information is not sufficient in communicating the “assemblability” of the design, wireframe information cannot be used further downstream during the delivery of the project to conduct tests such as clash-detection, assemblability, swept volume analysis or any other test which relies on a high level of detail model.

Wireframe models provide designers with a visual method to inspect digitally whether design elements are dimensioned correctly and whether the parts relate to one another appropriately as intended by the designers.

In the Yas Island, the use of wireframe information enabled the validation of the structure by exchanging linear element information with the structural consultants. This information was effortlessly translated into simple data schedules enabling the gridshell structural consultant Schlaich Bergermann and Partner(SBP) to analyse the construction of the gridshell by using a neutral representation of the information embedded in the wireframe model. The use of wireframes and of data schedules that represent the wireframes enabled a platform-agnostic method of transferring geometric information across different tools and disciplines.
Figure 3.8: An Elevation drawing of the Yas Island Digital Mock-up. The three regions rendered above are zoom levels of the same drawing. As the Zoomlevel increases, we can see the high levels of detail and resolution at which the Yas Island Digital Mock-up represents the design intent.
Wireframe information enables a convenient and cost-effective method of managing large model assemblies with high component counts.

Wireframe models were convenient for storing placeholder data for the Yas Island Facade (see Figure 3.9). For example, the facade panels, the steel gridshell beams and the steel gridshell nodes were represented as wireframe data. These wireframe models represented the design intent as surfaces, lines and points respectively.

Figure 3.9: Wireframe geometry of the YAS Island Marina project. This image shows the node points, node normals and beam elements. Different beam types are indicated by the difference in colour.

The placeholder wireframes were effectively used as a coordination device between all stakeholders, allowing the coordination of quantities through a bill of material. The wireframe enabled us to coordinate the lengths of elements, alignment vectors for beams, alignment vectors for node normals, the node centre of steel coordinates, the node’s top and bottom of steel coordinates and lastly through the use of parametric modelling technology the geometric dependency
between elements was used to manage the topology of the Gridshell [de Leon and Shelden, 2012].

The topology of the panel was maintained as a list of the indices of the four beams required to enclose a panel. The topology of the beam was maintained as a list of the indices of the two points required to limit the length of an infinite line. Lastly the nodes of the grid Shell were maintained as a list of the index and XYZ coordinates of the points from which the beam centre line hangs (centre of steel).

Figure 3.10: The detailed bottom-up assemblies (top) were automatically instantiated dressing-up the wireframes (bottom). The assembly on the right is a sample of the result of executing the code on 12 panels.

When developing parametric models for projects with an unconventional level of geometrical complexity, wireframe model are enough for coordinating the
fabrication intentions during the early design stages, requiring a minimal up-front time and effort investment. However, as the project evolves and the design structuring becomes more well-defined, parametric models tend to increase in resolution, this resolution is commensurate to the availability of newfound information, emerging from the design process itself as well as from the stakeholders involved in both the design and fabrication process. While for standard projects designers can develop low-level of detail wire-frames, on unconventional projects designers are required to develop models with a much higher level of detail during earlier design phases. As the risks associated with building projects with unconventional geometries are higher than projects which use conventional geometries, the use of a DMU facilitates the process of pre-building and thus evaluating digitally the design and construction process prior to committing to the construction of a project.

As has been pointed out by [Shelden, 2002] the use of unconventional geometry in design projects has increased the cost of producing relevant construction documentation information. Wireframes allow trade-offs between the cost of producing information and the minimum amount of data necessary to describe design intent to other members of the construction team. Early in the development of the parametric model of the Yas Island, it became evident, that in order to validate the Gridshell a DMU model was necessary. However, the high level of complexity in this project coupled with its unconventional geometry presented a challenge in the development of this model. This DMU model was necessary to analyse "assemblability". It did this, by modelling the design intent models at a level of resolution which replicated the full-scale prototypes built on the site. This was achieved by modelling the components following the shop drawings
provided by Waagner Biro (the gridshell fabricator). From the pre-validation of the fabrication model, it was possible to avoid potential clashes, misfits or other dimensional control problems which could have occurred on-site during the construction of the project. From the use of a DMU model, design errors can be highlighted and corrected early, avoiding unplanned work and rework during the construction of the project.

Figure 3.11: The wireframe shown left was built from mounting curves and points on the rationalised surfaces (top left), the instantiated beams and nodes dressing up the wireframe (right).

The DMU assembly of the Yas Island had a bill of material count, which far exceeded 200,000 parts. These parts were required to assemble the steel gridshell rain-screen facade. To instantiate this high number of components, the DMU instantiation of the Yas Island was managed through an unsupervised VisualBasic script controlling Digital Project™ and Excel™. The external automated workflow tool takes the existing wireframe models together with the Excel™ spreadsheets to develop an extreme level of detail model. The automated workflow depends on the Excel™ spreadsheets for automating the instan-
tiation of high-level of detail components. The high-level of detail components are generated as re-usable knowledgeware from CATIA™, built by hand (see Figure 3.12 for a non exhaustive display of reusable components). These complex bottom-up parts were built by hand, due to their 3D spatial complexity. Automating complex spatial assemblies is difficult. Instead of automating the development of the complex components, a hybrid approach was developed. In this hybrid approach, the complex spatial components are built by hand using CATIA’s knowledgeware system, enabling the development of re-usable parametric objects embodying complex geometrical rules. These complex CATIA re-usable components are then instantiated by the external automated workflow dressing-up the wireframe geometry.

To represent the design intent of the model as materialised in the full-scale mock-ups, it was necessary to build more than twenty smart assemblies into the DMU parametric model. The external automated tool developed for the instantiation of the DMU re-used and re-arranged the pool of smart assemblies into more than 200,000 unique spatial configurations. This massive level of customisation was possible through a rule-checking algorithm. This algorithm is a generation zero cellular automata. The algorithm needed to query information from the Excel spreadsheet about any node, beam and panel in the gridshell. With the information obtained, the algorithm was able to perform various lookups of different key-value pair attributes associated with the nodes in the parametric model. For example, by gathering information of the neighbour nodes around a particular node, the algorithm was able to compute which kind of smart assembly it needed to instantiate in the parametric model. After the smart feature was instantiated, the algorithm extracted information out of the smart feature and
Figure 3.12: A non exhaustive catalogue of the various re-usable parametric models of the YAS ISLAND gridshell.
stored it in the Excel spreadsheet to be used by other model components. This three-tier level of control: Wireframes, Data schedules and bottom-up highly detailed smart features, provided the necessary tools to automate the production of an extreme level of detail DMU model of the Yas Island Gridshell.

INDEXING AND NAMING CONVENTION

Because gridshells are for the most part composed of a large number of elements, when modelling gridshells parametrically and programmatically designers must take into consideration a search strategy. While Search and Search algorithms are categorised below the umbrella of computer science, the field of search is broad, and its applications have an impact upon many more fields beyond computer science. A search strategy, enables to develop a heuristic to search a finite search space computationally while taking the shortest span of time to acquire the desired information. Regardless of how relatively fast the speed of computation seems to the end-user today; modern day computers are extremely slow at computing exponential problems and combinatorial problems. The search of information on a gridshell with more than 200,000 data records could become a problem if developers use naive algorithms for the exploration of the search-space. When performing tasks that depend on the lookup of data, if the techniques used for searching are inefficient it will render the task unreasonably slow. In order to develop faster search strategies, developers must understand the computational complexity of algorithms. By understanding the computational complexity of algorithms, developers can develop a deeper understanding of the relationship between the size of the datasets they manipulate and the algorithms themselves.
The speed of searching an element within a massive dataset is proportional to the number of computational cycles required to find the element. When developing a search strategy, it is useful to analyse two extreme scenarios which could occur during the search. The first scenario is the best case scenario. The second scenario is the worst case scenario. A best case scenario is the case where the search algorithm takes the least amount of time to find the record desired, ideally requiring a finite number of steps regardless of the size of the dataset. The worst case scenario is when the algorithm needs to traverse the entirety of the search space to find the record desired.\(^5\)

\(^5\)For example, in a dataset comprised of 200,000 data records and where the records have associations to other records, it would require 400,000 computational cycles in the worst case scenario to find a single record and one of its associations. This means the computer would have to iterate 80 billion times to locate all the records while simultaneously computing only one of the associations of each record. Assuming the computer takes one nanosecond, one microsecond or one millisecond in order to compute a single step in the algorithm, this process could last 80 seconds, 22.2 hours or 925.925 days respectively. For this reason, naive search strategies must be avoided by implementing search strategies that minimise exposure to potential worse case scenarios. The Big-O time complexity of most data structures are well known. The Big-O notation measures how algorithms respond to the size of the input data they work with. Hash-Tables or Hash-Maps have a computational complexity of O(1) during search, insertion and deletion. A computational complexity of O(1) means it takes a fixed and finite amount of time to perform the computation.
By maintaining the data of the grid shell in hash-maps\textsuperscript{6}, the search speed is constant, an improvement over storing the data in a conventional array, the speed of retrieving data from a hash-map is independent of the number of elements in it. Because of the use of a Hash-Map the search speed of a node and its nearby neighbours takes a fixed and finite amount of time. This key decision enables the use of massive datasets in reasonable computational time lengths. In the Yas Island, the use of Hash-maps facilitated the process of handling the complexity of a gridshell with more than 200,000 components. From the use of this data-structure it was possible to store key-value pair attributes from the Excel\textsuperscript{TM} spreadsheet or from the CATIA\textsuperscript{TM} 3d model in-memory, allowing the algorithm to compute fast look-ups and with it, enable the creation of new model information.

Besides the use of a Hash-Table, a naming convention system was also implemented for the naming of the nodes, beams and panels. This naming convention used an indexing sub-system from which the spatial and topological position of the element could be determined simply from decomposing the name of the element. This name or index was determined by encoding the topology of the model into the name of the element. The gridlines of the gridshell are the base of the naming system. Nodes are named by combining the names of the two

\textsuperscript{6}In computer programming, a hash map or hash table is a data structure which can map keys to values. The hash map contains an internal hash function which it uses to compute an index in which to store a value on a conventional array. The location in the array is called a bucket. The same hash function used to store a value, is used to retrieve the correct value from a bucket. Hash maps can store and retrieve data without iterating over all the buckets in an array, due to the hashing mechanism. A hash map takes a constant time for both insertion time and retrieval time of either small or large arrays.
gridlines which intersect to produce the node. For example a node indexed as H100_V350, is located at the intersection between gridline H100 and gridline V350. The gridlines are diagonal lines which are inclined either forward or backwards. Forward inclined gridlines are indexed as V lines and backward inclined gridlines are indexed as H lines. A beam indexed as H100_ V349_V350, is oriented facing gridline H100 and is trimmed between gridlines V349 and V350. A panel indexed as H100_H101_ V349_V350, is a rectangular region of the master design surface located between gridline H100, H101, V349 and V350. Because all components of the gridshell depend on the gridlines which subdivide the master design surface, it was straightforward to make the gridline names the base of the naming system for all components of the gridshell DMU.

Figure 3.13: Indexing and Nomenclature system for the YAS Island Marina. The system indicates how panels, beams and nodes all share a common naming system which indicates where within the gridshell the component is located.

**COMPUTATION**

Equal to the problem of hashing described in the preceding section, managing the total execution runtime of the tool was paramount to its success at producing a model within a reasonable timeline. The model had to be produced within a timeline that enabled the team to use it for decision-making. Computation time
was a recurrent issue during the development process of the DMU, due to the large number of elements in the gridshell and the high-level of complexity of the individual parts. During the Yas Island, we ran the tool on a Dell™ Precision Workstation with quad core Xeon processors and 30 gigabytes of Ram, even on a machine with these high specs (for that time), the tool would have taken weeks to complete its execution.

Inspired from the work of Google’s "map-reduce" [Dean and Ghemawat, 2008], I implemented a similar process, albeit in a manual mode. In "map-reduce" a very large time-consuming task is divided into smaller manageable tasks. The smaller tasks are executed each on a separate computer in parallel. After all smaller tasks complete, a final process collects the output from the smaller tasks and aggregates the result into a final output. During the Yas Island, we executed the tool on ten low-end laptops simultaneously. Each laptop ran the tool, producing only the components of a limited region of the gridshell. Because the laptops were connected to the same network, and had access to the project wide folder, the CATIA parts generated by each laptop were stored within the project wide folder. In this fashion, the first laptop executed its code on beams 1 to 200 and then the second laptop executed its code on beams 150 to 350, etc in groups of 200 beams with a 50 beam overlap. By overlapping the execution segments, redundancy was achieved, protecting the execution from possible failures which could result in the execution of the code. Once all models were generated by the ten laptops, a final process was required to collect the model components and integrate them into CATIA products. This final process was done by hand, without requiring extra automation.

With this "map-reduce" strategy the ten gigabyte parametric model of the Yas
Island steel gridshell was instantiated in a matter of hours. Without the use of a "map-reduce" strategy for the computation of the steel gridshell DMU, it would have taken days to perform the computation of the DMU on a single computer. Because of the fast computation gained from implementing a "map-reduce" strategy, the DMU was used extensively during crucial moments of the decision making process of the Yas Island. This DMU model enabled the design team to identify assemblability misfits and potential design errors which could have crept during construction. The DMU prevents rework by avoiding the unplanned work which results from design errors passing undetected to the construction phases.

The software development strategy of the instantiation tool had to be coordinated effortlessly with the modelling strategy of the bottom-up smart features. The integration of these two separate activities would enable the flexibility required to tackle the production of such a massive model assembly. After completing the rationalisation of the master design surface, the team was divided into two groups. The first group was in charge of creating the hand-modelled bottom-up smart assemblies described by Waagner Biro and Front Inc within the shop drawings they provided to the design team. The second group worked on the development of the computer code required to instantiate each of the parametric assemblies developed by the first group. The development efforts of both groups had to overlap, enabling through the parallelization of work, the speed-up of the process of developing the model. To solve dependency problems which could have taken place, both teams had to agree on the interfaces of the model and the interfaces of the computer code before both groups commenced their work.

The interfaces which the two groups agreed upon, would not have to change
during the implementation phases of the modelling efforts or the computer coding efforts. This loose-coupled relationship between programming and the modelling of components, enabled flexibility in the production of the DMU model. For example, as long as inputs and outputs of the programming and modelling were kept in agreement with one another, the development process was able to produce DMU models which enabled the design team to catch-up to design changes as they accrued.

The software tool which instantiates the bottom-up assemblies, would loop through the wireframe data and search for extra attributes in the Excel spreadsheets. Based on the data extracted from Excel and the wireframe data in CATIA, the tool made the decision of which smart feature had to be instantiated on top of the wireframe data. These rules were hard coded into the propagation software as conditions managed by a cellular automata global function. In a Cellular automata, decisions are made by analysing a context, in this case by analysing the neighbourhood of a particular node in the gridshell. Depending on the type of nodes in the neighbourhood and their attributes the tool would make a decision on which bottom-up assembly to instantiate. After the bottom-up assembly is instantiated, the component is configured and associated with other model components by the tool.

By analogy, the grid of cells in a cellular automata process, were represented as the grid of nodes in the Yas Island gridshell. The cellular automata process made possible the unsupervised propagation of more than 200,000 components within the DMU model of the Yas Island gridshell.
SUMMARY OF YAS ISLAND

Wireframe data is convenient for managing models with high component counts. A wireframe model is a low level of detail representation of a physical entity through basic geometric primitives. Because of the low level of detail of wireframe models, they are not appropriate for the investigation of “assemblability” issues and misfits. A new model had to be built over the wireframe model, with a level of detail equivalent to the full-scale physical mock-up for conducting clash-detection, reachability analysis, and swept volume analysis [de Leon and Shelden, 2012].

The unsupervised automation workflow of the Yas Island Marina Hotel, was made possible by a novel application of the cellular automata concept. The high performance searching strategy allows fast lookups over massive datasets. These datasets control the nodes, beams and panels of the Yas Island Marina Hotel. The speedup in the lookup of data was possible from the implementation of an indexing heuristic. This indexing heuristic embeds spatial information as well as topological information in the naming field of a component. The embedding of spatial and topological information in the naming allows faster storage and retrieval of data associated with a node, for example, the node’s neighbourhood data can be retrieved by executing simple string manipulations over the node name field [de Leon and Shelden, 2012].
3.3.3 MUSEO SOUMAYA:
EXAMPLE TWO OF EXTERNAL CODE STRATEGY

This section is related to the following publications:


MUSEO SOUMAYA PROJECT DESCRIPTION

Fernando Romero Enterprise (FR-EE) contracted Gehry Technologies Inc (GT) to rationalise the exterior façade of the Museo Soumaya (MUS) on June 2009, GT was contracted to wrap the façade with a seamless uniform hexagonal pattern over the frozen master design surface [de Leon, 2012, Peña de Leon, 2012, Romero and Ramos, 2013]. The primary structure was being erected in parallel to this panelling exercise and due to the procurement of materials and other dependencies of the subcomponents to the master design surface, the design of the envelope had been frozen and set as an abstract surface offset 1.0 meter away from the edge of all slabs on the project.

The desired aesthetic qualities in the outcome of the facade construction defined a finite set of rules which guided the rationalisation process.
Figure 3.14: A photograph of the completed Museo Soumaya as seen from street level.

Figure 3.15: Comparison of Museo Soumaya Digital Mock-up with real-word Mock-up
DESIGN PROBLEM

The selection of the Geometrica Freedome [Castano, 1999a, 2001a,b] space frame construction system, facilitated many aspects of the rationalisation. This construction system acted as the panel positioning device and the support structure for the panels. Because of the interior elements of the project being erected concurrently to the rationalisation process, the rationalisation strategy inherited a fixed non-negotiable master design surface (MDS). Therefore, the Master Design Surface had to be frozen [Peña de Leon, 2012].

The selection of Geometrica’s space frame strut and node system streamlined the rationalisation process substantially. Since triangular meshes are the duals of hexagonal meshes, it is possible to use triangular space frame configurations to support hexagonal panel configurations [Peña de Leon, 2012]. The centre of gravity of each hexagonal panel corresponds to each node in the Geometrica space frame structure (see Figure 3.16), this secondary structural system provides support for suspending the outer aluminium panels and the inner waterproofing panels [Castano, 1999b].

The rationalisation used a geometric strategy involving the use of a parametric sphere-packing algorithm for mapping hexagonal planar panels on to a doubly curved surface while preserving the isometries of lengths and the conformality of angles.

The work of [Schiftner et al., 2009] demonstrates how the problem of mapping a freeform surface with hexagonal tiles of equal shape and dimension can be achieved by using a mathematical numerical optimisation process. In this process, a triangular mesh is optimised by constraining the in-circles of the triangular mesh to conform to constraints which optimise the packing of circles and
spheres on the freeform surface. The vertices of the mesh are optimised to move on the surface in response to an objective function which restricts the relaxation process to only those configurations which allow in-circles of similar radius to form by the triangles of the mesh. The mesh conforms to the surface through the minimisation of the objective function, which in turn allows the mesh to meet the hexagonal properties desired [Schiftner et al., 2009].

The approach taken in the Museo Soumaya provides a similar result without the use of an iterative numerical method. The approach used a parametric sphere-packing process to compute the hexagonal panelling. This approach is easily mapped through a parametric modeller namely Digital Project™. The sphere-packing system was implemented by utilising a geometrical strategy. This strategy emerges from the use of a geometric heuristic inspired from the research of
[Thompson, 1961] on the geometries of cellular aggregate structures. Tissues and cellular aggregate structures as indicated by Thompson, inscribe hexagonal patterns from the intersection of circles and spheres [Thompson, 1961]. In his work, [Thompson, 1961] relates the similarities between the morphological growth of cellular aggregate structures and the boundary conditions of intersecting spheres [Peña de Leon, 2012].

In "On Growth and Form" [Thompson, 1961] identifies a "geometrical strategy" for the formation of the emerging hexagonal structural patterns which are found abundantly throughout the natural world. By following this geometrical strategy, analytical sphere surfaces can be intersected with the free-form surface of the façade. From these intersections, we obtain a quasicircular \(^7\) embedding in the uv-mapping \(^8\) space of the surface, the self-intersections of the quasicircle embeddings are the vertex nodes of our hexagonal packing. A gap had to be incorporated in the panelling to hide the discrepancies which emerge from the local variations in the panel geometry and because of the use of a low number of unique panels. These discrepancies emerge from the resulting hexagonal panels not being exactly hexagons with interior angles of 60° and with all six sides having equivalent dimensions [Peña de Leon, 2012].

\(^7\)since the surface of the façade is not a plane nor a sphere the resulting intersection curve resembles a circle but is not a circle.

\(^8\)The letters “U” and “V” indicate the axes of the two-dimensional euclidean space \(\mathbb{R}^2\), these letters are the equivalent of the XYZ coordinates in three-dimensional space, in two-dimensional space, the letters UVW denote the three axes of the coordinate, in this case the dimension W denoting depth is void.
SPHERE-PACKING

While in the case of the Museo Soumaya, the sphere-packing system was used to produce the hexagonal panels in the façade. The sphere-packing algorithm can be used in a variety of other design situations where the designer does not know upfront the surface class of a design construction. The use of the k-means clustering algorithm can be used to conveniently reduce a large number of unique components by classifying their differences and grouping them accordingly. How precisely the clustering algorithm works depends on the relevance of the data per instance provided to the algorithm [Peña de Leon, 2012].

Contemporary architectural practices have developed sophisticated methods for the representation of complex surface shaping. These surfaces exhibit curvature on both the u and v directions of its parametrisation. This double curvature, in turn challenges the rationalisation of the structure due to the production limitations of available materials and methods of construction. Therefore, a trade-off must be met between surface curvature continuity and its discretisation into fabrication building components. This trade-off becomes evident when the construction is taking place in a limited resource setting where the project is constrained by the availability of technology [Peña de Leon, 2012].

The pattern had to maintain three opposing dimensional constraints. Firstly, all panels had to be equally shaped ideal hexagons. Secondly, all panels had to have 63.0 cm diameter. Lastly, all panels had to keep a uniform gap of 3.0 cm against any of its adjacent neighbours.

The first attempt to solve the problem was to fold a two-dimensional pattern of equally sized hexagons on to the UV parametric space of the surface. This solution would be plausible if the master design surface had been negotiable.
Because the parameter curves of a NURBS (Non-Uniform Rational B-spline) surface deform according to the parameterisation of the surface. Using the UV space of the surface to guide the two dimensional construction of the geometry of the façade provides little control over the relative uniformity that can be obtained from this method [Piegl and Tiller, 1997]. Technically reparameterising the parameter space of the surface while simultaneously preserving the shape geometry control is a possible solution to the mapping problem [Piegl and Tiller, 1997, 241]. However, this solution would require extensive mathematical optimisations of the knot spacing which would be difficult to map using a parametric modeller due to the limitations of parametric modellers in mapping iterative computational processes [Peña de Leon, 2012].

Because the parameter space of a surface is not analogous to the metric space of a surface, mapping two-dimensional curves onto the UV parameter space of a surface does not enable the user to have control over the exact metric dimensions of the hexagons or the metric dimensions of the gap between panel to panel. The project specifications required an explicit definition of the hexagonal diameter dimension deduced from the local dimensions of the diameter of the panel plus the desired gap.

The second solution attempted, was to map the surface using the Gaussian curvature of the surface as a splitting operator. By splitting the surface over the fault lines produced between areas of different surface curvature continuity, this operator can sort surface regions by degrees of curvature. The operator enables to distinguish areas of the surface which are flat from areas of the surface which are curved. A new problem emerged from the use of the Gaussian curvature operator. As a result of using this operator, the master design surface becomes seg-
mented into regions of curvature ranges. It is difficult to manage the blending of the panels from one region into another. Since the original design goal required the panels to wrap continuously throughout the façade without a seam or visible discontinuity; the Gaussian curvature operator solution was abandoned in favour of a solution which could accommodate the required aesthetic. For the reasons discussed above, the Gaussian curvature operator solution was not successful at satisficing the design requirements [Peña de Leon, 2012].

The third and last solution used a geometrical heuristic which worked in two-dimensions. Intersecting circles produced the best hexagonal distribution in two dimensional space (see Figure 3.17, the diameter of the circles control the diameter of the hexagons produced; while the development of this pattern in two dimensions is trivial, developing it in three dimensions is non-trivial.

![Figure 3.17: Diagram indicating the relation between the diameter of spheres, the gap and the hexagons inscribed.](image)

Inspired from research on how cellular bodies grow in nature, mathematical & theoretical biologist Darcy Thompson, examined how the morphology of cellular aggregates in the natural world closely resemble the geometrical patterns
which emerge from the intersection of spheres. The intersection of spheres and the hexagonal patterns which emerge from these intersections, have been identified by Thompson, as structures similar to the patterns which emerge from the formation of tissues and cellular aggregates in biological systems. In chapter 4 of “On Growth and Form”, biologist [Thompson, 1961] reconstructs hexagonal patterns emerging in natural forms from intersecting spheres, finding stable conditions as they are attracted towards each other until they cannot get compacted further. There exist sphere-to-sphere configurations with stable and unstable properties. When 4 intersecting spheres meet at a point the configuration is unstable if the same spheres are arranged in a different configuration, such that the same four spheres are configured so that only three spheres meet at a point, then the configuration could be considered stable [Peña de Leon, 2012]. As Thompson points out “the four cells do not meet in a common centre, but each cell is in contact with two others, a so-called polar furrow, the visible edge of a vertical partition-wall, (joins or separates) the 2 triple contacts and so gives rise to a diamond shaped figure, identified more than a hundred years ago by Rusconi in a salamander and called by him a tetracitula” [Thompson, 1961]. In other words, when two circles intersect they produce two points, one north and the other south relative to a left and right sphere [Peña de Leon, 2012]. If we draw a sphere on each of those two points, we should arrive to the polar furrow configuration (see Figure 3.18).

By following Darcy Thompson’s description of the polar furrow, we can construct any hexagon as a local property of the intersection of a doubly curved surface with two spheres and then intersect the two resultant curved circles supported on the surface and subsequently get the north and south point. If we
hang a sphere only on the south point and repeat the process of intersecting two spheres obtaining two cardinal points on each iteration, we can obtain a hexagonal point distribution network over the surface with as many rows and columns as needed (see Figure 3.19) for the algorithmic process that creates the mesh, (see Figure 3.19) for the result of the algorithm [Peña de Leon, 2012].

From the circle mesh grid (see Figure 3.21 for circle mesh before hexagons are inscribed), the hexagonal pattern is extracted by hanging the inscribed hexagons from a circle offset half of the desired gap from the original circle mesh grid [Peña de Leon, 2012]. This process is executed by an external software tool as seen in Figure 3.20.

Although the mesh applied to the MUS surface using this technique was close to the desired result, the hexagonal mesh contracted vertically on the regions of high curvature, and as a result the lower edge of the hexagonal pattern rose up and retracted relative to the bottom edge of the master design surface. This distortion to the pattern is due to the reduction in the height of the panels which result from the sphere-packing algorithm reacting to areas of high-curvature. In
Figure 3.19: Sphere-Packing algorithm for developing the circle mesh grid from which to inscribe hexagons.
order to correct the visual impact of this effect, a post-production process was executed over the resultant mesh to correct the height reduction (see Figure 3.22 for the before stretch (top) and after stretch (bottom)). This stretching process takes a squashed hexagon and corrects its height based on the fixed proportions of an ideal hexagon’s width and height dimensions. The filtering process which extracts the panels with abnormal squashing is shown in Figure 3.22, in the same figure a line indicates the result of stretching the pattern to meet the façade surface edge. To extend the pattern, the panel heights must be corrected [Peña de Leon, 2012].

The process of correcting the heights of the panels required me first to identify all the panels that require correction, and second to stretch them by using rail
Figure 3.21: An example of a circle-mesh grid executed on the lower half of the Museo Soumaya Façade.

curves mounted on the design surface. The rails extend past the boundary curves which define the upper and lower edges of the design surface. These curves were used as rails to scroll the surface downwards until the vertical dimension of all hexagons corresponded to the horizontal dimension of the hexagons. The height to width proportion of an ideal hexagon can be used to regularise the vertical dimension of a squashed hexagon. The first step to regularise the squashed panels is to identify the panels which require stretching, this process is shown in Figure 3.23. Panels which exhibit abnormal proportions are filtered and labelled as custom panels while the panels which exhibit acceptable proportions are labelled as standard (see Figure 3.24). From the custom panels, a series of

130
Figure 3.22: Hexagon mesh before stretch (top) Hexagon mesh after stretch (bottom).

rail curves are extracted for the stretching of the panels. Using these rail curves, the panels are stretched vertically across the surface in order to fix the height squashing problem which results from the areas of high Gaussian curvature.

The sphere packing algorithm is sensitive to initial conditions. This means the circle mesh grid (see Figure 3.21 and the hexagonal mesh which results from sphere packing vary substantially depending on the initial conditions which initialize the algorithm. Through various trial and error experiments the design team was able to identify the best initial conditions for running the sphere-packing algorithm. These initial conditions produce the best aesthetics results. Through these experiments, we found that the hexagonal pattern is most regular
Figure 3.23: Diagram indicating the filtering process of standard versus custom panels and how the custom panels are clustered into families by measuring the panel-to-panel differences.

Figure 3.24: Hexagonal mesh stretching process. Indicating how rails are first mounted on the squashed panels (left) and how the panels are vertically stretched while mounted on the rails, leaving the width of the panel unchanged. This vertical stretching transforms the squashed hexagons into uniform hexagons (right).

when growing the pattern horizontally in the middle of the envelope and starting from the flattest point of the surface. For these reasons, the location of the initial sphere in the packing controls the aesthetic qualities of the resultant mesh. The search of the flattest point on the surface was executed as an optimisation process using the CATIA knowledgeware optimiser. This optimisation process used a circle passing through three points located respectively on the top, mid-
dle and bottom waterlines\(^9\) of the master design surface (these three points and three waterlines as well as the circle which passes through them, can be seen on Figure 3.25). The optimal starting location would be the circle with the greatest radius, hence by finding the circle with the greatest radius we can find the point on the surface with the least curvature. This would yield the position within the middle curve with the least curvature as shown in Figure 3.25. Figure 3.25 also shows the optimisation process to determine the ideal radius of the spheres in the first row of the sphere packing system [Peña de Leon, 2012].

**CLUSTERING**

Due to cost and logistics, the 16,000 unique panels of the MUS project had to be reduced to only a few unique panel templates. These templates would then be repeated approximating the originals. The first exercises started by searching for family distributions of 7 and then 24 unique templates, eventually this number would rise to 1000’s of family templates. The design team wanted the family distributions to begin from the horizontal waist of the facade and to end at the upper and lower edges of the facade. From this initial belt like family, all other families derive, in a similar fashion to the way ripples in a pond reflect the shape of the stone which initiates the ripples [Peña de Leon, 2012].

The Museo Soumaya initially contained more than 16,000 unique panels. The high panel count increases complexity and cost of manufacturing. The high number of unique panel shapes would require a large number of panel moulds. Typically the cost of producing moulds is high. The high throughput that can

\(^9\)A curve produced by intersecting a plane parallel to the XY plane with the Master Design Surface.
Figure 3.25: Two optimisation routines executed on the Museo Soumaya. First, to identify the origin of the sphere-packing (top) by finding the flattest point on the waist of the facade. Second, to determine the best sphere radius for the sphere-packing (bottom).
be generated from a single mould offsets significantly the high cost of producing the mould. This is typically achieved through obtaining economies of scale. Economies of scale are obtained due to the panel manufacturing cost decreasing as a function of how many times the mould gets reused throughout the manufacturing process. However, while economies of scale can be easily obtained from projects which rely on mass-production. For projects which rely on mass-customisation, the use of moulds does not provide the same cost savings. Therefore, when dealing with mass-customisation, the reduction of the number of moulds for a project is a desirable manufacturing outcome. To reduce the number of unique instances of geometrical elements, a classification algorithm can assist the designer in determining the trade-off between the number of unique elements and the resultant aesthetic which results from this selection.

One of the most user friendly algorithms for data clustering and classification is the "Lloyd" k-means clustering algorithm. This clustering analysis algorithm is available in most statistical packages, such as the statistical language R [Peña de Leon, 2012]. With the invocation of six lines, we can sort all 16,000 panels into 49 clusters. This code also exports the cluster data and the centroids of each cluster:

```
A statistical k-means clustering analysis over the panel population using 21 parameters as attributes and weighing the area of the panel as the key attribute yielded the result wanted. Since the panel area increases as panels tend to move
```

away from the waist of the facade; the family distribution which result from this analysis appears to cluster panels by following a striation effect which is the natural consequence of the panels increasing their size directly proportional to their distance to the waist of the facade (see Figure 3.27).

Figure 3.27: 21 family k-means clustering result. This clustering uses the area as part of the calculation and thus produces a banding effect.

The inclusion of the panel area parameter in the calculation of the clustering analysis is crucial to achieving this banding effect over the distribution of panels. The following code snippet demonstrates how the clustering was implemented:

```r
Mydata<-read.table("C:/paneldata.csv", Header=TRUE, sep="", row.names="UniqueID")

fit <- kmeans(mydata, 49,100,1,"Lloyd")
write.table(fit$cluster, "C:/panel_cluster.csv", sep="",)
write.table(fit$centers, "C:/panel_centers.csv", sep="",)
```

Figure 3.26: Example of “Lloyd” K-means clustering code for the statistical programming language R
the families. If the panel area parameter is omitted in the calculation, a more patchwork distribution is achieved [Peña de Leon, 2012].

This patchwork distribution is more faithful to the original geometry (see Figure 3.28), but it increases the logistics of fabrication since there would be more fragmentation over the distribution requiring heavy use of schedules for the location of each panel [Peña de Leon, 2012].

Figure 3.28: 49 family k-means clustering result. This clustering does not use the area as part of the calculation and thus produces a more patch-like result. This result is more faithful to the original pattern in comparison to other family distributions.

While a patchwork distribution is more faithful to the original design intent model. A banding effect is produced when using the area of the panel in the clustering calculation. This enables the assembly team to easily locate panels within family boundaries visually. This distribution is convenient for projects which rely on unskilled labour by making the task of assembling straightforward. For example, by using visual cues for the placement of panels rather than relying on data schedules [Peña de Leon, 2012].
The k-means clustering process is outsourced to Excel™ using a statistical plug-in for the k-means clustering function. The data required for each panel was extracted from the wireframe data in the CATIA model. From the panel model, 21 parameters are used to measure intrinsic properties of the panel that help the clustering algorithm determine the uniqueness or similarities between all the panels [Peña de Leon, 2012].

While this analysis produces a clustering of the panel data, the information must be visualised in order to gain meaning. The visualisation of this data also allows one to see how it would be possible to construct a façade with 16,000 panels using only 49 moulds. The visualisation is created by replacing each panel with the centroid of the family in which the panel was clustered as shown in both Figure 3.27 and Figure 3.28. A greater number of families reduces the panel-to-panel gap discrepancies and approximates more faithfully the master design surface [Peña de Leon, 2012].
3.3.4 FABPOD:

EXAMPLE THREE OF EXTERNAL CODE STRATEGY


FABPOD PROJECT DESCRIPTION

(a) Digital Mockup  (c) Photograph by John Gollings

(b) Digital Mockup  (d) Photograph by John Gollings
This section will address the development of the parametric models used for the fabrication of the FABPOD project. The fabrication intent models of the FABPOD were developed in CATIA™ using the “Parametric Models with External Code” strategy. The models were developed as a way to manage the geometry control and tolerance modelling of the detailed fabrication planning data for the FABPOD project.

The research that was developed during the FABPOD project extends the work developed for the Responsive Acoustic Surfaces (RAS) cluster in SmartGeometry 2011 [Burry et al., 2012b]. This work extends the work developed for the RAS by developing a further understanding of how to manufacture hyperbolic geometries and their complex intersection patterns. This research was possible from the application of Computer Aided Design (CAD) technology with Computer Aided manufacturing (CAM) technology [de Leon et al., 2013].

The Flexible Automated Design for Production Workflow (FADPW) system developed for the materialisations of the FABPOD fabrication intent enabled the flexible orchestration of change management in the earlier stages of design and in the stages of fabrication planning (see Figure 3.31) [de Leon et al., 2013].

The FADPW deployed a novel use of a declarative simple geometry codex as the *lingua franca* for the exchange of information. This open exchange provided the design team with a decentralised collaborative environment [de Leon et al., 2013].

During the FABPOD, the automation workflow became the informational backbone of the “conceptual framework” enabling a “quilt-like” process of fabrication and assembly [de Leon et al., 2013].

The separation of the feature design from the design-logic automation was
paramount in enabling the flexibility goals stated earlier.

**AUTOMATION WORKFLOW**

In the FABPOD project, Rhinoceros\textsuperscript{11} with Grasshopper\textsuperscript{12} were used as a design authoring tools. The wall and cell assemblies would be declared and specified according to a highly flexible set of design principles and functional

\textsuperscript{11}Rhinoceros (Rhino) is a stand-alone, commercial NURBS-based 3-D modeling software, developed by Robert McNeel & Associates.

\textsuperscript{12}Grasshopper\textsuperscript{TM} is a visual programming language developed by David Rutten at Robert McNeel & Associates. Grasshopper runs within the Rhinoceros 3D CAD application.
Figure 3.31: The FABPOD automation workflow. This diagram shows how wireframe definitions from Grasshopper™ flow to Digital Project™. In Digital Project™ these wireframe definitions increase in resolution and are made ready for machining as solid models.

requirements for room acoustics [de Leon et al., 2013].

The models developed in Grasshopper were not Digital-Mockups (DMU). The models only contained enough information for configuring basic dimensions and the topology of the design. To fabricate the FABPOD, it would be necessary to build a DMU, a more robust detailed model. This model would have to represent the NC (Numerical Control) features required to operate the NC machinery. The model was built within Digital Project™ using user-defined machine features that represented the level of detail commensurate to the fabrication planning required [de Leon et al., 2013].

The complexity of the model required that this model would have to be built using an automation strategy. The model was built as a hybrid model. The
model used both hand-built features and computer code to represent the design. The hand-built features represented the components of the fabrication cells, and thus describing the model assembly. The computer code, built using VisualBasic .net as an external application, took care of the development of the complex parametric model schema in Digital Project™ [de Leon et al., 2013].

To transfer the data from Grasshopper to Digital Project™, we had to develop a custom file format as a CSV (comma separated values) file. The file described each wall assembly and each fabrication cell. The following section will describe the file transfer protocol [de Leon et al., 2013].

To read the custom file-format, I had to develop a custom file reader as part of the automation workflow. This reader would read the file and from the information described in it, the reader would have to configure the high-level of detail models in Digital Project™ [de Leon et al., 2013].

The team members working in Grasshopper developed a custom file-format exporter that would export to a file all the information required. For example a fabrication cell was described only as an attribute indicating the Identification name of the fabrication cell, the front and back material types, the cell centre coordinates, The cell hyperboloid centre axis, and the list of planes which trim the hyperboloid to form a Voronoi cell. This information was recreated in Digital Project as a product model assembly [de Leon et al., 2013].

The Automation workflow, first built the simple geometry data transferred in the file into the model. After this basic wireframe data was in the model, it was possible to instantiate the high-level of detail features. These features can be instantiated either manually or through an automated process. Due to the number of features required, manually instantiating the model would take too
long for the model to be effective. The automated instantiation of the complete
FABPOD model in Digital Project™ took 4 hours  [de Leon et al., 2013].

This Flexible Automation Workflow system enabled the regeneration of fabrication-
geometry control models reconfigured according to strict yet robust fabrication
conformance guidelines  [de Leon et al., 2013].

Figure 3.32: FABPOD custom user interface to Digital Project™, for the instan-
tiation of the product model assembly.

"The looseness of the Grasshopper™ automated workflow in allowing design
changes and the robustness in the Digital Project™ automated workflow, with its
ability to incorporate design changes while strictly conforming to the fabrication rules and manufacturing guidelines, highlights the great potential of automated workflows in enabling the integration of flexibility in design while maintaining robustness at the fabrication level” [de Leon et al., 2013].

**FILE TRANSFER PROTOCOL**

"The simple file format developed for the interoperability between the two automation workflows, required the necessary conditions indicated by [Kvan, 2000] for the production of a loose-coupled collaborative process [de Leon et al., 2013]. Furthermore Kvan points to the “exclusive” collaboration type noted by [Maher et al., 1996], they define it as a process where by each participant “work[s] on separate parts of the problem, negotiating occasionally by asking advice from the other” [Maher et al., 1996]"

![Image of a custom CSV file format developed to transfer information from Grasshopper™ to Digital Project™](image)

Figure 3.33: An example of the custom CSV file format developed to transfer information from Grasshopper™ to Digital Project™. Each line in the file represents a cell, each wall was represented as a single text file.

From the use of a simple file format, the design team was able to enhance collaboration. This simple file format provided the team with a process with which to transfer information from one software into another. The use of a text file with a simple structure as the central exchange protocol, enabled the team to exchange complex design models via email messages. The use of this simple
exchange protocol made possible the coordination between the design team and the fabrication team [de Leon et al., 2013].

The simple file format described above enables to transfer simple mathematical constructs such as points, lines, planes [de Leon et al., 2013]. These constructs are valid regardless of the software they are built in.

"These constructs could then be transferred using our simple text-based file format from Grasshopper™ (where the acts of designing were being made) into Digital Project™ (where the acts of detailed fabrication planning where being made) with the assumption that the mathematically defined objects in Grasshopper™ would have the same stable characteristics in Digital Project™” [de Leon et al., 2013].

"These constructs would be transferred out of Grasshopper™ through automated routines, and the exported constructs would be transferred into digital project through an automated routine as well, as such automation was seen as the key integration device throughout the project in contrast to relying on standardised file formats or manual intermediation” [de Leon et al., 2013].

**ROBUST USER FEATURE**

According to Karam and Kleismit “Building flexibility into a model is, for example, critical in facilitating downstream engineering changes” [Karam and Kleismit, 2004] further in facilitating downstream changes designers must address the “ability to anticipate the requirements of potential change [and acknowledge that this ability] is built on experience, training, and a natural intuition on how the design may have to be modified.” [Karam and Kleismit, 2004]

Following the downstream considerations identified by Karam and Kleismit,
the FABPOD “Fabrication Cell” features were developed in isolation, ahead of the automation system that would subsequently glue them together into a seamless assembly. The rationale for taking such a counterintuitive decision, stems from the idea, that, in the event that the fully automated system would fail, there would still be the ability to instantiate the components manually, albeit with a considerable time penalty [de Leon et al., 2013].

TECTONICS

THE TIMBER FRAMES

The assembly requirements of the "timber framing system challenged the use of a traditional parametric modelling strategy. Although parametric modelling systems such as Digital Project™ enable designers to map design intent as flexible models, these associative modelling paradigms present a great deal of difficulty in coping with topological variations” [de Leon et al., 2013]. For this reason in the FABPOD project, we implemented the “hybrid” approach as illustrated in Figure 3.1. In contrast to other methods, the “hybrid” approach enabled the flexibility required in varying the total number of framing components per fabrication cell [de Leon et al., 2013]. The number of cell sides in the frame data was variable, the numbers fluctuated based on the design data arriving from the Grasshopper™ Voronoi distributions over spherical surfaces [de Leon et al., 2013]. While parametric modellers do not generally facilitate topological variations without computer code in their off-the-shelf modalities, the “hybrid” "strategy is a best of both worlds solution, enabling the end-users to articulate the smart features and the scripters to define the topological variation inherent in most algorithmic design-logic processes. With the separation
of the tectonic issues from the logic issues using feature-based components, it was easy to manage the geometrical complexity required without this complexity getting entangled with the complexity of the design logic” [de Leon et al., 2013].

Figure 3.34: Assembly product for Wall003. This figure shows the assembly complexity of a single wall of the FABPOD. In this image one of the fabrication cells is highlighted.
Figure 3.35: FABPOD fabrication. This figure shows the complexity of a single fabrication unit, and how the timber frames must be mitred and bolt holes inserted, as well as scoop holes removed from the timber on each fabrication cell to facilitate hand assembly.
Figure 3.36: Timber frame geometry. This figure indicates the complexity of the timber frame geometry. The timber frame is trimmed by many cutting features, including the hyperboloid front facing panel, the two adjacent timber frames, and the fabrication cell back panel. The green surface shown above is a linear swept surface approximating the hyperboloid for machining purposes. This surface is also at an offset distance from the hyperboloid panel to accommodate for fitting tolerances during assembly.
THE CENTER PLATE

The dependency of the centre plate\textsuperscript{13} “stiffener on the variable number of timber frames and scoop holes respectively, highlighted the necessity of having a flexible strategy for orchestrating this kind of topological variation. The dependencies on the timber frame topology affected the number of scoop holes\textsuperscript{14}. The scoop holes "facilitated human hands in gaining access to bolts during the fastening process [de Leon et al., 2013]. Lastly, the centre plate feature was trimmed by the hyperboloid front face and the adjacent timber frames at a variable angle non-orthogonal to the plane normal to the centre plate, this would imply a 5-axis machining process for manufacturing, through end-user-defined “Machining features” simple geometrical representations translated seamlessly into 5 axis machining data” [de Leon et al., 2013].

UNFOLDING

The last Component of the FADPW system was the translation of the 3D assemblies into unfolded nested components for easy NC data extraction (see Figure 3.37 for unfolded data of the FABPOD cells). Using Digital Project’s\textsuperscript{\textsuperscript{TM}} axis-to-axis transformations and coupled with computer programming logic, the process was automated effectively and all the unfolded data was transferred into the last process converting the unfolded control-geometry into Automatically

\textsuperscript{13}The center plate component supports the cell from the inside. Its principal role was to provide additional mass to the cell for low frequency sound absorption.

\textsuperscript{14}The scoop holes, are holes which need to be carved from the center plate, in order to allow for a human hand to reach to the inner sides of the timber frames. The scoop holes also allow to lighten the load of the cell as less material is required. For every side in the Voronoi cell, there must be a scoop hole. There is a one-to-one relation between the number of timber frame sides and the number of scoop holes in a cell.
Programmed Tool (APT) data. "This step was developed outside of Digital Project™ in Rhinoscript™, as an alternative to the use of Dassault DELMIA™ for the production of tool paths” [de Leon et al., 2013]. Both DELMIA™ and Rhinoscript™ versions were developed [de Leon et al., 2013].

Figure 3.37: Unfolded fabrication cells of the FABPOD project. The automated workflow of the FABPOD project simultaneously creates unfolded data as it creates the 3D assembly units.
3.3.5 GLORY FACADE HELICAL STAIR: EXAMPLE FOUR OF EXTERNAL CODE STRATEGY

GLORY FACADE HELICAL STAIR PROJECT DESCRIPTION

Figure 3.38: Elevation screenshot of the Sagrada Família Basilica Glory Façade Helical Spiral Staircase in context.

This section will document the parametric model development process for the pre-design of the Glory Façade Helical Spiral Staircase (GFHSS)\(^\text{15}\) of Antoni Gaudí Sagrada Família Basilica.

The pre-design exercise comprised the development of a 3D parametric model of the staircase. The model had to represent the building code constraints while respecting the current construction layout of the hyperbolic stair shaft. The tool had to provide an interface in Excel to enable the team to configure the number of stair steps per landing and the number of landings in the whole stair. The tool

\(^{15}\)The work presented here is Work In Progress, the project is currently ongoing
had to be flexible enough to accommodate the configurations made in Excel with the 3D modelling constraints modelled in CATIA.

**DESIGN PROBLEM**

At first inspection, the problem of mapping a traditional linear stairway onto an spheroid of revolution such as the GFHSS may seem trivial, but after consideration of a preliminary set of nuances in implementing such a design, the designer is presented with a handful of issues. When mapping an ellipsoid to a flat surface, it is not possible to map a curved surface onto a flat map without some degree of distortion.

The systematic representation of a round surface on to a plane is called Map Projection [Snyder, 1997]. Although there exist infinite possible mappings that satisfy the condition, only a few have been published and applied. [Snyder, 1997]

While Map Projection is an important aspect of solving the developability of the stair system around the ellipsoidal surface, a brief survey of the literature suggest that most methods for computing the forward translation of topographic coordinates onto the ellipsoidal coordinate system require iterative methods for accurate results; for example, Vincenty’s formulae\(^\text{16}\) or some form of approxi-

\[16\text{Vincenty’s formulae are two related iterative methods used in geodesy to calculate the distance between two points on the surface of a spheroid, developed by Thaddeus Vincenty in 1975 They are based on the assumption that the figure of the Earth is an oblate spheroid. See [Vincenty and Bowring, 1978] for the details of the formulas.}

The Direct Problem: Given an initial 2D coordinate on the ellipsoid and a distance S along a geodesic line of the spheroid, Vincenty’s formulae can be used to compute the end point.

The Indirect Problem: Given two 2D coordinates, Vincenty’s formulae can be used to compute the geodesic distance s between the two coordinates.
mation with a low margin of error. The most important mathematical notions and considerations are the known behaviours of flattening curved surfaces and the metrics for measuring distortions before and after the flattening transformation [Desbrun et al., 2002]. When a mapping preserves lengths it is said to be isometric and when the mapping preserves angles it is said to be conformal [Desbrun et al., 2002]. Hence we can minimise the distortion of one at the expense of gaining more distortion in the other. Only when flattening a developable surface or a plane, can we achieve a pattern whose metric has both isometric and conformal preservation.

An accurate stair mapping is a mapping where the stair preserves the angle of bearing\(^\text{17}\) (conformal) as it wraps the ellipsoidal surface creating a rhumb line or loxodrome around the ellipsoidal surface. Since a loxodromic ellipsoidal spiral preserves the angle of bearing, it guarantees conformality thus providing the stair flights with a constant slope (constant riser to tread ratio). If the steps are positioned on this rhumbline, the steps would have a constant angle of bearing as they rise and taper along the ellipsoidal stair shaft. In order to achieve the mapping I had to geometrically project our two dimensional stair system onto a virtual cylinder placed tangent to the ellipsoidal surface at the start of every stair flight (see Figure 3.39). From this virtual cylinder the stair had to be projected towards the outer and inner ellipsoid in order to generate the step geometry.

In the following section I will provide an account of the process of mapping

---

Both problems require iterative functions which need to converge on the correct answer. \(^\text{17}\)the direction or position of something, or the direction of movement, relative to a fixed point or line. It is usually measured in degrees, typically with magnetic north as zero. [Oxford Dictionary]
the loxodromic ellipsoidal spiral stair system as a solid model using the parametric modelling environment Digital Project™ through an innovative hybrid strategy, where automation routines are used to compile and edit larger complex parametric assemblies that leverage disparate hand-built piece-wise flexible components.

The development of a flexible tool for configuring the design of the stair system was clear, given the pre-rational agenda. The role of the automation routine was to join all the diverse problem solving strategies into a coherent whole.

**FLEXIBLE COMPONENTS**

The process of automating the stair propagation was twofold. The first aspect required hand modelled assemblies prior to the coding exercise, they contained
geometrical and logical functions necessary for the conformality requirements of the stair as well as building code compliance. The second aspect of the automation was the code itself, a minimal set of instructions required to instantiate and configure the parametric components in situ.

While mapping a rhumb line onto an arbitrary surface is a complex mathematical operation, in a solid modeller these operations are trivial even when persisting associativity between components. For this reason I modelled the rhumb line components as hand-modelled reusable components. These components could then be instantiated either manually or through automation. The automation strategy followed the spirit of quilting assemblies where parts are designed off-site and then assembled together only in the final stages.

**RHUMBLINE**

As outlined above, the rhumb lines need to preserve slope, even after conforming to the geometry of an ellipsoid. While mathematically computing the geodesics of an arbitrary surface is non-trivial, many parametric solid modellers provide in their object factories a line operator, which is a parametric primitive which provides the functionality required. In the case of Digital Project™ these functions are provided by the Line operator. Creating a rhumb line on a cylinder only required the creation of a line supported on a surface at an angle from a latitude line and specifying a length from an origin input point.

The rhumb line would serve as a device for controlling the slope of a single stair flight as well as the step positioning framework, from the rhumb line we distributed points evenly, a step would then be instantiated between every two points in the series.
Figure 3.40: Rhumbline wireframe system on the left, indicating how the rhumbline preserves the angle of bearing as it wraps along the ellipsoid shaft. On the right the step geometry is mounted directly on top of the rhumbline, guiding the steps around the ellipsoid shaft.

The stair step translates the points generated on the rhumb line cylinder, shifting the geometry from a cylinder towards an ellipsoid surface.

**THE STAIR STEP**

While the rhumb lines were relatively easy to map parametrically onto the virtual cylinder, the step geometry requirements presented more complications. The development of a reusable step component, was carried out as a “trial and error” exercise, as can be observed in Figure 3.41 the construction process of the step assembly is not as intuitive to map as the rhumb lines. Complications were due to both the geometry and the strict building code compliance requirements.

The minimal tread depth, width and the inner and outer surface of the shaft shaped the step geometry. The step was created as a solid model of a typical spiral stair trimmed against two ellipsoidal surfaces, the inner and outer surfaces of the shaft respectively. The step was the most complex component to achieve computationally because we needed the outer and inner surface to be a patch of an ellipsoid. The underside surface is a patch of a helicoid surface. except for the inner, outer and underside surfaces of the step, all other surfaces in the step
Figure 3.41: Geometry of the step for Sagrada Família Basilica Glory Façade Helical Spiral Staircase. The step is a solid model, trimmed between the two hyperboloids of the stairwell. The bottom side of the step is a helicoid surface.

gallery of models are planar.

The desired properties in the stairway step determined all of the required constraints for the stair system. The modelling strategy for the entire stairway starts from this primary component. The step must respect certain non-negotiable properties such as the riser height and the geometry it needs to adhere to.

INSTANTIATION AND CONFIGURATION

The design of the flexible tool was implemented as an external VisualBasic.net application. The application took control of both Digital Project™ and Excel™. Parametric modelling within PLM (Product Life-Cycle Management) software packages like Digital Project™ allows designers to make distinctions between modelling (the creation of geometry and associations) and assembling (the in-
integration and matching of parts). While it is relatively easy to instantiate the components manually, the sheer number of parts and their elaborate connections could potentially confuse even the most advanced of users. Instantiation of large assemblies is also a time-onerous labour intensive task.

The flexible tool had to provide the design team with a process of integrating and configuring the isolated smart assemblies such as the rhumb lines, the step and landing solid component. In Figure 3.44 we can see the VisualBasic code and in Figure 3.42 we can see the VisualBasic GUI used to instantiate the components and configuring them as specified in an Excel stair schedule seen in Figure 3.43. With the tool in place users can model different versions of the rhumb lines or different versions of the step, without needing to change the tool, provided they followed the protocols the code requires to instantiate the geometry.

The hybrid approach of combining user-features hand-built with computer algorithms which instantiate them, afforded the users opportunities for intervention throughout the various workflows embedded in the tool. For example, users could interrupt any running process and manually adjust any component within the assembly. This high-level of customisation was required for tweaking and configuring the model after the tool had generated a stair variant. Although it was rarely necessary, the tool also allowed the user to configure the stair components during the process of instantiation without having to interrupt the running process. While the Excel spreadsheet of the project controlled most of the stair parameter values, occasionally values had to be overwritten directly in the component tree of the CATIA model generated by the tool.

Computer algorithms enable us to represent procedural knowledge within
parametric models, however, not all aspects of a design problem need to be mapped using algorithms. When models need to represent design problems using both algorithms and manual parametric model associations, the hybrid approach enables us to represent both the algorithmic and non-algorithmic aspects of the design problem in a single model.
Figure 3.42: Sagrada Familia Basilica Helical Spiral Staircase User Interface. This figure indicates the script overlay develop for the instantiation of the stair system.
Figure 3.43: Sagrada Família Basilica Helical Spiral Staircase Excel Configuration File. This figure indicates the Excel configuration files together with the user interface to instantiate stair variants.
Figure 3.44: VisualBasic code for the propagation of the Sagrada Família Basilica Helical Spiral Staircase. This system takes the User-defined features created by hand, and instantiates them based on algorithmic logic and the configuration from Excel.

Figure 3.45: Sagrada Família Basilica Helical Spiral Staircase design variant. This figure shows a design variant instantiated from the scripting overlay tool.
3.3.6 SUMMARY OF EXTERNAL CODE STRATEGY

The case studies presented in this section illustrated the use of the “Parametric model with External Code” strategies. Here I will summarise the findings from the latter section. In the Yas Island Marina (YAS) project this strategy demonstrated its facility in developing large parametric models with a high level of detail.

In the Museo Soumaya (MUS) project the strategy demonstrated its ability to represent complex design problems such as the “sphere packing” method. During the MUS project the strategy also demonstrated its capacity in building complex computational workflows with softwares like Excel™ or statistical software for example R (Programming Language).

The FABPOD project demonstrated the facility of building complex custom interfaces to standard parametric modelling software and its speed at developing complex assemblies at fabrication grade in a relative short time.

The Helical Spiral Stair of the Sagrada Familia Basilica demonstrated the capacity of this strategy to facilitate links with external applications. For example, through Excel™ for developing more complex interfaces to complex design problems. In this project, the strategy also demonstrated the facility of generating mathematically complex design constructs by using the hybrid approach afforded by this strategy.

This strategy also demonstrated the value of decoupling geometrical complexity from design-logic complexity to facilitate a more visual process of interacting with tectonic issues and the need for topological variability in traditional parametric modelling paradigms. This proved to be one of the most valuable contributions of this strategy, as it enables practices to combine team members with or
without scripting background in the design process. The team members without a scripting background, can develop the smart parametric features (user-defined-features/power-copies\textsuperscript{18}) required in the project, while the team members with scripting knowledge develop the computer code which assembles the smart features into coherent assemblies. This means designers can use their visual intelligence to develop complex 3D component libraries or to resolve complex details visually. Resolving these details would be too difficult to make explicit in the form of an algorithm. In particular when a project has too many unique conditions, where each condition or detail needs a unique solution or approach.

\textsuperscript{18}I am borrowing here the specific terminology used for CATIA/Digital Project— the predominant software used throughout this thesis— to refer to feature objects created by the user. These are objects which use system primitive features to build higher-level abstractions of more complex objects which are specific to a particular discipline or project. For example, the Yas Island Nodes and Steel Beams, the Museo Soumaya Panels, as well as the components of the Museo Soumaya triodesic structure (Nodes, Beams, Purlins). Other softwares may refer to this capability of parametric design software differently, for example generative components refers to it as “user generated features”.
Chapter 4

BESPOKE APPROACH

This chapter will document software development strategies which enable the development of: 1) bespoke software prototypes that use standard software libraries, 2) bespoke software prototypes that create custom libraries specific to flexible modelling and 3) bespoke software prototypes that incorporate a custom scripting layer for programming flexible models. These strategies demonstrate the development of alternative user interfaces to the more traditional off-the-shelf parametric software.

There are times when off-the-shelf parametric design software cannot represent a design problem due to the restrictions of the acyclic graph on the kinds of problems it can solve. For example, problems which require bidirectional constraint modelling are difficult to represent using parametric modelling [Kilian, 2006]. When designers are looking for freedom from the restrictions of the parametric schema, bespoke design softwares provide freedom to represent the design problem using any kind of representation system which the programming process permits. Speed of computer code execution is another important factor
for determining the use of a bespoke software strategy versus an off-the-shelf parametric design software. For example, when implementing a process such as the “Dynamic Relaxation” solver, the number of cycles required by the relaxation solver would render the use of a parametric schema too slow for the computation to become useful. If the solver does not compute fast, the user will abandon the process in favour for a quicker solution. Due to their low computational overhead Bespoke design softwares can compute design problems at faster rates than their off-the-shelf parametric software counterparts.

The following sections will document three strategies for the development of bespoke software. They will each build on the other successively, describing both the advantages and limitations of their use.

4.1 STANDARD LIBRARY

4.1.1 OVERVIEW

This section will expose the development of flexible models through the development of bespoke softwares. Bespoke software refers to software developed from source code. It is software written specifically to address an intended functionality which standard softwares do not provide. This section will illustrate a bespoke software development process where component object libraries are used to accelerate the software development process. Component object libraries are software components pre-designed prior to the development of the bespoke software. Complex bespoke software applications for design modelling, can be developed by using preexisting software libraries which have been pre-designed to perform a complex task. These libraries abstract the complexities of specialised
modelling problems. Most 3D CAD software packages incorporate pre-designed software modules. These standard software libraries encapsulate knowledge in the development of 3D CAD software applications.

Software component libraries are one of the many available deployment strategies for releasing computer code. End-users for the most part, interact with a finished product; the compiled and ready-to use computer softwares which ship ready for use in their off-the-shelf state. These softwares arrive ready-to use out of the box, without substantial set-up and without the need to hire consultants to help in the installation of the software. The software for the most part can be used directly by the end-user from day one.

These off-the-shelf software packages are made from off-the-shelf software component libraries. The developers of most proprietary CAD (Computer Aided Design) softwares, do not need to develop all the technology required to build their software product offerings. Component library suppliers license their software to CAD software developers. CAD software developers use licensed software libraries from third-party companies with expertise in the development of software components for CAD software development. These third-party companies provide 3D geometry kernels and scene-graph visualisation kernels. For example, Parasolids™, a 3d kernel developed by Siemens PLM, is used in Autodesk Inventor™, Solidworks™, Microstation™, Vectorworks™, Abaqus™ and NX™(Unigraphics). Equally HOOPS Visualize™ from Tech Soft 3D, is an example of a scenegraph software component library, which enables visualisation of the geometry produced from the 3D kernels into specialised 3d viewports

---

for interaction with the end-user. Other scenegraph libraries include: vtk\(^3\) (The visualization Toolkit), coin3D, OpenSceneGraph\(^4\) and Redsdk\(^5\) (used in TopSolid).

These software components are available to anyone interested in developing CAD software. While the majority of these libraries depend on costly licensing and royalty fees, there are open-source alternatives. openCASCADE\(^6\) is an open-source 3D geometry kernel which performs the same operations as the proprietary 3D kernels: ACIS\(^7\), Parasolids\(^8\) and CGM\(^8\) (Convergence Geometric Modeller). While the features in openCASCADE\(^6\) are not as mature as the features in the proprietary systems, openCASCADE\(^6\) has enough features to develop a full-blown proof of concept prototype and can also be used for developing experimental design software to test bespoke CAD software research.

The use of these software components requires a level of expertise above the average knowledge of the typical end-user. The use of standard 3D library components requires an advanced-end-user developer; an end-user who has a higher order understanding of the software development process, above scripting and who understands how to program computers in a low-level programming language such as c++.

While off-the-shelf software is a finished polished product; APIs (Application Interfaces)

\(^3\)http://www.vtk.org/
\(^4\)http://www.openscenegraph.org/
\(^5\)http://www.redway3d.com/pages/redsdk.php
\(^6\)http://www.opencascade.org/
\(^7\)http://www.spatial.com/products/3d-acis-modeling
\(^8\)http://www.spatial.com/products/cgm
Programming Interfaces\textsuperscript{9}, SDKs (software development Kits)\textsuperscript{10}, Frameworks\textsuperscript{11}, Class Libraries and sourcecode are the raw materials of the advanced-end-user developer. The advanced-end-user developer, combines these raw materials to develop bespoke software applications addressing domain specific problems, which are not addressable through the use of the existing standard off-the-shelf software available in the market.

All 3D applications need to have a minimum of features, which are set by the standard of the industry in 3D software. The end-user is already expecting these features in a 3D software application regardless of whether it is an off-the-shelf application or a bespoke application. The rest of the application depends on the domain problem the application is addressing, and the user experience the developer creates for the end-user interacting with the application.

Depending on how important flexibility becomes to the developer, the developer may focus on developing an application with flexibility or robustness in mind. These two constraints oppose each other and must be managed as a trade-off. The software development methodology inherent in the use of standard of the industry in 3D software.

---

\textsuperscript{9}``API(Application Programming Interface) is a well-defined interface that provides a specific service to other pieces of software. An API is an interface designed for developers, in much the same way that a Graphical User Interface (GUI) is an interface designed for end users'' [Reddy, 2011]. API’s could be private or public.

\textsuperscript{10}``Software Development Kits: are typically a set of software development tools that allows for the creation of applications. They go beyond providing API’s, but may also include other software which aids in the development of a particular type of software. The QT Framework is an SDK, as it provides API’s, a programing development environment and its own compiler.

\textsuperscript{11}``A software framework is an abstraction in which application-specific code can be written with little user-written code. Frameworks include API’s and SDK’s, they are integrated in such a way that developers can start creating applications with minimal effort.
standard software libraries is non-trivial. The traditional software implementation process uses the following steps: programming-compiling-linking-debugging-refactoring-compiling-linking.

While the use of the standard library affects all the steps in the traditional software development process; I will focus only on the implications for using standard libraries in the programming step, where the design of software mostly takes place.

Besides enabling re-use of prepackaged knowledge, the standard library enables the abstraction of complex operations into simpler operations. A standard software library or framework is a complex package of ready to use features which the advanced-end-user developer combines to create complex bespoke 3D applications. For example, the openCASCADE™ software development framework is composed of many software libraries and object oriented software components at the disposal of the advanced-end-user developer. The task of integrating all these disparate components into a coherent 3D bespoke application is non-trivial.

Unless the developer has developed expertise in using a particular software framework or library, the use of a framework for rapid prototyping software applications might be slow and the learning curve steep.

When developing software, a useful design principle is to separate the computer software into distinct sections, were each section addresses a separate concern [Dijkstra, 1982]. This “separation of concerns” (SoC) principle enables the developer to abstract the use of a standard library into another library, providing a modular approach to developing software which depends on standard libraries.

When using standard libraries or frameworks without using the SoC principle,
the software development process becomes slow and the code developed from this process becomes unorganised and illegible.

The development of bespoke software applications with standard software libraries enables the advanced-end-user developer to re-use knowledge encapsulated in the software library. The reusable source code represents many years of debugging and usability from tried and tested exposure to a large user base. The software also represents the best practices across an entire industry.

**ADVANTAGES**

The time to develop a working software prototype starting from a blank slate is significantly reduced when using a prefabricated standard library to develop bespoke software. Most off-the-shelf parametric modelling software are based on commercial grade standard software libraries. Two of the major geometry cores of modern cad software are Parasolid™ and ACIS™ [Porter, 1995, Zhao and Wang, 2004]. Parasolid™ is developed by Siemens and ACIS™ by Spatial. Open-source alternatives are the OpenCASCADE™ technology by OpenCASCADE SAS. All these libraries enjoy a large user-base and are easy to use for implementing bespoke software applications with BREP (Boundary Representation) and NURBS (Non-uniform rational B-spline) requirements.

The use of standard libraries in the development of bespoke software enables the standardisation of the software prototyping practices [Lakos, 1996, Reddy, 2011]. It also reduces the uncertainty of the prototyping process, by focusing the programming attention only on the graphical user interface development and on the domain specific problem mapping. Since the standard geometry kernel library provides all of the geometry features, the development of bespoke soft-
ware then takes advantage of the many years of testing and quality control these libraries have undergone for the last 10 years.

Equally because of this tried and tested feature of standard libraries, the bespoke software benefits from the large community of users of these standard libraries and from the many years of feedback these libraries encapsulate.

LIMITATIONS

The use of the standard library improves the development of bespoke design applications by enabling the development team to inherit the knowledge embedded in these libraries. However, the use of the library in its “off-the-shelf” state requires the software developer to develop and re-invent the wheel for every design problem he faces. A far more effective strategy would be to develop a custom library which wraps the standard library, the wrapper would contain all the re-usable knowledge of how to use the standard library, removing the difficulty developers might face with learning how to use the library. The library wrapping technique will be discussed in detail on section section 4.2.

Working with a standard library as-is presupposes the developer has extensive knowledge of its use. At times, the developer only needs to use the standard library as a black box or Input-Process-Output (IPO) system. When using the standard library as-is it is difficult to use the library as a black box or IPO system, as some components do not provide default behaviour and might need configuration prior to instantiation. To solve this issue, the developer must create a wrapper library as discussed in section section 4.2. The wrapper takes care of configuring the components and instantiating them. This practice enables the developer to encapsulate knowledge of using a component object library into a
wrapper library.

Working with the standard library as-is, promotes unorganised code, leading to the code becoming “spaghetti code” this term refers to computer code which as Conway points out has “the same clean logical structure as a plate of spaghetti” [Conway et al., 1978]. Spaghetti code renders itself un-editable by the difficulty it presents to understand it. Because the code is not legible by other members of the development team and possibly by the same developer at a later time, developers often have to discard the code and start from a blank-slate. This inefficient practice becomes a form of erasure-technique similar to the issues faced with the development of parametric schemas in standard parametric modellers. The use of Software libraries without the SoC principle leads to spaghetti code, which in-turn could lead to programming rework. For example, if new information of the domain problem becomes available and a change in the architecture of the domain problem occurs, programmers need to jump curves in the refactoring of the computer code. These type of disruptive changes are not incremental, and are difficult to integrate, unless the software is developed as a modular system of interchangeable software components.

A major drawback of developing code with the standard library and without using a modular approach, is the difficulty this method presents when programming is done as part of a team. Because there is no standardisation of how to use the standard library, two developers might have different ways to address the same issue with the same standard library. There is the potential for two developers to be working on the same solution. The two developers could be working on the same solution without either of the two knowing they are each overwriting the others work. When this occurs, one of the two becomes redundant.
Due to the reasons mentioned above, it is difficult to decouple the “separation of concerns” from problems which arise in the use of the standard library and from problems which arise from the domain problem. Both the code addressing how to use the standard library and the code addressing the domain problem coexist in the same source code. Using the standard library as-is promotes a poor use of the “separation of concerns” principle.

As the description of the case studies within this strategy will demonstrate, there are limits with simply using the software libraries as-is. A greater effort will be required from the advanced-end-user-developer to implement flexibility in the development of tools to solve projects with higher levels of complexity.
4.1.2 THE DERMOID

PROJECT DESCRIPTION

This section will expose the Dermoid Project, a wood gridshell structure developed at the Royal Danish Academy of Fine Arts (RDAFA). The project focused on the material properties of wood beams and how these can be represented digitally in near real-time. This project was the result of an 18 month period of intensive workshops, as part of the Velux visiting Professor invitation of Prof. Mark Burry at the Centre for Information Technology and Architecture (CITA) in RDAFA [Burry et al., 2012a].

The workshops worked successively on each aspect of the construction of a final gridshell pavilion. The main aspects were: the conceptualisation of the design, the design development and the fabrication phases.

Both the researchers from CITA and SIAL (Spatial Information Architecture Laboratory) collaborated with the students at RDAFA to develop the design of the gridshell, using a combination of standard software technologies with bespoke software tools, developed in-situ for the project.

The development of the Dermoid concept, began with Prof. Burry, suggesting the use of the ellipsoid surface as the overall form for the pavilion. The ellipsoid surface, has interesting mathematical properties. For example, the uv space of an ellipsoid is represented as a rectangle in two dimensions. Interestingly, two of the four edges in the rectangle vanish and become the “poles” of the ellipsoid when translated to 3D. This makes the ellipsoid surface difficult to tile with two dimensional patterns. Direct conformal mapping translations cannot be performed on ellipsoids to develop patterns over its surface. The ellipsoid geometry
requires the simulation of physical properties to wrap patterns over its surface.

The second premise for the project was the idea of using a reciprocal frame truss as discussed by Prof. Burry in “Between Intuition and Process: Parametric Design and Rapid Prototyping” [burry2003_intuition]. Reciprocal frame trusses have traditionally been used to cover extensive surface areas with lamella like structures made of short span lengths. This enables designers to work with a limitation of small span lengths while covering large span lengths in gridshells, without the use of a node. The reciprocal frame truss is a beam-to-beam structure, instead of a beam-node-beam structure.

The combination of these two premises and the computation of the consequences of merging the two, set the scene for the development of the project.

The third component of the Dermoid project, was the use of physical experimentation as a complement to the digital tools developed, through the physical experimentation, many of the material properties of a specific wood variety could be computed digitally and approximated physically.

The digital tools used combined different standard software tools, such as Maya™, Digital Project™, Excel™, Rhinoceros™, Grasshopper™. And bespoke software tools developed using Processing and OpenCASCADE Technology™ (OCCT).

During the workshops at CITA, Prof. Popovič brought to our attention the limited existing knowledge in the mechanics of reciprocal frame trusses and their interesting properties [Larsen, 2008]. While it is easy to engineer them using heuristics, most engineers need to abstract the problem into plate elements and disregard the problem as a beam problem, as the loading conditions create a cyclic problem that makes it hard for simulation software to calculate the load.
paths.

The other issue we encountered had to do with the limitations of parametric software to enable designers in wrapping two-dimensional patterns of reciprocal frame trusses on an ellipsoidal surface.

The non-trivial aspects of the problem made the search for working solutions worthwhile.

**UV SPACE MAPPING**

With the design goal of wrapping two dimensional patterns forming reciprocal frame-trusses in 3D, I developed a series of experiments in 3D Studio Max™, for developing custom scripts to translate two-dimensional mesh designs, drawn manually in 3D Studio Max™, into a file format representing two-dimensional coordinates. These coordinates would subsequently be used to locate points on a surface as uv coordinates. Surfaces use two dimensional coordinates to evaluate parametric locations. Transferring a two dimensional pattern into the UV parameter space of a surface is a straightforward task. The coordinates of the source 2D pattern should be within the parameter bounds of the surface. All parametric surfaces, for example spheres and ellipsoid, have a parametric mapping between 0 and \(2\pi(6.2831)\). The surface parameter range can be converted to be bound between a number from 0.0 to 1.0 for convenience. By making sure the bounding box of the source 2D pattern fits within the parameter space of the surface, it is straightforward to map a two dimensional pattern parametrically over a parametric surface.

As shown in Figure 4.1, because of the mathematical and geometrical properties of the ellipsoid, the north and south region of the two dimensional parameter
space degenerates towards a point, producing a north and south pole. Any point on the north or south edge of the parameter space, will all share the same xyz coordinates. These properties are not desirable for fabrication. This first exercise of mapping a two dimensional pattern over the surface. When wrapping an ellipsoidal surface with a two dimensional pattern, a special treatment must be given to the translation of its poles or otherwise another method must be used for wrapping the two dimensional pattern other than using projection mapping techniques.

Figure 4.1: Reciprocal frame-truss based on pentagon distributions, the geometry was wrapped on the UV space of the surface. This figure indicates how the pattern degenerates near the poles of the surface.

In similar fashion to the MUS project, the projection mapping technique using UV surface parameter space was discarded. The Dermoid could not rely solely on the UV space parametrisation technique. While this technique was used to mount the two dimensional pattern on the surface, another technique was necessary for unwinding the poles and other areas where the pattern stretches, since an
ellipsoid does not have constant curvature. Complementing the UV space mapping with another technique, would make it possible to avoid the production of degenerative geometries, which would render the geometry with non-favourable fabrication properties.

**SPHERE PACKING**

The concerns identified with the UV space exercise, opened up the search to alternative methods for wrapping surfaces without relying on its UV space surface properties.

Similar to the MUS project, I investigated the possibility of using the sphere packing algorithm to distribute the pattern. This distribution is shown on Figure 4.2 the pattern provided a better solution to the issue at the poles compared to the results obtained from the UV space wrapping, but it also did not wrap the pattern as uniformly as expected.

![Figure 4.2: Example of sphere-packing the Dermoid surface. As the circles step away from the initial row, the circle-mesh deviates from a uniform distribution.](image)
The patterning routine was implemented using VisualBasic.Net as an external scripting overlay to Digital Project™. The VisualBasic.net bespoke software used Digital Project’s™ geometrical operations to split the surface of the ellipsoid with spheres, the intersection of the spheres with the ellipsoid surface produces circular curves lying on the surface of the ellipsoid, the intersection between all circular curves results in intersection points, on these intersection points the reciprocal frame system is developed.

During the development of the parametric models of the Dermoid project, extensive use of explicit modelling was necessary. The need to use explicit models during quick iterations of design, exposed the issues introduced by Maher and Burry [Maher and Burry, 2003]. They stated the need to develop explicit models first to understand the relationships between components, prior to committing to the development of a parametric schema. This level of resilience and grit, is still necessary in the development of complex projects. As parametric technologies currently stand, relying solely on the technology during conceptualisation, might produce an unwanted “creative bottleneck”.

The sphere packing algorithm enhanced the previous results obtained from the UV space wrapping of patterns. However, the design goals of wrapping a two dimensional grid as homogeneous as possible, were not achievable through the use of sphere packing.

The search for a strategy to wrap the pattern remained open, despite having attempted to resolve it using two available techniques.
DYNAMIC RELAXATION

A process which facilitates the resolution of the shrinking “poles” of spherical surfaces is the Dynamic Relaxation process. This process has been previously used by Chris Williams on the British Museum [Williams, 2001] and by Evolute for the YAS Island Marina [Eigensatz et al., 2010].

The Dynamic Relaxation process was implemented as well for the Dermoid project using a variety of technologies, while the last version of this process was implemented using Grasshopper™; During the conceptual stages of the design process, many variations of the system were developed in both proprietary systems as well as bespoke 3D application systems. This section will expose the bespoke 3D application system developed prior to creating the final system in Grasshopper™, however in this exposition I will only document the bespoke 3D application development as well as the initial design explorations of the system in 3D Studio Max™, the final Grasshopper™ version of the tool is outside the scope of this document.

There are two possible ways to resolve the Dynamic Relaxation method, with an analytical method or a brute-force method. The analytical method uses mathematics and the brute-force method uses an iterative geometrical heuristic. Chris Williams and Evolute used the mathematical approach for the British Museum

---

While the early experiments of the Dynamic Relaxation system was developed using 3D Studio Max™ and Qt C++ with openCASCADE™ technology, for reasons of facilitating the usage of the process, Grasshopper™ was selected as the final environment in which to conduct the investigation due to the fact Grasshopper™ was the common denominator software among the project participants and workshop champions. This would facilitate both the sharing of the models and the modifications of the models as the design intent evolved.
and the Yas Island respectively. The brute-force method has two possible implementations: the first implementation of the brute-force method relies on particlespring systems which use Hook’s law for attraction and Coulomb’s law for repulsion (See “Electrical and Gravitational Force Fields” [Tremblay, 2004, p222]). The second implementation of the brute-force method uses a particle-only system using a simplified spring heuristic which does not rely on Hook’s law nor Coulomb’s law. A spring is a mathematical model which describes the oscillatory pattern between the connection of two objects which exert an equal and opposite force to each other [Bourg, 2002, p64] [Tremblay, 2004, p245]. However, if visualising the oscillation is not as important as obtaining the result from the computation, the spring model can be simplified through a heuristic which extracts only the necessary aspects of the model which are needed to compute the relaxation. In the Dermoid we opted in using a force-directed method which ignores both Hook’s law for attraction or Coulomb’s law for repulsion, by instead determining the position of the node by centring it around its neighbours without a predefined set length between the node and its neighbours. These force-directed methods are typically used to draw two dimensional graphs with desirable aesthetic properties [Eades, 1984, Eades and Tamassia, 1989].

For the Dermoid project, it was more straightforward to implement the force-directed brute-force heuristic method described above. The geometric heuristic method requires a lower cognitive load than using the mathematical approach, however while the brute-force method is more intuitive it has a lower performance than using a more mathematical approach. Since performance at this stage was not necessarily an issue, using the intuitive method would enable to satisfice the design constraints while facilitating the development process.
The principle of dynamic relaxation follows a basic pseudo-algorithm (see Figure 4.3) that is easy to replicate as a script by any standard software. This method is also easy to develop as a bespoke application for executing the algorithm over both analytical surfaces and freeform surfaces.

```
Set tolerance = 3mm.
Set step = 0.01 (range between 0 and 1)
Set Energy = 1kilometer;
1-For each point in the two dimensional pattern:
   1a-Select a point
   1b-Find the neighborhood of the selected point.
   1c-Find the center of gravity(COG) of the neighborhood.
   1d-Project the COG towards the ellipsoid surface.
   1e-Calculate Energy = distance(selected point,COG)
   1f-Find vector between selected point and projected COG.
   1g-Move selected point, in direction of vector, step amount.
2-Repeat all instructions until Energy less than tolerance.
```

Figure 4.3: Pseudo algorithm for the dynamic relaxation of two-dimensional meshes on curved surfaces.

The pseudo-algorithm illustrated above, shows the simplicity of an iterative solver for computing the dynamic relaxation. Iterative solvers require an exit condition. This condition determines when the algorithm should end the iteration. In the case of the dynamic relaxation solver, the exit condition is the overall energy of the system. Once this energy is below an established tolerance, the algorithm can safely assume that it has satisfied the goals of the design problem. Dynamic Relaxation solvers converge quickly when the relaxation step is increased. The size of the Dynamic Relaxation step has a trade-off between convergence speed and relaxation accuracy. Playing with the relaxation step interactively enables an increase in convergence speed by increasing the step and an increase in accuracy, by reducing the step.
The development of a near real-time dynamic relaxation solver gives access and support to designers in decision making, at an interactive iteration rate.

The first working prototype for the geometrical heuristic indicated above, was developed as a script in 3D Studio Max™.

Having understood the conceptual basis for the Dynamic Relaxation solver. It was seamless to implement the algorithm using any programming language. For reasons of speed and rapid prototyping the software, 3D Studio Max™ was selected to implement the script. This selection was due to the 3D studio Max™ easy to use scripting language (MaxScript\(^{13}\)) and the facility this language has in interacting with the topology properties of polygonal meshes.

Implementing the dynamic relaxation solver as a MaxScript relied on the scripting facilities of meshes in 3D Studio Max™. The topological properties of meshes in 3D Studio Max™ expedited the implementation of the dynamic relaxation system.

Polygonal meshes are data structures for storing the properties of polyhedral meshes in 3D software. They contain lists of data which store the vertices and their coordinates, the indices of the edges of a mesh, the indices of the edges vertices, the polygonal faces of the mesh and the index of their surrounding vertices and edges. This means two-dimensional meshes in 3D Studio Max™ can be used to store the topology of the gridshell in two dimensions. These meshes represent the UV point coordinates of the gridshell nodes on the ellipsoidal surface.

\(^{13}\)Maxscript is a built-in scripting language which ships with 3D Studio Max out of the box. It enables to script the creation and the extraction of model data from 3D Studio Max objects [Bicalho and Feltman, 2000].
I developed an algorithm which would tag vertices with useful attributes. For example, the free boundary vertices, were tagged as being boundary vertices, all other vertices would sit on top of the ellipsoidal surface during relaxation. After tagging, executing the dynamic relaxation solver would interactively show the relaxation of the mesh in the viewport. This prototype script helped understand the behaviour of the solver and assisted in the drafting of the final specifications for the final solver. The last solver was developed as a robust C++ application. This application used both the openCASCADE™ Technology and the Qt™ framework.

The 3D Studio Max™ MaxScript Dynamic Relaxation solver, helped in sketching the behaviour of a Dynamic Relaxation solver. However, this solver was not ready to satisfice the constraints required in the Dermoid project. Two issues affecting the script required the development of a bespoke application. The first issue is with the use of BREP geometry. 3D Studio Max has limited Nurbs programming interfaces. It is not straightforward to implement BREP capabilities in 3D Studio Max™ using MaxScript. But while BREP capabilities are not present in 3D Studio Max™, it does have an advanced programing interface for interacting with polygonal meshes. The polygonal data structure of 3D Studio Max™ was required for designing the necessary two dimensional meshes. Once these two dimensional meshes are converted to a CSV (Comma Separated Values) file the Dynamic Relaxation process could be executed on another external application.

The second reason for developing the dynamic relaxation process as a bespoke application is speed. Both 3D Studio Max and CATIA™, have slow update times in running code. This is because both applications require advanced features and
are built to be robust. These softwares address many problems in 3D modelling, speed of execution is not the main priority of the software. A specialised application was necessary to run the dynamic relaxation at a more interactive rate. The interactive rate would mean the user could edit the underlying ellipsoid surface as the Dynamic Relaxation solver computed the pattern.

The Dynamic Relaxation process requires projection of the point relaxation to a freeform surface as the relaxation solver executes. The process requires the use of a software library which uses BREP geometry. The openCASCADE™ technology (OCCT) is a BREP geometry kernel. This kernel provides many facilities for creating BREP geometry and evaluating properties on the geometry. For example, the library enables the projection of a point normal to a surface, to find the intersection curve between two surfaces or to project a point normal to a curve. This library also enables the definition of solid models based on both BREP and NURBS primitives, and the computation of Boolean operations between them [SAS, 2014].

Determining which vertices should be constrained to the bottom edge of the truncated ellipsoid and which vertices should be constrained to the visible surface was straightforward. The boundary vertices in the two dimensional mesh (free vertices) constrain to the bottom edge of the truncated ellipsoid, all other vertices in the pattern constrain to the visible surface of the truncated ellipsoid— this means the vertices lay on the area of the surface which is not trimmed and are free to move on the surface. The boundary vertices in contrast, are constrained to the bottom edge of the truncated ellipsoid, this edge forms an ellipse, the vertices a free to move within this curve freely.

Because the development of the two dimensional pattern is a trivial task in 3D
Studio Max™, the software was used to develop the two-dimensional patterns. 3D Studio Max™ was used for producing multiple two-dimensional patterns to test as candidate patterns over the ellipsoid surface. Before these patterns could be used, a pattern design tool was developed as a script which extracted the topological data of a polygonal mesh as a CSV file.

With this data system in place, it was easy to design various two-dimensional patterns in two-dimensions and export the designs as CSV files.

Now that the topology file system was ready, it was necessary to implement the dynamic relaxation solver to consume the CSV topology files exported from 3D Studio Max™. This Dynamic Relaxation solver had to work with BREP/NURBS geometry as the solver required the projection of the topology to a surface and to a curve.

The solver was implemented using the openCASCADE™ technology for BREP/NURBS operations and the Qt™ framework for developing the GUI (Graphical User Interface) of the bespoke application. The application was developed in the C++ programming language. The application was developed without wrapping the OCCT library. This software development design consideration, presented various challenges to the software development process.

While the use of openCASCADE™ proved to be a successful library for developing bespoke 3D applications, using the library directly presented various challenges as indicated in the introduction to this approach.

The robustness of the OCCT technology, enabled the implementation of a fast executing, iterative, dynamic relaxation solver, the flexibility of the Qt™ library in developing custom GUI’s also enabled the development of an intuitive bespoke GUI for computing the relaxation process.
Computing the projection of points towards a freeform surface is a non-trivial problem. OCCT technology trivialises the projection problem, to both surfaces and curves, while maintaining BREP/NURBS accuracy. Without having to understand in detail how the projection algorithm works, the OCCT library abstracts operations on BREP/NURBS solids, surfaces and curves. OCCT abstracts operations such as intersections, projections, trimming, Boolean cut and merge, on solids, surfaces and curves.

The GUI for the bespoke system shown in Figure 4.4 shows how the system was implemented as a node-based data-flow system. The user specifies the topology files in the GUI, and then interactively changed the dynamic relaxation step. A large step makes the dynamic relaxation system move quickly and a small step makes the dynamic relaxation system move slowly but more accurately.

After the user finds the pattern relaxation satisficing, the user can then export the relaxation pattern as either an IGS file or STEP file with the geometry data. The user can also export the model as a CSV file with both topology data and XYZ coordinates for each point in the network. These data exports can then be used on the next process, wrapping the reciprocal frame truss system on the relaxed mesh.

After computing a relaxed base mesh, the mesh needs to be converted into a reciprocal frame truss. An initial system for computing this step was developed using OCCT, but the final system which computes the reciprocal frame truss was developed in Grasshopper™ by the other members of the team.

The use of the OCCT library and C++ for developing a bespoke design software, opened up an experimental possibility in design computing that was not possible with the use of standard parametric software. With this system an
Figure 4.4: Bespoke application developed for the Dermoid project to compute the dynamic relaxation over the ellipsoid. This tool is a node-based parametric model which enables importation of any two-dimensional mesh design from a custom CSV file format and interactive manipulation of the relaxation process.

Advanced-end-user-developer can develop a custom software and a custom graphical user interface to resolve complex design problems not easily resolved using standard software.

While this system enabled the mapping of the dynamic relaxation geometrical heuristic efficiently, the code developed for this project was not easy to recycle on other projects. The domain specific code written to solve the dynamic relaxation problem was mixed with the code written to interact with OCCT library. Because of this lack of modularity the prototype left room for improvement. This search, culminated in the development of an effective strategy to increase the re-use of programming knowledge and to enhance the process of developing software prototypes for design problems.
Figure 4.5: Dynamic Relaxation Process of pattern one. This figure shows different time snapshots of the dynamic relaxation process of the Dermoid in 3D space (bottom) and UV space (top). The pattern begins in a distorted state, as the process converges, the pattern becomes uniform, removing the distortion at the “poles”.

Figure 4.6: Dynamic Relaxation Process of pattern two. This figure shows different time snapshots of the dynamic relaxation process of the Dermoid in 3D space (top) and UV space (bottom). With this pattern the process of relaxation becomes more evident.
The following section will show the use of a library wrapper as a possible solution to issues encountered in the software development process of the Dermoid OCCT bespoke software prototype.

4.2 STANDARD LIBRARY WRAPPER

This section will expose the development of Bespoke software by developing a wrapper library on top of an existing standard library. By wrapping standard libraries with a wrapper library, developers improve the process of developing bespoke design applications significantly.

The standard library wrapper (SLW) enables the reuse of domain knowledge built from the use of the standard library. By wrapping the library the developer is able to reuse knowledge and extend the standard library.

The wrapper library enables the developer to concentrate knowledge about the use of the standard library into one source and knowledge about the domain into another source. This enables the effective application of the “separation of concerns” (SoC), that is, the separation of the main software into distinct sections, each tackling its own concern [Dijkstra, 1982]. The developer and other team members can develop prototypes much more quickly and can both acquire and share knowledge in the use of the library, by separating the code which abstracts the use of the standard library from the code which depends on these abstractions.

Another reason for the “separation of concerns” is the need to use a programming idiom closer to the domain problem. For example, the openCASCADE™ library, has features for performing geometrical operations, these features are
for instance, the extrusion of curves, swept surfaces and complex Boolean operations which are required in solid modelling. While most CAD modelling users by now understand how to use these geometrical operations, these operations are used to build higher level objects, for example: a complex wall, a façade system, a column, a girder, a beam, or a window. When developing the standard library wrapper, the wrapper can use this domain specific idiom to refer to combinations of low-level operations from the library to create higher-order objects. Referring programmatically to a wall, instead of an “extrusion” is intuitive. This in turn, supports work with complex design problems. The encapsulation of one library into another, allows the mapping of more intuitively complex design problems with computer code.

The SoC principle also enables better code reuse. Whereas without it, it becomes difficult to reuse the generic code gathered from the lessons learned in the experience of using the library. In the library wrapper the developer can reuse the generic aspects of the code developed from bespoke applications. This software design principle also enables better maintenance of the code as new requirements and errors are found. The SoC allows the developer to evolve the code and maintain a continuous integration of new releases.

The most important advantage of the SoC is the speed gained from developing quick turnover cycles in application prototype development. The SoC enables the developer to quickly iterate and develop new software prototypes, while recycling the main generic aspects of application building, as new prototypes emerge.

Because the knowledge developed from the use of the standard library lives in a distinct source, separate from the standard library, it simplifies the process
of identifying errors and solving problems with the use of the standard library. Wrapper libraries enable the decoupling of library application solutions from domain specific solutions. When not using a wrapper library, it becomes difficult to distinguish problems with the domain of application from problems with the use of the standard library. For example, when implementing a bespoke 3D application for computing the “dynamic relaxation” of a pattern over a free-form surface. It becomes difficult to distinguish between errors from the incorrect use of the 3D kernel and errors from the domain specific problem of implementing a dynamic relaxation solver. This requires the developer to separate the concerns of computer code written to solve the dynamic relaxation problem, from the code written to abstract operations from the standard library. If the code for solving the Dynamic Relaxation is mixed with the code required to interact with the standard 3D Library, the difficulty in maintaining the code increases. This difficulty also applies when new design requirements emerge, rendering the code inflexible to adaptation to the new requirements. This type of inflexibility can leave the developer no choice other than to resort to erasure-redraw techniques, by having to abandon the code and begin from a blank-slate.

The standard library wrapper enables the separation of the multiple concerns of the software development task into discrete modules. This code wrapping enables a more effective use of software, as the developer can quickly address the source of problems in well-defined localised modules.

For example, the extrusion and splitting of a surface object in a 3D modelling application, can be implemented by using a standard library as-is as shown on the right of Figure 4.7 and with the library wrapper as shown on the left of Figure 4.7.
Figure 4.7: Example of Wrapped C++ code and Non Wrapped C++ code.

This simple example shows how the wrapped library, when compared to a pure use of the standard library, is a more accessible and easier to understand as code. An advantage of using modularisation, is that both wrapped and non wrapped code can easily coexist in a single project. This provides flexibility to develop custom code with ease.

The wrapped code helps standardise the use of the standard library and to develop a well-defined interface for interacting with the source code in the standard library. The wrapped library technique can be used with both open-source and proprietary software libraries. Because the developer is not making changes to the standard library, the distinction between open-source libraries and closed sourced libraries is trivial. The wrapper libraries use the components of standard libraries as-is, providing integration code where necessary to abstract complex
use-patterns.

Wrapping standard libraries with another library which simplifies the use of the first, is not the only method to abstract the use of complex programming sequences with simpler programing sequences.

Meta-programming\textsuperscript{14} is an alternative software development technique, which enables developers to use abstraction in programming complex applications. The process of abstraction of meta-programming, is different from the process of abstraction enabled by the modularisation of SoC. Meta-programming enables the development of a high-level programming language by wrapping an existing low-level programming language with a macro language. The meta-programming system works by developing text-based transformations from one language into another. For example C++ enables meta-programming by using a preprocessor macro interpreter. The meta-programming macros enable the developer to create a new language by transforming textual commands from one language into another by performing only lexical analysis of the text. The pre-processor performs simple text replacements of tokens from one language towards tokens of another language, using user-defined rules. For example the C++ programming language has a restrictive syntax, which relies on semicolons to mark line endings and carets to indicate the opening and closing of program scope. To convert from one syntax into another, the developer writes a macro which pre-processes the code (see Figure 4.8 for an example of a meta-program and its comparison to normal C++).

In this example the macro-language enabled the elimination of the concern to

\textsuperscript{14}Meta-programming is a programming technique which enables software code to write or manipulate other software code.
Figure 4.8: Example of Meta-Programming, showing the differences between C++ normal syntax and the syntax extension provided by creating a meta-program.

<table>
<thead>
<tr>
<th>Normal C++</th>
<th>Meta-Program</th>
</tr>
</thead>
</table>
| double VARNAME = VALUE;
| If(condition){
| }Else If(condition) {
| }Else{
| }                                      | SETDOUBLE(VARNAME,VALUE) = double VARNAME = VALUE;
| IF(CONDITION) = If(condition){
| ELSEIF(CONDITION) = }Else If(condition) {
| ELSE = }Else{
| END = };                                |

**C++ Example**

| double A = 50.0;
| If(A > 20){
| dosomething
| }Else If(A < 60) {
| dosomethingelse
| }Else{
| dosomethingdifferent |
| }                                      | SETDOUBLE(A,50.0)
| IF(A > 20)
| dosomething
| ELSEIF(A < 60)
| dosomethingelse
| ELSE
| dosomethingdifferent
| END |

use the opening caret “{” and closing caret “}” to signal opening and closing of code scope. It also removed the need to define a variable by its type and the need to indicate line ending with a semicolon.

While this example is trivial, the macro-preprocessor technique can scale to complex language translations enabling the encapsulation of programming knowledge into a new high level language\(^{15}\). This enables the programmer to develop an auxiliary programming tool, which enables a developer to program by using his own syntax and programming directives. By this meta-programming technique, the developer can enhance code expressiveness over syntax. The code

\(^{15}\)In fact the Bourne Shell, the default command-line interpreter in what was Unix Version 7, was developed by Stephen Bourne while trying to make C look more like Algol [Kernighan and Mashey, 1979]. He used the meta-programming technique described in this section.
also becomes more human legible, and as such, it becomes easier to modify and
share. This technique also enables novice programmers to learn how to program
using the meta-programming language. They can then gradually transfer into
using the low-level programming directly.

Meta-programming is a computer programming technique where the devel-
oper writes a meta-program. This meta-program is preprocessed by a preproces-
sor program, the output of the preprocessor becomes the input to the compiler.
This programming technique allows the programmer to translate code written in
a high-level language to code written in a lower-level language before it is fed to
the compiler, and converted into a binary application.

Both the library wrapper and the meta-programming abstraction techniques
are employed in the project which will follow. These two advanced methods
have their advantages and limitations and can be used together to provide a high
level of code modularity and code expressiveness respectively.
4.2.1 RESPONSIVE ACOUSTIC SURFACING

PROJECT OVERVIEW

Figure 4.9: The workshop space and participants of the Smart Geometry 2011 at CITA in Copenhagen.

This section will expose the Responsive Acoustic Surfaces (RAS) project, a wall system developed to scatter sound using hyperbolic plaster surfaces. The project focused on the acoustic properties of hyperbolic surfaces when arranged in a wall system. The project brief started, from an anecdote from the musicians at the inauguration ceremony of the Sagrada Família Basilica who reported that there was a diffuse acoustic. This led to an investigation into the sound scattering properties of intersecting hyperboloid surfaces of hard reflective material (in common with the internal nave walls of the Sagrada Família Basilica).

The project was setup to develop different geometrical configurations of hyperbolic surfaces and their intersecting patterns to produce wall assemblies made of plaster components. Two walls would be developed, one with intersecting hyperboloids and another as a flat curved wall. This workshop would test whether
the hyperbolic surface would scatter sound better than a smooth base reference wall. It would also lead to comparative sound scattering measures over frequency ranges for different specific applications of the hyperboloid arrays for different surface patterns.

The project was developed as part of the SmartGeometry 2011 cluster “Responsive Acoustic Surfaces” in CITA, Copenhagen.

The workshop main objectives were to develop a wall system which would scatter sound by using hyperbolic surfaces as scattering features. This section will expose only the components of this project related to the development of the parametric modelling system for fabricating the wall system and the use of the openCASCADE™ technology for testing different scattering textures by the workshop participants.

HYPERBOLOID PATTERNING

To test the hypothesis of whether hyperbolic surfaces were responsible for sound scattering, a test was required on a physical prototype. Sound scattering is not easily simulated in computers [Peters, 2010, Rindel, 2000]. The most advanced acoustic software used by acoustician’s, ODEON™, is limited for applications to sound scattering simulation and analysis. Most of these tools rely on raytracing techniques to simulate the sound scattering of 3D surfaces. Raytracing has limited capabilities of modelling correctly sound scattering [Gomes et al., 2004]. For this reason it is necessary to obtain the sound scattering coefficient of materials by using scaled physical prototypes [ISO, 2004]. The ODEON™ software requires the sound scattering coefficient of materials in order to increase the accuracy of the raytracing computation. Surfaces are modelled as flat surfaces in
this software. Their sound scattering coefficient is applied as a material property. Most products are tested by their original manufacturers and their sound scattering coefficients are provided as part of the material specification sheet.

In the Responsive Acoustic Surfacing project, under the guidance of Brady Peters from CITA, the wall system had to be abstracted to small plate disks, where the hyperbolic intersection patterns were treated as surface features modeled at 1/10 scale. The disks were 3D printed and subjected to an ISO standard sound scattering coefficient measuring box [ISO, 2004]. The values extracted from these tests are then inputed in ODEON™ to test the acoustic properties of the rooms. The results from the test conducted on both the hyperbolic wall and the base wall, have been published in many publications [Burry et al., 2012b]. This section will focus primarily on the parametric generation of design models for the 3d printing of the disks.

The process of designing candidate designs was developed using the OCCT technology by the workshop participants, using a software development kit developed specifically to address the production of bespoke design softwares.

Each participant used the SDK (Software Development Kit) provided to them, to develop both a 3D software and with their own software, generate 3D designs of disks to be 3d printed. These disks had to be tested using the sound scattering measurement box provided by Tobias Olesen16. However, each participant also developed their own bespoke software prototype in C++. It is important to highlight that most participants did not have previous experience in developing bespoke software.

---

16The diffusion testing chamber for 1:10 models was set up and the data capture done by Tobias Olesen, Danish Technical University. This setup was arranged through his research partnership with Brady Peters, CITA Royal Danish Academy of Fine Arts.
ing software using C++ (See Figure 4.10 for a sample application developed by participant, Ben Coorey).

Figure 4.10: Sample application of Ben Coorey’s image based bespoke software developed for the Smart Geometry 2011 “Responsive Acoustic Surfaces” cluster. The custom interface on the left was developed using the Qt GUI editor.

For this reason the SDK was designed using both a modular design for wrapping the OCCT technology, but also by providing a Meta-programing language in C++, the meta-programming language enabled the users to write complex C++ bespoke applications using a simple to use programing syntax which got rid of many of the syntactic issues of low level languages such as C++. By using the meta-language, I was able to hide many complicated operations which were necessary to develop a bespoke application in C++. Complex operations such as the initialisation and configuration requirements of openCASCADE™ and Qt™ were wrapped seamlessly into a meta-programming language. Rather than teach the participants how to use C++, I could focus on teaching them the much simpler programming language. The participants were using complex operations
and putting together complex 3D bespoke applications, without no previous experience of using C++ or openCASCADE™.

This enabled novice users to jump straight into developing software prototypes without having to learn the specifics of C++. This system also provided expert users the chance to jump directly to C++ if they wanted to override the meta-language provided, since the meta-language is simply a text-to-text lexical converter. The compiler makes no distinction between code written using the meta-language or code written using C++.

At the same time the SDK developed for the SmartGeometry 2011, used the library wrapper technique discussed in this section. Where the OCCT technology was nicely wrapped into a user friendly library that enabled both Novice and Expert users with little experience in OCCT technology to use the library proficiently. Without the use of the library wrapper, the users would have to learn the documentation of the OCCT technology and internalise the mechanics of the library, before the users would be able to develop working prototypes. Most of the participants in the workshop had previous experience in using CAD modelling and scripting.

While I provided the SDK to the participants, each participant provided the design brief for how they wanted their bespoke software to function. For example, the participant Ben Coorey wanted his software to use grayscale and coloured images to control the distribution of hyperboloids over a disk (see Figure 4.11). His bespoke software enabled him to use images as the controller of a parametric model. The array of pixels in an image and its value was used to multiply the radius or the angle of the hyperboloids.

The bespoke software prototypes developed by the participants created fast
interactive applications for testing experimental design ideas. Nevertheless, we
found a bottleneck in the solid modelling functions of OpenCASCADE™ Ver-
sion 6.3.0; the Boolean cut operations on solids, did not perform as quickly as
required. For this reason I had to develop a script using Digital Project™ to
perform the Boolean computation from the disc plate to be tested in the diffu-
sion chamber. The hyperboloid geometry was still generated and configured in
the openCASCADE™ based tools developed by the participants. The only dif-
ference to the workflow, was the additional step of computing the booleans in
Digital Project™. The models from the bespoke applications were exported as
organised IGS (Initial Graphics Exchange) files. Because the files were organ-
ised internally, the organisation of the model in openCASCADE™ transferred
into Digital Project™.

While the Boolean operation on OCCT technology were slow back in 2011,
today the technology has enhanced the Boolean operations significantly in open-
CASCADE version 6.7.0.

The SDK I developed for the Smartgeometry 2011 participants, required them
to install a list of softwares prior to beginning the software development exer-
cises. For example the participants had to install Visual Studio™ 2008, open-
CASCADE™ technology, the Qt™ library, and other prerequisite technologies.
Additionally the software development process used, relied on the visual studio
software development process for C++ applications, where the users would write
code in the editor, compile, debug, and run the compiled applications. The error
messages returned by Visual Studio™ were too complicated for most users to
understand. The process of compiling and linking software was also too cum-
berson and slow for the majority of the users, even for expert users. Every
Figure 4.11: An earlier version of Ben Coorey’s image based bespoke software (Top) and the result of the Digital Project boolean operations (Bottom).
change in the code regardless of how simple or complex, requires compilation of the entire application. Even when incremental compiling techniques are used the compile process took many minutes. This process is too long for the kinds of changes required in making design prototypes.

The set-up process was found too be cumbersome for many novice users. The need to install many softwares and the highly advanced configurations steps required to setup Visual Studio™ on each participants computer, developed the need to create a new strategy. This strategy had to forgo the use of many installations of software by using a pre-packaged Integrated Development Environment (IDE). The IDE would need to use a simple-to-use scripting language, rather than working in C++ and having to compile. The software had to evaluate the script in run-time without requiring to compile software.

The working system describing how this scripting overlay works is exposed in the next section, the scripting interface.

4.3 SCRIPTING INTERFACE

This section will expose the development of bespoke software by extending the last two strategies with a scripting layer. The scripting layer enables the end-user to use the code written in a low-level language, such as C++, in combination with a high-level language, such as Javascript.

“A scripting language is a programming language that is used to manipulate, customize, and automate the facilities of an existing system. In such systems, useful functionality is already available through a user interface, and the scripting language is a mechanism
for exposing that functionality to program control. In this way, the existing system is said to provide a host environment of objects and facilities, which completes the capabilities of the scripting language. A scripting language is intended for use by both professional and non-professional programmers.” [Ecma, 1999]

A scripting layer enables the abstraction of the use of a low-level software framework as a high-level macro language.

A software library, abstracts domain knowledge for use by another user; this user does not need to know the domain knowledge encapsulated by the software library. It enables wrapping and using knowledge as a black box. This black box enables other users to use the domain knowledge, without knowing its specifics.

Developing a 3D application can be a daunting task, but the knowledge of writing a 3D viewport has been encapsulated by the OpenGl™ library. The OpenGl™ library encapsulates the knowledge required to communicate with a graphic cards to display 3D images on a computer display. The OpenGl™ library wraps the mathematics required to transform vector primitives into pixels on the screen (rasterization) [Shreiner et al., 2005].

When combining multiple software libraries, the end-user gains the knowledge encapsulated in all the libraries, combined as a technology stack. For example the OpenCASCADE™ framework, uses the openGl™ library to create a viewport object. This viewport object is an abstraction of the OpenGL 3D viewport, but at a much higher-level. This viewport object, abstracts typical viewport operations in most CAD software. For example, Pan View, Zoom extents View, Rotate View and Selection of Objects in the viewport.
Because the technology stack already contains the requirements to develop most off-the-shelf software, the end-user-developer does not have to worry about the development of these standard features. The end-user developer instead focuses on how to combine these features to design new softwares. Since the standard library already has the majority of the required features embedded, the designer can then focus on the development of the user interface and the domain specific aspects of the software.

Traditional parametric modelling software uses a data abstraction called a directed acyclic graph to propagate changes across associated elements in the model. I have documented in Chapter Two: The Graph, the issues which emerge from using graphs as representations of the design intent. In this section, I will show alternatives to using graphs for the representation of the design intent.

The research reported in this thesis investigates an alternative strategy to parametric modelling, where explicit modelling and parametric modelling are combined in a hybrid composite through the use of a scripting overlay.

Acyclic graphs are needed to process the dependency graph of feature based software. Acyclic graphs also enables the application to traverse all components once during model update. This feature of acyclic graphs, enables the parametric modeller to be effective and run at a fast execution speed. In bespoke software, where features are not used, the associativity in the model can be computed by using a scripting language.

If the speed of execution of the script is fast, then using a script or a directed acyclic graph does not make a difference to the end-user. By using a script as the model, the user can represent procedural knowledge more easily. The script as the model requires that the scripting language be simple to use and make sure
it does not introduce too many restrictions to the end-users.

The scripting interface developed in this section was composed of two key elements and two sub elements:

1. An Application Programming Interface, which encapsulates the openCASCADE™ technology as a wrapped C++ software library. This API was developed using the SoC principle outlined in the standard library wrapper section.

2. An Integrated Development Environment (IDE), for writing parametric models as scripts. With an easy to use Javascript language. The Javascript scripting language, wraps the openCASCADE™ API with a simple to use scripting language, enabling the use of the API both as a pure C++ Library and as an interpreted Javascript library.

   a) A 3D graphic viewport, for visualising the result of executing the script.
   
   b) A front-end panel where visual sliders appear for tuning parametric model variables.

To enhance ease of use, the script editor and the front-end slider deck are mutually linked. Changes in the code editor produce changes in the slider deck, and changes in the slider deck produce changes in the values of the code. By using this mutually linked representation system, the variables in the code automatically become sliders in the slider deck. If the name of a variable is changed, the slider which corresponds changes its name as well. If the variable is removed from the code editor, the slider is removed from the slider deck. This behaviour
of the IDE, is automatic. The user does not need to create sliders and then link
them manually to the code. The link between variables in the back-end (code
editor) and the front-end (slider deck) is black-boxed to the user, enabling the
end-user to worry only about the code, and not to worry about how the software
makes links between visual interfaces and computer code (see Figure 4.12). In
traditional parametric modelling systems both visual and non visual, the user
spends a considerable amount of time doing mechanical activities, such as link-
ing sliders with variables. The easy to use slider-variable relation was developed
to avoid a distraction from the design process.

Figure 4.12: Diagram indicating the slider-variable relation between sliders in
the slider-deck and variables in the computer code.
The final iteration of this scripting system is shown in Figure 4.13. This system enables the development of complex parametric behaviour from simple script instructions. To flex the model, the user switches to the front-end and manipulates (by dragging the slider handle or typing the value directly in the number box) the sliders which control the model. The simplicity of the IDE, means the user can begin using the system by simply opening the interface and pasting a model from text or by writing new code.

The scripting system was developed in the C++ programming language using the Qt™ framework for application development and GUI development; the openCASCADE™ framework was used for 3D geometry creation and manipulation as well as 3D graphic display. The Qt™ framework provides facilities for developing Graphical User Interfaces with default behaviour. The Qt™ frame-
work, also provides a Javascript binding system which enables to wrap C++ objects and functions, using a Javascript scripting engine\textsuperscript{17}. This engine is the Qt\textsuperscript{TM} QtScript\textsuperscript{18} engine. Developing a script language using the QtScript system was used for convenience, given the rest of the application was developed using the QT library, there are other alternative scripting language binding technologies that can be used to develop a scripting language from C++ code. Examples of these Libraries include Google V8 a javascript wrapper for pure C++ applications, PySide Shiboken a python wrapper for Qt applications, QtLua an alternative to the QtScript module to make Qt applications scriptable in Lua, etc.

\textsuperscript{17}While the QtScript Javascript system was used for convenience, given the rest of the application was developed using the QT library, there are other alternative scripting language binding technologies that can be used to develop a scripting language from C++ code. Examples of these Libraries include Google V8 a javascript wrapper for pure C++ applications, PySide Shiboken a python wrapper for Qt applications, QtLua an alternative to the QtScript module to make Qt applications scriptable in Lua, etc.

\textsuperscript{18}The QtScript scripting engine, is a scripting engine provided by the Qt framework. It enables to expose Qt based objects(derived from the QObject) into a javascript(ECMA–262) interpreter. This allows for C++ objects to interact with javascript code with ease. Both C++ can make calls to javascript code, and javascript can make calls to C++ defined code( as long as this code is registered to the interpreter).
tem is relatively direct. Each command available in the standard library wrapper was again wrapped by using the QtScript binding process\textsuperscript{19}. The wrapping of the standard library wrapper with a simple to use scripting language; allows the end-user to use the openCASCADE\textsuperscript{TM} API as a simple high-level scripting system, rather than having to work directly with the mechanics of developing low-level C++ applications. For example, Figure 4.15 shows the successive wrapping of the makecircle command; in this command, the user must specify 3 points as input for the creation of a circle which passes through the 3 points. Figure 4.15 shows how the script command \texttt{makecircle}, is in fact a complex instruction that has been wrapped in C++, as shown in the \texttt{C++:Binding Code} of the same figure. The code in the \texttt{C++:Binding Code}, also shows extra code around it to avoid basic user input errors, such as making sure none of the three input points are coincident. Equally Figure 4.15 shows how the \texttt{C++:Binding Code} is making a call to the \texttt{C++:Wrapper}. The \texttt{C++:Wrapper} is an extract from wrapping the openCASCADE\textsuperscript{TM} circle function as shown in the AddNewCircle function. This example shows the two levels of abstractions described both in the preceding section and in this section. The first level, provides abstraction to the advanced-end-user developer by wrapping the openCASCADE\textsuperscript{TM} code into a pre-assembled wrapper library for building bespoke 3D software. The second level, provides abstraction to the end-user to enable them to use the low-level API as a simple to use scripting language.

\textsuperscript{19} A binding process, is the process by which a C++ function or component is wrapped by a scripting command. Enabling the script engine to make calls to the C++ code when the user invokes the script command.
Figure 4.15: Diagram indicating the successive abstraction concept. Script command “makecircle” is an abstraction of the C++:Binding Code “make3pointcircle” which is an abstraction of the C++:Wrapper “AddNewCircle”.

The use of a scripting overlay above a bespoke software, enabled a successive separation of concerns, with a greater flexibility\(^\text{20}\) in the programming development process than the process described in the Standard Library Wrapper.\(^\text{20}\)

The kind of flexibility referred to here is the flexibility to produce structural changes to the behaviour of the application without having to stop the application. For Example, the scripting interface enabled one to change on the go the entire design-logic of the application, whereas on the last two strategies any change regardless of how big or small requires the designer to stop the application recode and recompile the application. Compared to the scripting interface, the last two strategies are sluggish at least in terms of how they enable the user to experiment quickly with the behaviour of the application.
section. This flexibility for example, enables the writing, editing and execution of scripts on the fly without ever leaving the programming IDE. For example, the standard library provides the abstraction of the BREP/NURBS operations available in the openCASCASDE™ library; the standard library wrapper simplifies the openCASCADETM library by providing well-defined interfaces to this code. The scripting overlay provides the same benefits as the wrapper library in a scripting language rather than as a programing language. The differences between a programming language versus a scripting language are subtle. One major difference between these two types of coding, is in the way the coder runs the code. In programming the developer writes computer code, compiles the code and executes the resultant executable application, if errors are found the developer has to stop the application and begin from the first step in the process by modifying the code. In scripting, the developer starts the application and creates code directly, he evaluates his code without having to stop the application, the code is evaluated Just-In-Time by the interpreter and errors are highlighted and the code is modified without having to stop the application. The scripting process is much faster, and requires less code than the programing process, because the scripting process has been developed to extend an existing application. Whereas the programming process creates a new application. One major difference though is in the performance of the software. Because scripting relies on interpreting the code on the fly, there is a performance overhead which is
not present when software is compiled\textsuperscript{21} rather than interpreted. However, while scripting languages may have a lower performance when compared to using a low-level language such as C++, the high-level abstraction afforded by scripting languages enables a level of expressiveness not attainable through low-level languages \cite{stackoverflow, 2014}.

While the scripting system described here was developed using C++, the compiled end product of this system, is a stand-alone IDE which enables the user to write, edit, debug and compile scripts on-the-fly (Just-In-Time compiling). The system enables an advanced-end-user-developer to extend the system in C++ to add script commands. In this way, both expert and novice users can use and extend the system according to the user’s level of expertise. Users which know how to script can extend the logical system using the scripting language. Users who can program in C++ can extend the physical design\textsuperscript{22} of the system. Equally, expert users can also use the scripting system to test and debug their C++ code, and to speed up the process of development.

In this section I showed the use of both programing and scripting as hybrid techniques for developing flexible bespoke applications which enable the ends-

\textsuperscript{21}Because scripting skips the explicit compilation step it speeds up the process of development, however the mechanisms which enable scripting languages to skip explicit compilation decreases the performance of the code. For example, the environment that enables the scripting might need to do reflection to examine and modify the structure and behaviour of the script at runtime.

\textsuperscript{22}Most books on software programming, focus on the logical design of software. However, a poorly implemented physical design (how the software is actually implemented into components, classes and interfaces) of the software may cripple the elegance of the logical design of the system. For a thorough discussion on the distinction between logical design and physical design of software systems see \cite{Lakos, 1996}.
user to develop flexible models, and the advanced-end-user to extend the under-
lying application capabilities. Because the system is extensible both at the do-
main level and at the infrastructure level, the use of a strategy which combines
explicit modelling and parametric modelling in a hybrid composite through the
use of a “scripting interface” proved to be the most flexible strategy in the be-
spoke chapter.

The scripting interface strategy enabled a more flexible modelling process
compared to the other strategies in this chapter; however, the bespoke applica-
tion with scripting interface, is not as flexible as the standard application with
scripting interface in some complex design scenarios. The standard software
may be more flexible in some problems than the bespoke software. However,
the bespoke software can be used for problems which require real-time feedback
and where the standard software would be too slow. For example, the complex
puzzle-like detailing of the FABPOD and Yas Island projects are best addressed
as hand-modelled bottom-up assemblies in standard parametric modelling soft-
ware, using algorithms occasionally only to instantiate the parts and configure
them to their context. Whereas the Dynamic Relaxation process described in
the Dermoid and the sphere-packing process described for the Museo Soumaya
are best addressed through a bespoke software with a scripting interface, due to
the algorithmic nature of iterative design problems. However, both bespoke and
standard software can be combined. For example, wire-frame geometry can be
generated in the bespoke software and then used by the standard software to in-
stantiate more robust and complex detailed models developed using standardised
interfaces; for example, the FABPOD used low-res wireframe geometry gener-
ated in Grasshopper™ to develop the high-res Digital Mock-ups in CATIA™,
the Grasshopper™ step can be replaced with the scripting interface described in
this section.

This section demonstrated a flexibility strategy for the development of a be-
spoke application with scripting facilities. The use of scripting facilities over
bespoke software, enable the end-user of the bespoke software to write custom
domain specific scripts. The advanced-end-user who created the bespoke appli-
cation can extend the application by adding more commands in the programing
language used by the end-user. This strategy allows these two kinds of users to
work together and develop complex bespoke software by integrating programing
and scripting in the development of complex parametric models.

4.4 SUMMARY OF BESPOKE APPROACH

The use of the standard library demonstrated in the Dermoid project, showed
the many limitations which emerge from developing bespoke softwares without
a modularisation strategy. For example, reuse of the knowledge developed in
the project was difficult, and thus the code became difficult to understand. This
meant the code developed for the application could not be used for other soft-
ware. One of the central features of standard parametric modelling systems like
CATIA™, is its facility for representing modelling knowledge and thus enabling
knowledge capture and reuse. To provide knowledge reuse, I developed the stan-
dard library wrapper strategy, as a strategy which enables the abstraction of the
use of other API’s and SDK’s in the development of bespoke software. Lastly
to enable the end-users to take advantage of the API’s developed, I created a
simple to use scripting language, which abstracts the standard library wrapper,
and thus enables the end-user to access the facilities provided in the library as normal script instructions.

In the Standard Library Wrapper, I address how a bespoke software could be developed to enable the same domain agnostic benefits of standard parametric software. The use of a modular approach by wrapping software libraries and the use of a meta-programming language enabled the advanced-end-user to capture and reuse software development knowledge. The “separation of concerns” (SoC) principle was introduced and the need for modularity. The use of wrapper libraries, demonstrated how this software development method enables the encapsulation of use-case knowledge. This method also demonstrated how by wrapping a library, the complexity of a library is turned into an Input Process Output system. This in turn, enables the creation of complex systems with minimal efforts.

The meta-programming technique, demonstrated how this old technology can be used to enhance the process of learning of a novice user and for speeding-up the programming process for advanced end-user developers.

While both the library wrapping technique and the meta-programming technique provided an enhanced method for developing bespoke software applications, these techniques were slow and cumbersome for developing fast iteration of designs.

The scripting interface section showed the development of a bespoke application with an Integrated Development Environment and an embedded scripting language interpreter. This strategy removed the need for compiling. The system was redesigned, so that only one application is needed for using the system. This is similar to processing, where the process of writing a sketch, is simple and re-
quires only to download the IDE; no setup requirement is necessary other than to learn the scripting language. This language was designed from an end-user’s point of view, while enabling advanced users to add new features by extending the programming language in C++. This enables both novice and expert users to use the scripting language for developing complex models. Only when the language does not have a command, the expert user needs to go back to the typical process of coding and compiling to add the command to the underlying system. This layered approach enables the development of a complex bespoke scripting system, with a standard way of writing domain specific models as scripts.
Chapter 5

SUMMARY OF FINDINGS

This thesis has explored strategies which examine the extent to which the parametric modelling paradigm can cope with ways of enabling disruptive change within flexible design environments. My research has been tested primarily through project work. These projects have been documented in-depth throughout the case-study narratives of chapter 3 and 4. In this chapter, I will discuss the implications of my research, addressing the challenges I faced during the delivery of the projects described in the case-study narratives and the key result of this thesis: that a “strategic manipulation of the parametric design software can provide designers with a higher-level of control and flexibility in the delivery of ill-formed design problems”

Central to this thesis, is the idea that to overcome inflexibility in design modelling, designers should be more critical towards the ability of the parametric design software in providing design flexibility. To minimise the exposure of architectural design projects to the limitations of parametric design software, I have developed strategies which require the development of computer softwares
which control the parametric design software and assist it in overcoming inflexibility and greatly enhancing control over the parametric design process. To test whether these strategies can provide designers with a higher-level of control and flexibility in the delivery of ill-formed design problems I have tested my proposition through eight case-studies of projects with unconventional levels of design complexity.

5.1 OVERVIEW

In Chapter 1, I introduced the aims and scope of this research and the problem of design modelling inflexibility. From the outset, this thesis posed the question of whether the parametric modelling paradigm could be more effectively augmented through a substantially revised technology and strategy to address limitations in flexibility found within the parametric design technology and process? By asking how specifically this augmentation to “flexibility” takes place, it was possible to investigate how these enhancements to “flexibility” enable designers to confront the challenges posed by the rationalisation of ill-formed design problems with the aid of parametric design software. In chapter 1 I presented the action-research and case-study methodologies and explained their advantages and limitations for testing the propositions of this research.

5.2 UNDERSTANDINGS OF THE PROBLEM

In Chapter 2, I examined a circumscribed literature review around challenges to flexibility within the parametric design process, as well as the background in
which this process takes place. This review explored key characteristics of the parametric design software and its role within the AEC industry. In the industry section, I presented some attitudes and concerns of key researchers over how the parametric design software affects the design process, as well as the advantages that these technologies bring to the design process. In the rationalisation section, I presented the rationalisation process, a key application of the parametric design software for the legitimation of ill-formed design problems. In the phenomenon section, I introduced the core topic of parametric design “inflexibility” and the challenges this “inflexibility” presents to the designer during crucial moments of design decision making. And finally, In the graph section, I presented some key characteristics of the graph data-structure and how when used to represent design problems, graphs become complex networks. During this section the negative characteristics of complex networks were identified as a potential factor conducive of “inflexibility” within parametric design software.

Within Chapter 2, I found the following central concerns impacting upon the inflexibility of parametric design software: issues which emerge from the use of the parametric design schema; Issues which emerge from the use of the graph as a design representation; Issues which emerge from the intractability of design problems and solutions. Next I will enumerate the observations within each of these three categories more specifically:

Issues which emerge from the use of the parametric design schema:

1. The parametric design schema is paradoxically both flexible and constraining. It enables deferral of design decision-making to the later stages of the design process, however this alleged flexibility comes at a price, as the
schema restricts flexibility in design modelling due to its highly structured design process.

2. Due to its restrictive schema, the parametric design modelling paradigm is best applied to the later stages of the design process.

3. The structured planning process required in maintaining parametric design models is incompatible with the loose planning process used with other design media. There is no room for ambiguity within the parametric modelling paradigm.

Issues which emerge from the use of graphs as a design representation:

1. The complex map of relations in a parametric model eludes understanding and makes it difficult for a designer to anticipate model performance.

2. The hyperspace produced between the networks of relations in parametric graphs, both facilitates and hinders “flexibility” in design modelling.

3. The model brittleness and the constrained variability designers are facing when using parametric models could be attributed to the characteristics of complex networks.

4. The acyclic directed graph of parametric design software does not enable parametric design software to represent comprehensively all kinds of design problems.

5. Graphs inherently are difficult to understand, present structural complexities, have dynamical complexities over time, have combinatorial complex-
ities (which emerge from its node diversity and connection diversity) and suffer from meta-complications [Boccaletti et al., 2006, Strogatz, 2001]

Issues which emerge from the intractability of design problems and solutions:

1. Design problems are inherently messy. They require interpretation and cannot be comprehensively stated at any stage of the design process [Lawson, 2006]. By having to develop a “schema” ahead of the conceptualisation designers must have an intentional clarity that may not always be present.

5.3 EVALUATING INTERNAL AND EXTERNAL CODE SOLUTIONS

In Chapter 3, I presented the Standard Strategy. This chapter introduced two “hybrid” strategies for enhancing flexibility in the design process supported by off-the-shelf parametric design software. This strategy presented first the “Internal Code” sub strategy and second the “External Code” sub strategy. Both the “Internal Code” and “External Code” strategies manipulate the structure of parametric models programmatically through algorithmic controls. By managing both the creation and editing of complex parametric schemas through software interfaces, designers do not need to interact directly with the restrictive schemas of parametric design software. By shifting control from the schema to a software interface, the designer is unburdened from the tedious task of maintaining the complex structure of parametric design models. However, shifting
control of the schema to a software interface still requires expertise in the manipulation of parametric design schemas. The software interface which manages the schema must be developed by the designer through sole authorship or by the collaboration between a designer and a scripter through shared authorship.

The “Internal Code” strategy, examined the opportunities and limitations of embedding user-written computer code within parametric models, increasing interactivity by reducing the latencies between user interaction and model feedback. In the “Internal Code” section I presented two case-studies of projects employing these user-written responsive computer codes.

First, the Sagrada Família Hyperbolic Bridge project: case-study one, examined the pre-design parametric model of a pedestrian bridge for the Sagrada Família Basilica. The strict geometrical codex of the hyperbolic geometries, the site conformance requirements, the need to use an optimisation process and the requirements of disruptive change make this project a highly complex project. In this case the use of the “Internal Code” strategy enabled the embedding of a software tool which provided the design team with extreme variability. The embedded tool also enabled the use of the CATIA knowledgeware optimiser on the user-control parameters. This highly adaptable and user-responsive model enabled the design team to cope with the high-levels of complexity required in this project. This parametric model utilised embedded algorithms to generate parametric design variants which could accommodate disruptive change by providing immediate feedback during user interaction.

Second, the retrospective analysis of the Sagrada Familia Passion Facade colonnade model: case-study two, examines the “what-if” scenario of how this embedded user-written computer code would have enhanced the design mod-
elling process of this project.

The “External Code” strategy, explored the opportunities and limitations of developing customised external software interfaces to pure parametric models. This strategy examined how the “Direct Modelling” features of standard off-the-shelf parametric design softwares can be combined with external “computer code” to create what I am defining as a “Hybrid” model. I define “Hybrid” models, as models which combine parametric models which result from algorithms with parametric models which result from “Human Computation”. The robustness of this strategy permits designers to model design problems of great complexity. The “External Code” strategy enables the end-user to shift his attention from schema building activities towards the development of external software interfaces which manage the parametric schema. In a sense, these external software interfaces become simpler interfaces to complex design problems. Through the process of implementing the “External Code” strategy, by building external software interfaces to parametric models and establishing the well-defined interfaces with which the external interfaces interact with the model; the user is able to separate the limitations which parametric modelling software present when representing complex design problems, from the limitations designers experience when defining or structuring complex design problems through parametric schemas. This strategy also demonstrated the value of decoupling geometrical complexity from design-logic complexity to facilitate a more visual process of interacting with tectonic issues and the need for topological variability in traditional parametric modelling paradigms. This proved to be one of the most valuable contributions of this strategy, as it enables practices to combine team members with or without scripting background in the design process. The
team members without a scripting background, can develop the smart parametric features (user-defined-features/power-copies) required in the project, while the team members with scripting knowledge can develop the computer code which assembles the smart features into coherent assemblies. This means designers can use their visual intelligence to develop complex 3D component libraries or to resolve complex details visually.

The “hybrid” modelling enabled by the “External Code” strategy enables the efficient utilisation of both human resources and computational resources within a team, by correctly matching design problems to either humans or computational means. This loose-coupling between many human resources and available computational resources creates a human-computer symbiosis\(^1\) which enables the achievement of an increase in the efficiency with which different kinds of problems are tackled, by allocating resources efficiently. For example, while the encoding of a simple re-usable parametric articulate part is an easy endeavour for a human designer, this task on the other hand is non-trivial for an algorithm or any kind of software development process. However to take the reusable component and propagate it using iteration and complex rules of instantiation requires little effort for an algorithm, whereas this task is non-trivial for a human designer.

Architectural designs are idiosyncratic due to their specific geography, client, point in time, programme and specific design considerations; The “hybrid” approach which the “External Code” permits, allows designers to capture the idiosyncrasies of ill-formed design problems efficiently through the mixture of

\(^1\)The use of humans as problem solvers without trying to understand their underlying cognition model has been proposed by many researchers [Licklider, 1960]
manual knowledge capture with the knowledge capture provided by algorithms, to capture complex design intent relations and rules. Allowing designers to manage the trade-off between elements of the design which due to their subjectivity require professional judgement, from elements of the design which due to their well-formed rules and parameters can be made explicit in the form of algorithms and well-defined software interfaces.

In the “External Code” section, I presented four case-studies which used the “External Code” strategy to manage the translation process between schematic design models and more robust design production description models: The Yas Island, The Museo Soumaya, The Fabpod and The Glory Facade Helical Stair of the Sagrada Familia Basilica.

In the Yas Island case-study: case-study three, I examined the use of the “External Code” strategy for the automation of Digital Mock-Ups (DMU). In this project account, I identified the trade-off between maintaining flexibility through wireframes and the robustness enabled by the high levels of detail which the DMU provide. One of the most important design considerations provided by the DMU, is its ability to enable designers to gauge the “assemblability” of a parametric design model. However the DMU requires a level of detail and model complexity that is not possible to achieve through the structuring of parametric models by using the keyboard/mouse interface. Projects like the Yas Island GridShell— which for instance was made up of more than 250,000\(^2\) inter-related parts, this number only reflects the amount of parts within the gridshell assembly. The overall model assembly also includes the buildings which are enclosed by the Gridshell. These support models were modelled using more conventional methods of parametric modelling, that is, without the aid of scripting, through the keyboard/mouse interface.
components—depend on the development of customised computer software interfaces which automate the production of these complex models. However, while the automation provided by the “External Code” interfaces were crucial to the project, it was the synergy between re-usable models developed through the hand-eye-coordination provided by human computation\(^3\) (with well-defined interfaces) and the external software tool which enabled the robustness in flexibility required to deliver this project in such a short amount of time and with this degree of complexity. The synergy between hand-eye-coordinated parametric models and automated parametric models refer to the “Hybrid” modelling concept defined above in the summary to the “External Code” strategy. Through this Hybrid modelling strategy it was relatively straightforward to develop complex models by assigning the design of re-usable articulate parts to human computation and the propagation of these articulate parts to external software interfaces written specifically to address the problem at hand.

In the Museo Soumaya case-study: case-study four, I examined the rationalisation process for the façade of the museum. The sophisticated methods used by the architect in the shaping of the facade, presented difficulties in the planning and manufacturing process of the facade. The facade surface of the MUS exhibited curvature on both directions of its parametrisation. This in turn challenged the rationalisation of the structure due to limitations of materials and methods of construction in representing faithfully the design intent in the model. Similar to the Yas Island, in this project, a combination of hand-built assemblies combined with “External Software” interfaces, enabled to map a complex rationalisation

\(^3\)“Human computation” has been defined by Von Ahn as “…a paradigm for utilizing human processing power to solve problems that computers cannot yet solve” [Von Ahn, 2009].
process. Through trial and error we arrived to a sphere-packing system for the
discretisation of the free-form facade shape into uniformly distributed hexago-
nal panels. The Museo Soumaya initially contained more than 16,000 unique
panels. The high panel count increases complexity and cost of manufacturing.
Through a series of K-means cluster analysis exercises, the unique panel counts
where reduced, lowering the fabrication and assembly complexity of the facade.

The flexibility enabled by the “External Code” strategy, is demonstrated in
the Museo Soumaya, by the ability of this strategy to enable to map ill-formed
design problems with ease. For example, while CATIA did not include a sphere-
packing tool out of the box, it was relatively straight forward to implement one.
The sphere-packing tool was developed by creating re-usable components as
well as external software interfaces which manage the re-usable components.
This ability to quickly represent a complex process by combining re-usable para-
metric models with computer code, enables designers to expedite the process of
trial and error necessary to find solutions to ill-formed design problems. De-
veloping computer code alone, is too slow a process to map complex design
problems. Had we mapped the sphere-packing process through computer code
alone, we would have had to invest upfront a large amount of time to develop
a robust sphere-packing software tool. The short attention span of the design
team and the speed at which design problems need to be delivered, do not allow
for the luxury of investing heavily in building software interfaces which take a
long span development cycle. By distributing the workload between re-usable
smart-features doing most of the heavy weightlifting and dumb software inter-
faces which simply tap into the capabilities of the smart-features, designers are
able to map complex design problems through a short-span development cycle.
For example, in the MUS, the sphere-packing system was mapped by developing a re-usable parametric component which computed the intersection between two spheres and the master design surface. This smart-feature would take the circle which results from the intersection of two spheres and intersect it with the master design surface, providing two points as outputs to the feature. The external software interface simply connected many of these features together, managing the complex relationship network and coordinating the instantiation. The software interface did not deal with the “sphere-packing” problem directly, it simply managed the complex map of relations which the user would have had to deal with, had we developed the model through a less automatic instantiation process. By mapping the “sphere-packing” as a re-usable component and not as a computer code problem, we were able to debug the “sphere-packing” process visually using the 3D interactive viewport. The propagation logic however was mapped as an external standalone VB.net application which managed the features and the ways in which they interrelate.

In the FabPod case-study: case-study five, I examined the flexible automated design for production workflow system developed for the materialisations of the FabPod. This fabrication intent model enabled the flexible orchestration of change management in the earlier stages of design and in the stages of fabrication planning. The models developed in Grasshopper were not Digital-Mockups (DMU). The models only contained enough information for configuring basic dimensions and the topology of the design. To fabricate the FABPOD, it would be necessary to build a DMU, a more robust detailed model. This model would have to represent the NC (Numerical Control) features required to operate the NC machinery. The complexity of the model required that this model would have
to be built using an automation strategy. Although parametric modelling systems such as Digital Project™ enable designers to map design intent as flexible models, these associative modelling paradigms present a great deal of difficulty in coping with topological variations. The model was built as a hybrid model. The model used both hand-built features and computer code to represent the design intent. The hand-built features represented the components of the fabrication cells, and thus describing the model assembly. The external software interface, built using VisualBasic.net, took care of the development of the complex parametric model schema in Digital Project™. To transfer the data from Grasshopper to Digital Project™, we had to develop a custom file format as a CSV (comma separated values) file. The file described each wall assembly and each fabrication cell. With the separation of the tectonic issues from the logic issues using feature-based components, it was easy to manage the geometrical complexity required without this complexity getting entangled with the complexity of the design logic.

In the Sagrada Família Glory Façade Helical Stair Case case-study: case-study six, I examined the pre-design model of the Sagrada Família Glory Façade Helical Stair. This project comprised the development of a 3D parametric model representing the building code constraints while respecting the current construction layout of the hyperbolic stair shaft. The tool had to provide an interface in Excel to enable the team to configure the number of stair steps per landing and the number of landings in the whole stair. In order to wrap the stair system onto the ellipsoid geometry of the shaft, it was necessary to mathematically map the curved surface of the shaft onto a flat map. This mapping enabled the preservation of the angle of bearing as the stair wrapped along the ellipsoid surface.
The development of a flexible tool for configuring the design of the stair system was clear, given the pre-rational agenda. The “hybrid” modelling provided by the “External Code” strategy enabled to capture the knowledge required for the development of the step and landings as re-usable components. The flexible tool then would instantiate the step geometry and landing geometry strategically following the Excel data schedules. The flexible tool had to provide the design team with a process of integrating and configuring the isolated smart assemblies such as the rhumb lines, the step and landing solid components. The hybrid approach of combining user-features hand-built with computer algorithms which instantiate them, afforded the users opportunities for intervention throughout the various workflows embedded in the tool. Algorithms enable us to represent procedural knowledge within parametric models, however, not all aspects of a design problem need to be mapped using algorithms. When models need to represent design problems using both algorithms and manual parametric model associations, the hybrid approach enable us to represent both the algorithmic and non-algorithmic aspects of the design problem in a single model.

5.4 THE PLACE OF ALTERNATIVE BESPOKE SOLUTIONS

In Chapter 4, I presented the “Bespoke Approach” strategy. This chapter introduced three strategies for developing customised design software from source code. This strategy unlike the Standard strategy, does not rely on existing parametric design software. Instead this strategy enables the designer to build design software from a blank-slate to address a specific intended functionality which
could not be addressed by using an off-the-shelf parametric modelling software. These experimental software prototypes are developed by using the OpenCASCADE Technology™, an open-source component object library. In this section I introduced the “separation of concerns” (SoC) principle developed by Dijkstra [Dijkstra, 1982]. This principle enables the developer to abstract the use of a standard library by using a modular approach to software development. Within this chapter I discuss three software development architectures for the development of bespoke design software: The Standard Library, The Standard Library Wrapper and The Scripting Interface.

In the Dermoid: case-study seven, I examined the development of the parametric design model for the Dermoid gridshell pavilion. This section outlines the development of the Dynamic Relaxation process used to map the gridshell components over the design surface. This workshop combined many digital tools, such as Maya™, Digital Project™, Excel™, Rhinoceros™, Grasshopper™ and bespoke software tools developed using Processing and OpenCASCADE Technology™ (OCCT). The limitations of parametric software to enable the representation of complex design problems such as the reciprocal frame truss problem of the Dermoid, created the need to use sophisticated mapping techniques and the use of a Dynamic Relaxation solver for the computing of the geometry. While first versions of the tool were prototyped in 3D Studio Max™ using MaxScript the limited BREP capabilities of 3D Studio Max™ and its slow speed of executing computer code, prompted the development of a bespoke tool using the OpenCASCADE Technology™.

In the Standard Library Wrapper section, I examined the reuse of domain knowledge, and the abstraction of computer programming knowledge through
the use of a software library which encapsulates this knowledge. The Separation of Concerns principle also enables better code reuse. Whereas without it, it becomes difficult to reuse the generic code gathered from the lessons learned in the experience of using the library. The most important advantage of the SoC is the speed gained from developing quick turnover cycles in application prototype development. Meta-programming is an alternative software development technique, which enables developers to use abstraction in programming complex applications. The process of abstraction of meta-programming, is different from the process of abstraction enabled by the modularisation of software libraries. The two abstraction techniques, both the software library and the meta-programming, were demonstrated in the Responsive Acoustic Surfaces project.

In the Responsive Acoustic Surfaces: case-study eight, I examined the parametric models for the Responsive Acoustic Surfaces smartgeometry 2011 workshop. This project dealt with the acoustic properties of intersecting hyperbolic surfaces. The development of scaled wall prototypes by the participants, was carried out in bespoke software prototypes developed by each participant. The process of designing candidate designs was developed using the OpenCASCADE Technology™ by the workshop participants, using a software development kit developed specifically to address the production of bespoke design softwares. Each participant used the SDK provided to them, to develop both a 3D software and with their own software, generate 3D designs of disks to be 3d printed. However this software development process was found to be too cumbersome for many novice users. The need to install many softwares and the highly advanced configurations steps required to setup Visual Studio™ on each participants computer, developed the need to create a new strategy.
In the Scripting Interface section, I examined the third and most robust strategy for developing bespoke design softwares. This strategy also develops customised design software, however the way in which the software is developed borrows substantially from the last two strategies also addressing the development of bespoke software. In this strategy the bespoke software prototype was developed as a generic software with a scripting language. The script which initialises the application is what makes it specific to the design problem at hand. The scripting language of the application is provided with facilities for the generation of solid models which can be accessed through script commands. This scripting language streamlines the process of developing custom applications, since the end-user can create, edit and debug the script directly in the running application during run-time. This dynamic way of developing the software enables the expedition of the process of software development. Since the application integrates all the tools the user needs, the integrated development environment provides a single space for the creation of customised design applications.

5.5 SUMMARY

The strategies outlined in this thesis address the limitations outlined at the introduction to this chapter. The “Internal Code” strategy and the “External Code” strategy address the issues encountered from the use of the parametric design schema in parametric design modelling. While the Standard Library strategy, the Standard Library Wrapper strategy and the Scripting Interface strategy take a different approach by investigating an alternative design representation system to the adjacency graphs used in parametric design modelling software. Both the
standard and the bespoke strategies had to deal with the intractability of design
problems and solutions as these issues emerge not from technology, but from the
design process itself.

The strategies outlined in this thesis provide an improved design modelling
process. By providing alternative interfaces to the parametric model schema,
designers are able to iterate design variants faster. The embedding of algorithms
in the model, enable the designer to capture and edit procedural knowledge. Em-
bedded algorithms also provide a more interactive approach to parametric mod-
elling providing designers with instant feedback during crucial decision mak-
ing. These interfaces go beyond the use of parameters and relations and span
across the use of algorithms, software libraries and tool-building frameworks.
A streamlined execution of complex processes enables the re-use of knowledge
across multiple project boundaries.

As the acyclic graph of off-the-shelf parametric design software restrict the
kinds of problems designers can represent using this software paradigm. The
bespoke approaches outlined in this section present different strategies for repre-
senting design problems through the development of specialised computer soft-
ware. Bespoke softwares not only provide freedom of representation, but also
enable— due to its speed of execution— represent complex dynamic iterative
processes which are difficult to be represented through an off-the-shelf paramet-
ric design software. These strategies enabled problems to be tackled which can
be mapped through algorithms due to their localised and well formed nature
as well as computational processes borrowed from other fields which have been
well documented, such as Sphere-Packing, Dynamic Relaxation, Spring-Particle
systems, Diffusion-Limited Aggregation, Fractals, Voronoi, etc. The representa-
tional limits of the bespoke software are only those of the software developer in translating his problem into an explicit computer code problem.
Chapter 6

CONCLUSIONS

The case studies discussed in the preceding sections tested the hypothesis that the parametric modelling paradigm can be more effectively augmented through both strategy and technology on some specific cases as they have been documented in this thesis. The three intervention modes: Internal Code, External Code and Bespoke Approach, addressed challenges to flexibility within parametric design software and ways to overcome them by different means depending on the nature of the problem and the suitability of the strategy to the design problem. From a practice point of view, I have demonstrated there are solutions to the delivery of complex projects with tools available to practitioners today. The projects I have documented in this thesis are indicative of the kinds of projects that can be tackled through the strategies outlined in this thesis and the impact advanced software development practices have over the improvement of the design process. The delivery of the projects outlined in this thesis would not have been possible without the use of these advanced software development practices. However the strategies outlined in this thesis are not all flawless. Their imple-
mentation should take into consideration that while they may have worked for the delivery of the specific cases demonstrated in the thesis, there are limits in the generalisation of knowledge drawn from the case-study method. Therefore, the application of the knowledge drawn from the case-studies discussed in this thesis should be done in such a way that prevents over simplification of the complexities and idiosyncrasies of the design problem at hand.

The research methodology I have embraced, which mixes the Action Research method with the case-study method, has allowed me to capture the idiosyncrasies and complexities of each project. Furthermore this research methodology was instrumental in allowing me to organise the strategies into appropriate categories ranging from the standard towards the bespoke. This mixed-methods research methodology has also enabled me to reflect into the actions within reach of my practice and to draw appropriately from the case-studies to address the research questions outlined from the outset.

Through my research on developing software interfaces above and within standard parametric modelling software, as well the development of specialised bespoke tools for complex design resolution, I have first-hand experience in common with other researchers of the parametric modelling inflexibilities identified in the literature review and have, based on this experience, developed approaches to overcome these problems from within the parametric modelling technology or by developing independent standalone tools.

When reflecting upon the implications of my research to design modelling practice, a salient element of this research has been a paradox between the need for more automated ways of modelling with the need for more “Direct Modelling” ways of interacting with models. They are contradictory because automa-
tion aims at reducing “Direct Modelling” interaction. The primary finding of this research is that we need better ways to integrate automation in design modelling while simultaneously we need more tacit interaction through “Direct Modelling” with our models in order to achieve a greater flexibility in design modelling. We need better ways of integrating automation because some problems such as the “sphere-packing” of the Museo Soumaya, the “Dynamic Relaxation” of the Dерmoid and the panelling problem of the Sagrada Familia Hyperbolic Bridge are necessary for streamlining complex operations into simpler operations which execute in reasonable time lengths, informing decision making and improving the design modelling process. However we also need tacit interaction through “Direct Modelling” features because there are elements of the design which cannot be streamlined and turned into explicit instructions. This kind of design problems and solutions require the subtle yet precise subjectivity of a human designer interacting with the model, for example as was required in the Sagrada Familia Hyperbolic Bridge and the Sagrada Familia Passion Facade for tuning the model, as well as for the puzzle like fabrication units of the Fabpod. Finding the sweet-spot at which one should be used over the other requires professional judgement, however the strategies outlined in this thesis can provide considerations, factors and guidelines to assist designers during this process of selection.

The “Internal Code” strategy was more suitable to address design problems which required immediate feedback. For instance, the panelling of the Hyperbolic Bridge of the Sagrada Familia required the design team to interactively see the impact of design parameter changes over the geometric result. The kinds of changes they required were not simple parametric design variations as are provided by the parametric software out of the box. Changes to design variables...
triggered complex “reactions” which executed procedural operations that managed the delivery of the disruptive variations required for the project. However, while this strategy provides interactivity, disruptive variation and a lower latency between design changes and model response, having the model updating to local changes all the time could be computationally intensive and lock the computer from design changes. When designing “Embedded Software” tools such as the ones documented in the Hyperbolic Bridge and the retrospective analysis of the Passion Façade, designers must be aware of this design consideration and develop software tools which take performance and usability factors into consideration for the design of robust “Embedded Tools”. For example when designing these kinds of systems designers should take into consideration the interdependence between model performance and model response, by developing solutions which manage this trade-off accordingly.

When projects have an unconventional level of complexity, they are large and require the collaboration of a multi-user team structure, the “External Code” strategy is best to address this kind of design problem and practice constraints. For example, both the Yas Island and Museo Soumaya project had more than 11 users creating, editing and interacting with a massive parametric model. While model organisation is key for these kinds of projects, the division of labour provided by the “External Code” strategy and its loose-coupling of modelling with scripting were also major contributors to the success of these projects at delivering complexity in short spans of time at high levels of detail. This strategy also enables the break-up of work and assign it accordingly to different team members of a project. For instance in the Yas Island, various team members built each of the smart-features required for the project, while other team members wrote
computer code that instantiates those features in place. This strategy allowed
the office to manage the scale of a model commensurate to the number of team
members in a project, splitting not only the manual modelling work but also the
scripting work required. However, the separation of the team into model mak-
ers and script makers, creates a coordination concern between how the model
makers and the script makers collaborate. For both the Yas and the Mus project,
early in the project, model interfaces and software interfaces were agreed upon
using a loose-coupled process; as model interfaces and script interfaces changed,
both teams would interact minimally to update each others work, each working
apart and only coming together to test the integration of each others work in a
centralised model. While for both the Yas and Mus projects the loose-coupled
process of integration between modellers and scripters worked, this could be a
potential site for issues on projects where the collaboration between modellers
and scripters is not addressed early.

The bespoke software strategies described in chapter 4, provide designers with
the means with which to develop alternative software architectures for the rep-
resentation of complex design problems in architecture. The three successive
abstraction systems: Software Libraries, Library Wrapping and Scripting In-
terface, provide designers with the tools to build bespoke softwares effectively.
However, these strategies are best applied for the kinds of design problems where
a customised software is required only for a localised and well-defined aspect of
the design intent and not as means to represent comprehensively the totality of
the design intent. It is difficult from a practice point of view to develop robust
non-faulty software to address specifically a single design project. When ap-
proaching the development of bespoke software, designers should be aware of
this difficulty, and aim instead to develop ad-hoc software which is effective but which does not however require too much effort and development time from a software development point of view, this avoids the software development process from infringing upon the design modelling process. After all, the software development process is being used only because of the inflexibility of the parametric design software, accordingly the degree with which the software development process enters the design modelling process should be commensurate with the degree of inflexibility the off-the-shelf modelling process exhibits.

All design problems require specific design considerations for the translation of design intent into effective parametric models which improve decision making, however the inflexibilities identified in the literature review could potentially obstruct this process of decision making if models cannot adequately reflect the design intent as it is being developed by its designers. In this thesis I have provided five approaches to minimise the exposure of design problems to the inflexibilities of parametric design software and have provided eight examples of how these strategies have improved the design modelling process on real-world built projects.

The five approaches analysed in this thesis have demonstrated through the eight specific case-studies how designers can reasonably achieve disruptive variation within the flexible design environments which the parametric design paradigm fosters. This expanse in flexibility is achieved through the strategic manipulation of parametric design software, successfully overcoming the inflexibility which hinders parametric design software today.
REFERENCES


Peter Beck. Collaboration vs. integration: Implications of a knowledge-based future for the aec industry, 2005. 18


Jane Burry, Mark Burry, Martin Tamke, Mette Ramsgaard Thomsen, Phil Ayres, Alex Pena de Leon, Daniel Davis, Anders Holden Deleuran, Stig Anton Nielsen, and Jacob Riiber. Process through practice: synthesizing a novel design and production ecology. In *ACADIA 2012*, 2012a. 177


Mark Burry. *Scripting cultures: Architectural design and programming*. John Wiley and Sons, 2013. 29, 31, 47


Francisco Castano. Split separable joint apparatus and method, 1999a. 120

Francisco Castano. Contoured cladding support apparatus and method, 1999b. 120

Francisco Castano. Method of making connector hub, 2001a. 120

Francisco Castano. Free span building, 2001b. 120


Dana Cuff. *Architecture: The story of a Practice*. Mit Press, 1992. 21

Xavier De Kestelier. The work of foster and partners specialist modelling group. 2006. 22, 28


255


Andrew Maher and Jane Burry. The re-engineering project. In *Synthetic Landscapes, ACADIA 2006*. The Association for Computer-Aided Design in Architecture, 2006. 20

Andrew Maher and Mark Burry. The parametric bridge: connecting digital design techniques in architecture and engineering. In *acadia22*, 2003. 34, 45, 52, 53, 182


Michael Meredith and Mutsuro Sasaki. *From control to design: parametric/algorithmic architecture*. Actar-D, 2008. 18, 28


Christopher Tremblay. Mathematics for game developers. 2004. 184, 267

260


James Woudhuysen and Ian Abley. *Why is construction so backward?* Wiley Academy, 2004. 18


GLOSSARY

ACIS (named after Alan Grayer, Charles Lang and Ian Braid as part of Three-Space Ltd) is a geometric modelling kernel developed by Spatial Corporation a Dassault Systems subsidiary. ACIS is a descendant of the Romulus BREP system developed in 1982 by Ian Braid. ACIS is one of the most popular geometry modelling kernels used in CAD software today.

Advanced End-User Developer are users who are not necessarily trained as software developers and are however programming computers. These end-user developers require the use of end-user-development tools which enable them to create or edit software without no specific training in the software development process. These users have a higher level understanding of customising the computer for the development of bespoke software.

Application Programming Interface “is a well-defined interface that provides a specific service to other pieces of software. An API is an interface designed for developers, in much the same way that a Graphical User Interface (GUI) is an interface designed for end users” [Reddy, 2011]. API’s could be private or public.

Assemblability the process of making assembly considerations during the early
stages of the design of a product. Together with constructibility, this process also examines disassembly and other manufacturing functions that might affect the design to manufacturing cycle. This process avoids over-the-fence design practices, by overlapping design with manufacturing. This process also takes into consideration tolerances and mating of parts to determine how parts join to one another to produce an assembly or subassembly.

**Bespoke Software** refers to software developed from source code. It is software written specifically to address an intended functionality which standard softwares do not address or address poorly. These softwares are either developed by specialty consultancy practices or by the end-users directly. Bespoke softwares can address geometric or non-geometric aspects of a design. They are developed using programming languages such as Visual Basic.Net, C#, C++, Python, Javascript, Lua, PHP, Erlang, Lisp and Ruby. They can be developed as standalone applications or as plugins which extend existing applications.

**Boundary Representation** is an underlying data-structure used by geometrical kernels to describe complex solid manifolds. This representation uses two parts to describe the complexity of the solid object: topology and geometry (surfaces, curves and points). The topology describes how the object is composed from other objects. The geometry describes analytical geometry objects such as NURBS surfaces and curves, as well as planes, vectors and points. For example to describe a solid the BREP data structure specifies an infinite space limited by a list of oriented surfaces or planes. A
Face is described by an infinite analytical surface limited by a wire. A wire is described as a list of connected edges. An Edge is described as an infinite analytical curve limited by two Vertices. A vertex is a Cartesian point in space.

**C++ Binding Code** is the process by which a C++ function or component is wrapped by a scripting command. Enabling the script engine to make calls to the C++ code when the user invokes the script command. Binding code, as the name implies, enables the developer to bind native binary code with interpreted code, this step is necessary in enabling the interaction between scripting languages and programming languages concurrently.

**C++ Wrapper Code** is code which wraps existing source code, providing a well defined interface as well as packaging the code as a module for re-use within a project or across multiple project boundaries. Wrapping code is a type of code abstraction which enables modularity in programming through the segmentation of code into distinct modules, classes or functions with a specific and yet distinct purpose.

**CATIA** (Computer Aided Three-dimensional Interactive Application) is a Product Life-cycle Management software— which due to its advanced surfacing, parametric capabilities and collaboration tools — enables designers to avoid the building of full-scale mock-ups. CATIA enables instead the prebuilding and preassembling of projects using Digital-Mockups through Virtual Prototyping. It is developed in France by Dassault Systemes.

**Clash-Detection** an application or use of Parametric Modelling softwares such as CATIA/Digital Project, which enable to identify the interference be-
tween components, assemblies and sub-assemblies of a design. This tool also enables designers to measure clearances between components and assemblies. By identifying clashes and clearances within components and assemblies, designers can catch design errors during the earlier phases of design. This system is specially useful to identify interferences between disciplines such as by highlighting any possible errors which could occur from the Mechanical Electrical and Plumbing system clashing with the primary concrete or steel structure.

**Digital Mock-ups** enable designers to virtual prototype products using CAD/CAM software. A Digital Mock-up is a parametric model, modelled at a level of detail and resolution commensurate to a Full Scale Mock-up. Due to the high-level of detail and resolution of these models, Digital Mock-ups enable designers to perform downstream functions such as Assemblability Analysis, Bill of Materials, Quantification, Computer Numerical Controlled Machining and Detail Drawing Production (Shop Drawings) for more craft based manufacturing methods.

**Digital Project** is a Catia V5 PLM solution for the architecture industry. It derives most of its code from CATIA’s V5 platform, with the addition of software tools which address domain specific modelling aspects of the Architecture Engineering and Construction (AEC) industry. This parametric modelling software has been used extensively in the architecture industry to deliver complex design projects. It is developed by Gehry Technologies Inc, a technology company owned by Frank Gehry. It was used for the delivery of the Guggenheim Museum in Bilbao, the Walt Disney Concert
Hall in Los Angeles, The Music Experience Project in Seattle and The Sagrada Familia in Barcelona. It is the defacto parametric modelling software used throughout this thesis when referring to standard off-the-shelf parametric design modelling software.

**Dynamic Relaxation** is an iterative algorithm, which enables designers to relax the nodes of a mesh by using a force field. This force field can be computed using a brute-force method or an analytical mathematical model. The brute-force method relies on particle-spring systems which use Hook’s law for attraction and Coulombs law for repulsion (See “Electrical and Gravitational Force Fields” in Chapter 8 of [Tremblay, 2004, p222]). A brute-force method can also be used whereby a heuristic calculates the position of the particles in the mesh without the use of Hook’s Law or Coulomb’s Law by simplifying the spring mathematical model to a few vector operations.

**Geometric Modelling Kernel** is the key software module component used in the development of a CAD software. This software component enables CAD softwares to execute complex geometric operations such as advanced surfacing, NURBS and BREP operations. Examples of Geometry Kernels are: Convergence Geometric Modeller by Dassault Systemes, Parasolids by Siemens, ACIS by Spatial Corporation, ShapeManager by Autodesk, Granite by Parametric Technology Corporation and Open Cascade by openCASCADE SAS.

**Generative Components** is a standard off-the-shelf parametric design software developed by Bentley Systems. This software is based on Bentley’s Mi-
croStation software. Generative Components is a feature based software, it also enables to group features into transactions and provides facilities for the user to extend the system through embedded computer code or by creating .Net features as Dynamic Link Libraries¹ (DLL) assemblies. Users can also group features into user-generated-features.

**Grasshopper** is a Node Graph visual modelling interface developed on top of Mcneel’s Rhinoceros NURBS modelling software. Grasshopper enables parametric design through its node graph editing interface. Nodes in the graph are edited directly. The Grasshopper node editor is a zoomable user interface (ZUI). This enables the display of large Graphs regardless of screen size. It can be extended with embedded VB.net or C# patches or by the development of .net DLL assemblies which create custom nodes. Complex Graphs can be collapsed into a single node for easy handling of complex graphs.

**Hash-Map** in computer programming, a hash map or hash table is a data structure which can map keys to values. The hash map contains an internal hash function which it uses to compute an index in which to store a value on a conventional array. The location in the array is called a bucket. The same hash function used to store a value, is used to retrieve the correct value from a bucket. Due to this simple design, hash maps can store and retrieve data without iterating over all the buckets in an array. Due to this,

¹Dynamic Link Libraries are specific to the Windows platform. In Unix, Linux and MacOSX Dynamic Link Libraries are refered to as Shared Objects or Shared Libraries. While linux files do not need file extensions, shared object libraries are typically named with the *.so extension.

268
a hash map takes a constant time for both insertion time and retrieval time of either small or large arrays.

**Human Computation** “Human computation” has been defined by Von Ahn as “…a paradigm for utilizing human processing power to solve problems that computers cannot yet solve” [Von Ahn, 2009]. With “human computation” designers can be used as a resource of computationally intensive tasks where certain aspects of the workflow cannot be resolved by computers.

**Input Process Output** is an abstract definition of a machine, described by specifying its inputs, process and outputs. The process part consists of operations applied to the inputs (information in some form) into the outputs (new information which results from the transformation of the inputs). For example a toaster transforms the input:bread through the process:radiant heat into the output:toast. This abstract system can also be used to define abstract software machines which apply some computational process on a given input to produce a desired output [Mitchell et al., 1987, p63–64].

**Integrated Development Environment** The development of computer software from source, requires various software tools which transform source code into machine instructions. An Integrated Development Environment provides comprehensive facilities for the development of software from a single software interface. For example by enabling the code authoring, compiling and linking phases to be conducted from a single application environment. Most IDE’s also provide code editing facilities such as Code Completion, Code Snippet storing and Code Templating which enable to
increase programming productivity and to manage the complexity associated with building large scale software projects.

**Knowledgeware** is a knowledge-based engineering language specifically built for the CATIA/Digital Project platform. It enables the designer to capture engineering knowledge through geometry, features, parameters, relations, loops, rules, checks, reactions, embedded scripts (Visual Basic macro with arguments) and an embedded optimiser (simulated annealing / hill climbing, non-linear constraints).

**Meta-Programming** is a computer programming technique where the developer writes a meta-program. This meta-program is preprocessed by a preprocessor program, the output of the preprocessor becomes the input to the compiler. This programming technique allows the programmer to translate code written in a high-level macro language to code written in a lower-level programming language before it is fed to the compiler where it is subsequently converted into binary machine instructions.

**Open CASCADE Technology** open CASCADE (Computer Aided Software for Computer Aided Design and Engineering) is an open source geometry kernel developed by Open Cascade SAS. The kernel dates as far back as 1980 when Matra Datavision created the Euclid CAD system, however it wasn’t until 1993 that Matra developed the development platform CAS.CADE. With CAS.CADE Matra developed the Euclid Quantum CAD modeller (Euclid Styler and Euclid Machinist), released in 1996. The technology behind Euclid was sold in 1998 to Dassault Systemes and was made part of CATIA. In 1999 the CAS.CADE development platform was made open
source and released as Open Cascade. Open Cascade SAS the company which maintains the source code and provides services associated with it, is owned by Euriware Group, a subsidiary of Areva.

**Parasolid** is a geometric modeling kernel owned by Siemens PLM Software. Parasolid has advanced solid modelling features which enable it to create and edit complex solid models. It is used in Softwares such as : Abaqus, MicroStation, Vectorworks and NX.

**Physical Design of Software** deals with issues related to how software is stored in files, compiled, linked, and executed [Lakos, 1996]. All of these activities are affected by programming language details [Lakos, 1996]. While Small systems can afford to ignore it large systems can become unmanageable and unmaintainable even if the logical design and coding are otherwise very good [Lakos, 1996].

**Product Life Cycle Management Software** is a software which enables the stakeholders (Designers, Engineers, Contractors, Subcontractors and Facility Operators) of a project to manage the entire life-cycle of a product from design ideation, analysis, value engineering, manufacturing, Assembly, Operation Manuals and disassembly/decommissioning. The PLM software connects all stakeholders together in a central model, providing information and informed decision making throughout the entire lifetime of a project. PLM software also provide collaboration features which enables collocated and non-collocated teamwork. CATIA and Digital Project are examples of PLM software.
Programming while both scripting and programming lead to the development of software from a clear definition of a well defined computing problem. In programming the software developer must be aware of the architecture of the machine in which the software will execute (the host hardware). Programming is thus the development of computer software which maximises the limited resources of a specific computer hardware or software environment. While algorithm design, problem analysis, problem structuring and problem solving are central aspects to programming, programming is defined in this thesis as a fit for purpose design task in which the software developer must have a deeper understanding of the underlying mechanics, structures and well defined interfaces which compose the environment in which the software will be operating. Typically, programming leads to a standalone software application or as a Dynamic Link Library which can be shared by other software applications. There are various degrees of abstraction in programming languages depending on how close they are to binary machine instructions. These range from languages which use machine instructions directly (Assembly Programming: NASM) to languages which abstract the machine instructions (Compiled Languages: C/C++ and Interpreted Languages: Javascript/Python/Lisp). The closer the language is to the machine, the more the developer needs to understand the architecture of the hardware. Whereas the further the language abstraction is from the machine, the less the developer needs to worry about the architecture of the execution environment, in this case the developer only needs to focus on formal logic, basic algebra and the specific domain of application (In our case Architecture Engineering and Construction).
Pure Code is the exclusive use of computer code to represent the design intent. In a Pure Code model, the designer hardwires the design intent as computer code, in order to create geometry and organise the parametric model. The text editor of the Integrated Development Environment becomes the primary interface where the designer interacts with the Pure Code model as source code on text. Due to the difficulty to represent complex 3D tectonic problems as computer code in text, Pure Code models are challenging to use for complex puzzle-like geometry problems (see “STANDARD APPROACH” Chapter).

Pure Parametric Model refers to the use of Parametric Models exclusively through the keyboard/mouse interface. A "Pure Parametric Model" is created by using the Windows Icons Menu Pointer (WIMP) interface provided, without recourse to programming techniques. Due to the inflexibility of off-the-shelf parametric design software, Pure Parametric Models are not as flexible when handling complex design models.

QtScript the QtScript scripting engine, is a scripting engine provided by the Qt framework. It enables to expose Qt based objects (derived from the QObject) into a javascript interpreter (ECMA–262). This allows for C++ objects to interact with javascript code with ease. Both C++ code can make direct calls to javascript code, and javascript code can make direct calls to C++ code (as long as both codes are registered to interact with one another in the script engine).

Reaction is a CATIA/Digital Project specific technology, part of the knowledgeware language, which enables designers to perform more complex
actions than are permitted by the software in its off-the-shelf state. A Re-
action is a feature that reacts to events generated by a source feature. The
source feature can be any kind of feature (Geometry, Parameters, Rules,
Parts, Products, User Defined Features, etc). The reaction is linked to
the events of the source feature which trigger its execution. Examples of
events the reaction can react to are: creation, deletion, update, drag and
drop, attribute changes and parameter value changes. Reactions are stored
in the parametric model. Reactions react to changes and can trigger mod-
ifications to the model. While complex processes can be coded directly
inside a reaction, it is best to call embedded VisualBasic macros with ar-
guments from the action. The combination of Reactions with Visual Basic
macros with arguments, enable a more interactive use of CATIA/Digital
Project, providing ways to customise and circumvent the default update
mechanism and to develop complex responsive models.

**Scripting Language** “A scripting language is a programming language that is
used to manipulate, customize, and automate the facilities of an existing
system. In such systems, useful functionality is already available through a
user interface, and the scripting language is a mechanism for exposing that
functionality to program control. In this way, the existing system is said
to provide a host environment of objects and facilities, which completes
the capabilities of the scripting language. A scripting language is intended
for use by both professional and non-professional programmers.” [Ecma,
1999]

**Software Development Kit** are typically a set of software development tools
that allows for the creation of applications. They go beyond providing
API’s, but may also include other software which aids in the development
of a particular type of software. The QT Framework is an SDK, as it
provides API’s, a programing development environment and its own com-
piler.

**Software Framework** is an abstraction in which application-specific code can
be written with little user-written code. Frameworks include API’s and
SDK’s, they are integrated in such a way that developers can start creating
applications with minimal effort.

**Separation of Concerns** (SoC), is the separation of the main software into dis-
tinct sections, each tackling its own concern [Dijkstra, 1982].

**Standard Library Wrapper** (SLW) is an abstraction in programming which
enables the wrapping of an existing software library to reuse its knowl-
edge in a different context. The wrapper library enables the developer
to concentrate knowledge about the use of the standard library into one
source-code and knowledge about the application of the library into an-
other source-code.

**Sphere Packing** refers to an algorithm whereby a limited area of space is packed
tightly with as many non-overlapping spheres of a homogeneous or het-
erogeneous radius. In the Museo Soumaya, we used a special kind of
hexagonal Sphere Packing algorithm, whereby rather than using a space
filling algorithm, a surface is packed as tightly as possible by computing
the intersection of spheres on the surface. The sphere packing algorithm
which is seen on the Museo Soumaya section, relies heavily on surface-to-surface intersections as well as surface-to-curve intersections for growing the packing.
Appendix A

Appendix

A.1 SAGRADA FAMILIA HYPERBOLIC BRIDGE

This section will show the code used for instantiating the Hyperbolic Bridge Assembly. It was written as 4 reactions within CATIA/Digital Project.

**Reaction:** Build Panel Proxies  This reaction creates the directrixes of the hyperboloid system based on parameters specified in the product tree. It creates polyline placeholders which are then used by the powercopy instantiation reaction to mount panels on without having to compute the intersections again.

**Reaction:** Build InstantiatePanels  This reaction instantiates the panel User Defined Feature using the path specified in the product tree. The user can design any panel as long as the panel UDF obeys the same input rule.
**Reaction: Build_MakeBodies**  This reaction Converts the surfaces of the Panels into Closed Solids.

**Reaction: Build_SolveSplit**  This reaction Splits the solids into 4 quadrants or 2 sectors or leaves the geometry intact.

### A.1.1 Reaction: Build_Panelproxies

This reaction builds the topology grid of all panels for the bridge. It creates a placeholder polyline which is later used by the instantiatepowercopy reaction to instantiate a panel using the 4 point data embedded in the polyline. In a way the polyline is used simply to store the topology of the panel for non linear instantiation. This means instantiation does not need to go in order it can occur at random access points.

```plaintext
1 sub main(parenthb, hyperboloidsset)
2 Dim part As part
3 Set part = CATIA.ActiveDocument.part
4 Dim sel As Selection
5 Set sel = CATIA.ActiveDocument.Selection
6 ' remove the panels
7 If (hbexist(hyperboloidsset, "panels")) Then
8 set ps = gethbset(hyperboloidsset, "panels")
9 sel.clear
10 sel.add ps
11 sel.delete
12 end if
13
14 If (bodyexist(part, "hbbodies_generated")) Then
```
set ps = getbodyset(part,"hbbodies_generated")
sel.clear
sel.add ps
sel.delete
dim hsf as HybridShapeFactory
set hsf = part.HybridShapeFactory
'check if pointsystem exist and delete
if hbexist(parenthb, "directrices") then
'create point system
set directrices = gethbset(parenthb, "directrices")
set inputgeo = gethbset(directrices, "inputs")
set geo = gethbset(inputgeo, "geo")
set paramsset = gethbset(inputgeo, "parms")
set params = part.parameters.sublist(paramsset, false)
'get parameters
myangle = params.item("directrices_angle").valueasstring
myangle = left(myangle, len(myangle) - 3)
scale1 = params.item("scale1").valueasstring
scale2 = params.item("scale2").valueasstring
scale = scale2
set inputline = geo.hybridshapes.item(1)
set inputaxis = geo.hybridshapes.item(2)
set symplane = geo.hybridshapes.item(3)
set scaleplane = geo.hybridshapes.item(4)
set panelproxies = dynamichb(directrices, "panelproxies")
set uvpoints = dynamichb(directrices, "uvpoints")
set ucurves = dynamichb(directrices, "ucurves_scaled")
set vcurves = dynamichb(directrices, "vcurves_scaled")
Set ucurves = dynamichb(directrices, "ucurves")
Set vcurves = dynamichb(directrices, "vcurves")
hideobj uvpoints
hideobj ucurvess
hideobj vcurvess
hideobj ucurves
hideobj vcurves
numrots = 360 / int(myangle)
'develop directrices:
dim lastuline, lastvline
For i = 1 To numrots
Set rotlineu = hsf.AddNewRotate(inputline, inputaxis, myangle * i)
rotlineu.name = "u" & i
ucurves.AppendHybridShape rotlineu
Set uscale = hsf.AddNewHybridScaling(rotlineu, scaleplane, scale)
uscale.name = "u" & i
ucurvess.AppendHybridShape uscale
Set rotlinev = hsf.AddNewSymmetry(rotlineu, symplane)
rotlinev.name = "v" & i
vcurves.AppendHybridShape rotlinev
Set vscale = hsf.AddNewHybridScaling(rotlinev, scaleplane, scale)
vscale.name = "v" & i
vcurvess.AppendHybridShape vscale
updateobj part, rotlineu
updateobj part, rotlinev
updateobj part, uscale
updateobj part, vscale
Next
set lastuline = ucurves.hybridshapes.item(1)
set lastvline = vcurves.hybridshapes.item(1)
' do last uline
Set lastrotlineu = hsf.AddNewRotate(lastuline, inputaxis, 0)
lastrotlineu.name = "u" & Int(numrots + 1)
ucurves.AppendHybridShape lastrotlineu
Set lastuscale = hsf.AddNewHybridScaling(lastrotlineu, scaleplane, scale)
lastuscale.name = "u" & Int(numrots + 1)
ucurves.AppendHybridShape lastuscale
' do last vline
Set lastrotlinev = hsf.AddNewSymmetry(lastrotlineu, symplane)
lastrotlinev.name = "v" & Int(numrots + 1)
vcurves.AppendHybridShape lastrotlinev
Set lastvscale = hsf.AddNewHybridScaling(lastrotlinev, scaleplane, scale)
lastvscale.name = "v" & Int(numrots + 1)
vcurves.AppendHybridShape lastvscale
updateobj part, lastrotlineu
updateobj part, lastrotlinev
updateobj part, lastuscale
updateobj part, lastvscale
'develop points :
For i = 1 To ucurves.HybridShapes.count
Set uspline = ucurves.HybridShapes.Item(i)
For j = 1 To ucurves.HybridShapes.count
Set vspline = vcurves.HybridShapes.Item(j)
If isintersecting(part, uspline, vspline) Then
  Set intpoint = hsf.AddNewIntersection(uspline, vspline)
intpoint.name = uspline.name & "_" & vspline.name
part.UpdateObject intpoint
uvpoints.AppendHybridShape intpoint
End If
CATIA.StatusBar = "Buidling intersection Points" & i & ":" & j
Next
Next
'develop panels :
For i = 1 To uvpoints.HybridShapes.count
Set curp = uvpoints.HybridShapes.Item(i)
buildpolyline part, curp, uvpoints, panelproxies, numrots+1
CATIA.StatusBar = "Buidling Panel Proxies" & i & ":" & j
Next
End If
End Sub

'utility code

Function buildpolyline(part, curp, uvpoints, proxyhb, ucount)
curpname = curp.name
Dim hsf As HybridShapeFactory
Set hsf = part.HybridShapeFactory
Set westp = curp
northp = getnextciclic(curpname, ucount, 0, 1)
eastp = getnextciclic(curpname, ucount, 1, 1)
southp = getnextciclic(curpname, ucount, 1, 0)
If itemexist(uvpoints, northp) And itemexist(uvpoints, eastp)
    And itemexist(uvpoints, southp) Then
Set northp = uvpoints.HybridShapes.Item(northp)
Set eastp = uvpoints.HybridShapes.Item(eastp)
Set southp = uvpoints.HybridShapes.Item(southp)
Dim pln As HybridShapePolyline
Set pln = hsf.AddNewPolyline()
pln.InsertElement westp, 1
pln.InsertElement northp, 2
pln.InsertElement eastp, 3
pln.InsertElement southp, 4
pln.Closure = True
proxyhb.AppendHybridShape pln
pln.name = westp.name & "_" & eastp.name
updateobj part, pln
End If
End Function

Function itemexist(col, key)
On Error Resume Next
Set myitem = col.HybridShapes.Item(key)
If Err.Number <> 0 Then
itemexist = False
Else
itemexist = True
End If
End Function

Function gethbset(container, hbname)
If (hbexist(container, hbname)) Then
  Set gethbset = container.HybridBodies.Item(hbname)
Function deletehb(container, hbname)
  'check if pointsystem exist and delete
  If (hbexist(container, hbname)) Then
    Set objtodel = container.HybridBodies.Item(hbname)
    sel.Clear
    sel.Add objtodel
    sel.Delete
  End If
End Function

Function dynamichb(container, hbname)
  Dim sel As Selection
  Set sel = CATIA.ActiveDocument.Selection
  'check if pointsystem exist and delete
  If (hbexist(container, hbname)) Then
    Set objtodel = container.HybridBodies.Item(hbname)
    sel.Clear
    sel.Add objtodel
    sel.Delete
  End If
  Set objtocreate = container.HybridBodies.Add
  objtocreate.name = hbname
  Set dynamichb = objtocreate
End Function
Function hbexist(hb, name)
On Error Resume Next
Set myhb = hb.HybridBodies.Item(name)
If Err.Number <> 0 Then
  hbexist = False
Else
  hbexist = True
End If
End Function

Function itemexistcol(col, key)

A.1.2 Reaction:Build_InstantiatePanels

This reaction instantiates the panel User Defined Feature using the path specified in the product tree. The user can design any panel as long as the panel UDF obeys the same input rule.

' this reaction automatically instantiates panels on the directrices of the Hyperbolic bridge

sub main(hbset,path)
  Dim partdoc As PartDocument
  Set partdoc = CATIA.ActiveDocument
  directory = Left(partdoc.FullName, Len(partdoc.FullName) - Len(partdoc.name))
  Dim part As part
  Set part = CATIA.ActiveDocument.part
path = directory & path.valueasstring
Dim sel As Selection
Set sel = CATIA.ActiveDocument.Selection
Set hyperboloidset = hbset
Set panelset = dynamichb(hyperboloidset, "panels")
Set inthb = gethbset(hyperboloidset, "int")
Set directriceshb1 = gethbset(inthb, "directrices")
Set panelproxies1 = gethbset(directriceshb1, "panelproxies")
Set outhb = gethbset(hyperboloidset, "out")
Set directriceshb2 = gethbset(outhb, "directrices")
Set panelproxies2 = gethbset(directriceshb2, "panelproxies")
Set hb = panelset
For i = 1 To panelproxies1.HybridShapes.count
Set curpln1 = panelproxies1.HybridShapes.Item(i)
If part.IsInactive(curpln1) Then
Else
If itemexist(panelproxies2, curpln1.name) Then
Set curpln2 = panelproxies2.HybridShapes.Item(curpln1.name)
buildpanel path, part, curpln1, curpln2, panelproxies, panels
End If
End If
CATIA.StatusBar = "............building Panel" & i
Next
end sub

'utility functions

Function gethbset(container, hbname)
If (hbexist(container, hbname)) Then
Set gethbset = container.HybridBodies.Item(hbname)
End If
Function dynamichb(container, hbname)
Dim sel As Selection
Set sel = CATIA.ActiveDocument.Selection
'check if pointsystem exist and delete
If (hbexist(container, hbname)) Then
Set objtodel = container.HybridBodies.Item(hbname)
sel.Clear
sel.Add objtodel
sel.Delete
End If
Set objtocreate = container.HybridBodies.Add
objtocreate.name = hbname
Set dynamichb = objtocreate
End Function

Function buildpanel(path, part, curpln1, curpln2, panelproxies, panels)
Dim hsf As HybridShapeFactory
Set hsf = part.HybridShapeFactory
Dim p1 As Reference
Dim p2 As Reference
Dim p3 As Reference
Dim p4 As Reference
Dim p1b As Reference
Dim p2b As Reference
Dim p3b As Reference
Dim p4b As Reference
69  getpoints hsf, curpln1, p1, p2, p3, p4
70  getpoints hsf, curpln2, p1b, p2b, p3b, p4b
71  Dim InstFactory As InstanceFactory
72  Set InstFactory = part.GetCustomerFactory("InstanceFactory")
73  FixInstanceFactory InstFactory, "quadpanel", path
74  InstFactory.BeginInstantiate
75  InstFactory.PutInputData "p1", p1
76  InstFactory.PutInputData "p2", p2
77  InstFactory.PutInputData "p3", p3
78  InstFactory.PutInputData "p4", p4
79  InstFactory.PutInputData "p1b", p1b
80  InstFactory.PutInputData "p2b", p2b
81  InstFactory.PutInputData "p3b", p3b
82  InstFactory.PutInputData "p4b", p4b
83  Dim instance As HybridShapeInstance
84  Set instance = InstFactory.Instantiate
85  InstFactory.EndInstantiate
86  updateobj part, instance
87  End Function
88
89
90  Function getpoints(ByRef hsf, ByRef curpln, ByRef p1 As Reference, ByRef p2 As Reference, ByRef p3 As Reference, ByRef p4 As Reference)
91  Dim curpoly As HybridShapePolyline
92  Set curpoly = curpln
93  Dim rad As Length
94  curpoly.GetElement 1, p1, rad
95  curpoly.GetElement 2, p2, rad
96  curpoly.GetElement 3, p3, rad
curpoly.GetElement 4, p4, rad
Set p1 = hsf.GSMGetObjectFromReference(p1)
Set p2 = hsf.GSMGetObjectFromReference(p2)
Set p3 = hsf.GSMGetObjectFromReference(p3)
Set p4 = hsf.GSMGetObjectFromReference(p4)
End Function

Function hbexist(hb, name)
On Error Resume Next
Set myhb = hb.HybridBodies.Item(name)
If Err.Number <> 0 Then
hbexist = False
Else
hbexist = True
End If
End Function

Function itemexist(col, key)
On Error Resume Next
Set myitem = col.HybridShapes.Item(CStr(key))
If Err.Number <> 0 Then
itemexist = False
Else
itemexist = True
End If
End Function

Function getnext(name, uoffset, voffset)
curname = Split(name, "_")
curuval = Right(curname(0), Len(curname(0)) - 1)
curvval = Right(curname(1), Len(curname(1)) - 1)
nextu = curvval + uoffset
nextv = curvval + voffset
newname = "u" & nextu & "_" & "v" & nextv
gnext = newname
End Function

Function getnextciclic(name, count, uoffset, voffset)
curname = Split(name, "_")
curuval = Right(curname(0), Len(curname(0)) - 1)
curvval = Right(curname(1), Len(curname(1)) - 1)
nextu = curuval + uoffset
nextv = curvval + voffset
If nextu < 1 Then
nextu = count + (uoffset + 1)
ElseIf nextu > count Then
nextu = 0 + (uoffset - (count - curuval))
End If
If nextv < 1 Then
nextv = count + (voffset + 1)
ElseIf nextv > count Then
nextv = 0 + (voffset - (count - curvval))
End If
newname = "u" & nextu & "_" & "v" & nextv
gnextciclic = newname
End Function

Function updateobj(part, obj)
On Error Resume Next

part.UpdateObject obj
If Err.Number <> 0 Then
    updateobj = False
    part.Inactivate obj
Else
    updateobj = True
End If
End Function

Function isintersecting(part, obj1, obj2)
On Error Resume Next
Dim hsf As HybridShapeFactory
Set hsf = part.HybridShapeFactory
Dim intersect As HybridShapeIntersection
Set intersect = hsf.AddNewIntersection(obj1, obj2)
part.UpdateObject intersect
If Err.Number <> 0 Then
    isintersecting = False
    part.Inactivate intersect
Else
    isintersecting = True
End If
End Function

Sub FixInstanceFactory(instfac, NameOfReference, NameOfDocument)
On Error Resume Next
instfac.BeginInstanceFactory NameOfReference, NameOfDocument
If Err.Number <> 0 Then
A.1.3 Reaction: Build_MakeBodies

This reaction Converts the surfaces of the Panels into Closed Solids.

'Start reaction

sub main(part, hbset)
If hbexist(hbset, "panels") Then
    Set panels = gethbset(hbset, "panels")
    Dim hsf As ShapeFactory
    Set hsf = part.ShapeFactory
    Set mybod = dynamicbody(part, "hbbodies_generated")
    part.InWorkObject = mybod
    For i = 1 To panels.HybridShapes.count
        Set curpanel = panels.HybridShapes.Item(i)
        Set curclosedpanel = hsf.AddNewCloseSurface(curpanel)
    Next
End If
end sub

utility functions

Function dynamicbody(container, hbname)
Dim sel As Selection
Set sel = CATIA.ActiveDocument.Selection

'check if pointsystem exist and delete
If (bodyexist(container, hbname)) Then
    Set objtodel = container.Bodies.Item(hbname)
    sel.Clear
    sel.Add objtodel
    sel.Delete
End If
Set objtocreate = container.Bodies.Add
objtocreate.name = hbname
Set dynamicbody = objtocreate
End Function

Function bodyexist(hb, name)
On Error Resume Next
Set myhb = hb.Bodies.Item(name)
If Err.Number <> 0 Then
    bodyexist = False
Else
    bodyexist = True
End If
End Function

Function gethbset(container, hbname)
If (hbexist(container, hbname)) Then
    Set gethbset = container.HybridBodies.Item(hbname)
End If
End Function

Function hbexist(hb, name)
On Error Resume Next
Set myhb = hb.Bodies.Item(name)
If Err.Number <> 0 Then
    bodyexist = False
Else
    bodyexist = True
End If
End Function
A.1.4  Reaction:Build_SolveSplit

This reaction Splits the solids into 4 quadrants or 2 sectors or leaves the geometry intact.

'solve the split of the hyperboloid bridge panels into 4
    quadrants or 2 sectors or leave as one

    sub main(part, hbset, splitdir, splitkeep, splitstate)
    if Not hbexist(hbset, "panels") Then
        Exit Sub
    Else
        End If
    If Not bodyexist(part, "hbbodies_generated") Then
        Exit Sub
    Else
        End If
    set panelbody = part.bodies.item("hbbodies_generated")
    hideobj panelbody
    If Not panelbody.Shapes.count > 0 Then
        Exit Sub
    End If
Set splitsys = gethbset(hbset, "Split_System")
Set planeset = gethbset(splitsys, "final_planes")
If Not planeset.HybridShapes.count = 3 Then
    Exit Sub
End If
Set xyplane = planeset.HybridShapes.Item(1)
Set zyplane = planeset.HybridShapes.Item(2)
Set zxplane = planeset.HybridShapes.Item(3)
curplanedir = splitdir.ValueAsString
curplanekeep = splitkeep.ValueAsString
curstate = splitstate.ValueAsString
Dim firstsplitplane
Dim secondsplitplane
' make dir sel
If curplanedir = "h" Then
    Set firstsplitplane = xyplane
    Set secondsplitplane = zxplane
ElseIf curplanedir = "w" Then
    Set firstsplitplane = zyplane
    Set secondsplitplane = zxplane
ElseIf curplanedir = "l" Then
    Set firstsplitplane = zxplane
    Set secondsplitplane = xyplane
End If
' make split sel
Dim firstsplit, secondsplit, thirdsplita, thirdsplith,
    fourthsplita, fourthsplith
Set splitsysgeo = dynamichb(splitsys, "split_congeo")
If curplanekeep = "keepone" Then
    Set firstsplit = splitbody(part, panelbody, splitsysgeo,
If curstate = "second" Then
    hideobj firstsplit
    Set secondsplit = splitbody(part, panelbody, splitsysgeo,
        secondsplitplane, False)
givecolor secondsplit, 0, 255, 0
    Set thirdsplita = splitbody(part, panelbody, splitsysgeo,
        secondsplitplane, True)
givecolor thirdsplita, 0, 0, 255
End If
ElseIf curplanekeep = "keepboth" Then
    Set firstsplit = splitbody(part, panelbody, splitsysgeo,
        firstsplitplane, False)
givecolor firstsplit, 255, 0, 0
    Set secondsplit = splitbody(part, panelbody, splitsysgeo,
        firstsplitplane, True)
givecolor secondsplit, 0, 255, 0
    If curstate = "second" Then
        hideobj firstsplit
        hideobj secondsplit
        Set thirdsplita = splitbody(part, firstsplit, splitsysgeo,
            secondsplitplane, False)
givecolor thirdsplita, 0, 0, 255
        Set thirdsplitb = splitbody(part, firstsplit, splitsysgeo,
            secondsplitplane, True)
givecolor thirdsplitb, 100, 0, 255
        Set fourthsplita = splitbody(part, secondsplit,
            splitsysgeo, secondsplitplane, False)
givecolor fourthsplita, 0, 100, 100
        Set fourthsplitb = splitbody(part, secondsplit,
splitsysgeo, secondsplitplane, True)
givecolor fourthsplitb, 100, 100, 100
   End If
End If
End Sub

'utility code

Function hideobj(myobj)
   Dim sel As Selection
   Set sel = CATIA.ActiveDocument.Selection
   sel.Clear
   sel.Add myobj
   sel.VisProperties.SetShow (catVisPropertyNoShowAttr)
End Function

Function givecolor(myobj, r, g, b)
   Dim sel As Selection
   Set sel = CATIA.ActiveDocument.Selection
   sel.Clear
   sel.Add myobj
   sel.VisProperties.SetRealColor r, g, b, 0
End Function

Function splitbody(part, bodytosplit, congeo, splitplane, side)
   Dim hsf As HybridShapeFactory
   Set hsf = part.HybridShapeFactory

Dim sf As ShapeFactory
Set sf = part.ShapeFactory
Dim bodref As Reference
Set bodref = part.CreateReferenceFromObject(bodytosplit)
Dim planeref As Reference
Set planeref = part.CreateReferenceFromObject(splitplane)
Set splitcongeo = congeo
Set split1 = hsf.AddNewHybridSplit(bodref, planeref, side)
split1.name = "firstsplit"
split1.AutomaticExtrapolationMode = False
splitcongeo.AppendHybridShape split1
part.InWorkObject = split1
updateobj part, split1
Set splitbody = split1
End Function

Function dynamicbody(container, hbname)
Dim sel As Selection
Set sel = CATIA.ActiveDocument.Selection
'check if pointsystem exist and delete
If (bodyexist(container, hbname)) Then
  Set objtodel = container.Bodies.Item(hbname)
  sel.Clear
  sel.Add objtodel
  sel.Delete
End If
Set objtocreate = container.Bodies.Add
objtocreate.name = hbname
Set dynamicbody = objtocreate
End Function

Function getbodyset(container, hbname)
    If (hbexist(container, hbname)) Then
        Set gethbset = container.Bodies.Item(hbname)
    End If
End Function

Function gethbset(container, hbname)
    If (hbexist(container, hbname)) Then
        Set gethbset = container.HybridBodies.Item(hbname)
    End If
End Function

Function deletehb(container, hbname)
    'check if pointsystem exist and delete
    If (hbexist(container, hbname)) Then
        Set objtodel = container.HybridBodies.Item(hbname)
        sel.Clear
        sel.Add objtodel
        sel.Delete
    End If
End Function

Function dynamichb(container, hbname)
    Dim sel As Selection
    Set sel = CATIA.ActiveDocument.Selection
    'check if pointsystem exist and delete
If (hbexist(container, hbname)) Then
    Set objtodel = container.HybridBodies.Item(hbname)
    sel.Clear
    sel.Add objtodel
    sel.Delete
End If
Set objtocreate = container.HybridBodies.Add
objtocreate.name = hbname
Set dynamichb = objtocreate
End Function

Function hbexist(hb, name)
On Error Resume Next
Set myhb = hb.HybridBodies.Item(name)
If Err.Number <> 0 Then
    hbexist = False
Else
    hbexist = True
End If
End Function

Function itemexistcol(col, key)
On Error Resume Next
Set myitem = col.HybridShapes.Item(key)
If Err.Number <> 0 Then
    itemexistcol = False
Else
    itemexistcol = True
A.2 FABPOD

This section will show the code used for instantiating the FABPOD assembly. It was written as a Visual Basic.net application. The following code snippet is only the first 200 lines of code of each file.

A.2.1 Fabpod_GUI

The code behind the FABPOD GUI

```vbnet
Imports System.Windows.Forms
Imports INFITF
Imports ProductStructureTypeLib
```
Imports MECMOD
Imports HybridShapeTypeLib
Imports SPATypeLib
Imports System.Text.RegularExpressions
Imports PARTITF

Public Class GUI 'start of class

Public pluginparent As selective_load
Public CATIA As INFITF.Application
Public selectedproduct As Product
Public containerform
Public paneldata As New Dictionary(Of String, Dictionary(Of String, Object))
Public fabpoddata As New Dictionary(Of String, Dictionary(Of String, Object))
Public selectedhyperbola As Object
Dim paths As New Dictionary(Of String, String)
Dim driverdata As New Dictionary(Of String, Dictionary(Of String, Object))
Dim innerlist As New List(Of Object)
Dim midplaneslist As New List(Of Object)
Dim flatsrfoutput

Public Sub New(ByVal parentobj, ByRef catiaobj)
' This call is required by the Windows Form Designer.
InitializeComponent()
' Add any initialization after the InitializeComponent() call.
Me.pluginparent = parentobj

Me.CATIA = catiaobj
End Sub

Sub DressUP_GrasshopperData()()

    If Not (CATIA Is Nothing) Then
        Dim mypart As Part
        mypart = CATIA.ActiveDocument.part
        Dim sel As Selection
        sel = CATIA.ActiveDocument.Selection
        Dim t0 = Me.selectedhyperbola
        Dim hsf As HybridShapeFactory
        hsf = mypart.HybridShapeFactory
        Dim paneldatabh As HybridBody
        paneldatabh = mypart.HybridBodies.Add
        paneldatabh.Name = "PanelData"
        Dim panelshb As HybridBody
        panelshb = paneldatabh.HybridBodies.Add
        panelshb.Name = "Panels"
        Dim body As Body
        body = mypart.Bodies.Add
        body.Name = "front flaps"
        Dim panelcounter = -1
        For Each Panel As KeyValuePair(Of String, Dictionary(Of

27
panelcounter += 1
Dim curpanel = Panel.Value
Dim panelid As String = curpanel("id")
Dim posarr = curpanel("pos")
Dim dirarr = curpanel("dir")
Dim panelarray = curpanel("planes")
Dim material = curpanel("paneltype")
Dim Topcappanel
Dim botcappanel
Dim curpanelhb As HybridBody
curpanelhb = panelshb.HybridBodies.Add
If InStr(panelid, ")") Then
Dim wallarr = panelid.Split(")")
If UBound(wallarr) > 0 Then
curpanelhb.Name = "W" & wallarr(0) & ":" & "C" & wallarr(1)
Else
curpanelhb.Name = "undefined:" & panelcounter
End If
Else
curpanelhb.Name = "Panel" & Format(Val(panelid), "000")
End If
Dim panelcongeo As HybridBody
panelcongeo = curpanelhb.HybridBodies.Add
panelcongeo.Name = "Panel Positioning and orientation Data"
Dim panelorigin = hsf.AddNewPointCoord(posarr(0), posarr(1), posarr(2))
panelorigin.Name = "PanelOrigin"
panelcongeo.AppendHybridShape(panelorigin)

mypart.UpdateObject(panelorigin)

Dim paneldir = hsf.AddNewDirectionByCoord(dirarr(0),
dirarr(1), dirarr(2))

Dim paneldirinv = hsf.AddNewDirectionByCoord(-dirarr(0),
-dirarr(1), -dirarr(2))

Dim panelnormal = hsf.AddNewLinePtDir(panelorigin, paneldir,
-1000, 1000, False)

panelnormal.Name = "Panel Normal"

panelcongeo.AppendHybridShape(panelnormal)

mypart.UpdateObject(panelnormal)

Dim normalplane = hsf.AddNewPlaneNormal(panelnormal,
panelorigin)

panelcongeo.AppendHybridShape(normalplane)

normalplane.Name = "Normal Plane"

mypart.UpdateObject(normalplane)

Dim backthick = 0

Dim splitarr = material.split("_")

splitarr = splitarr(0)

If splitarr = "hard" Then
  backthick = -33 + 3
ElseIf splitarr = "soft" Then
  backthick = -33 + 12
ElseIf splitarr = "metal" Then
  backthick = -33 + 2
Else
  backthick = -33

29
End If

Dim backpanel = hsf.AddNewPlaneOffset(normalplane, backthick, False)
panelcongeo.AppendHybridShape(backpanel)
backpanel.Name = "Panel Normal"
mypart.UpdateObject(backpanel)

Dim srfcongeo As HybridBody
srfcongeo = curpanelhb.HybridBodies.Add
srfcongeo.Name = "srfcongeo"

Dim Myfactory = beginfactory("panel", Me.udfpath.Text, mypart)
Dim pnludf As HybridShapeInstance = insthyberbolicpanel (Myfactory, mypart, srfcongeo, t0, t0, panelorigin, panelnormal, 0)
pnludf.Name = curpanelhb.Name & " hyperbolic Udf surface"
Dim srf = pnludf.GetOutputFromPosition(1)
hideobj(srfcongeo)

Dim planes As HybridBody
planes = curpanelhb.HybridBodies.Add
planes.Name = "planes"
hideobj(planes)

buildplanesfromarray(panelarray, planes, mypart)

Dim Myfactory2 = beginfactory("flap", Me.pnl8udfpath.Text, mypart)
Dim innerlist As New List(Of Object)
Dim innerlistflat As New List(Of Object)
Dim outerlist As New List(Of Object)
Dim midplaneslist As New List(Of Object)
Dim outerplaneslist As New List(Of Object)
Dim flaplist As New List(Of Object)
Dim flapcongeo As HybridBody
flapcongeo = curpanelhb.HybridBodies.Add
flapcongeo.Name = "FlapSet"

Dim maxplaneval = planes.HybridShapes.Count
For k = 1 To maxplaneval
Dim pln1, pln2, pln3
If k = 1 Then
    pln1 = planes.HybridShapes.Item(maxplaneval)
    pln2 = planes.HybridShapes.Item(k)
    pln3 = planes.HybridShapes.Item(k + 1)
ElseIf k = maxplaneval Then
    pln1 = planes.HybridShapes.Item(k - 1)
    pln2 = planes.HybridShapes.Item(k)
    pln3 = planes.HybridShapes.Item(1)
Else
    pln1 = planes.HybridShapes.Item(k - 1)
    pln2 = planes.HybridShapes.Item(k)
    pln3 = planes.HybridShapes.Item(k + 1)
End If

Dim flap As HybridShapeInstance = instflap(Myfactory2, mypart, flapcongeo, srf, pln1, pln2, pln3, backpanel, panelorigin, panelnormal, normalplane, material)
If mypart.IsInactive(flap) Then
    Continue For
End If

flaplist.Add(flap)
flap.Name = curpanelhb.Name & "_" & "Flap_" & Format(k, "000")
Dim innersrf = flap.GetOutputFromPosition(14)
Dim innersrfflat = flap.GetOutputFromPosition(5)
Dim outersrf = flap.GetOutputFromPosition(6)
Dim outerplane = flap.GetOutputFromPosition(7)
Dim midplane = flap.GetOutputFromPosition(8)

innerlist.Add(innersrf)
innerlistflat.Add(innersrfflat)
outerlist.Add(outersrf)
midplaneslist.Add(midplane)
outerplaneslist.Add(outerplane)
Next

If Not (innerlist.Count = maxplaneval) Then
    Continue For
End If

Dim framejoin As HybridShapeAssemble =
    hsf.AddNewJoin(innerlist.Item(0), innerlist.Item(1))
srfcongeo.AppendHybridShape(framejoin)
framejoin.Name = curpanelhb.Name & "_" & "Frame Join"
For f = 2 To innerlist.Count - 1
    framejoin.AddElement(innerlist.Item(f))
Next
A.3 DERMOID

This section will show the code for the Dermoid Dynamic Relaxation software.

**Pattern Maker** The 3D Studio Max Maxscript code used to generate Comma Separated Values from the Polygonal Meshes.

**Relax Mesh Node** Mesh Node Visual Representation. This Class represents the visual node element which the user interacts with. It is a wrapper of the underlying mesh node abstract class.

**Dynamic Relax Mesh Node** This class represents a single point within a Dynamic Relaxation Network. It is the underlying base class where all point geometry operations are computed.

**Dynamic Relax Graph** This class represents a network of Dynamic Relax Mesh Nodes. It is the underlying base class which iterates over the network to compute the relaxation.

The following code snippet is only the first 200 lines of code of each file.

### A.3.1 Pattern Maker

MaxScript code for the PatternMaker.ms

```plaintext
1  -- Pattern Topology Builder
2  -- Select an editable poly
```
fn buildpattern savepath polysel =
{
    mysel = convertToPoly(polysel)
    out_name = savepath + "/1-Pattern.csv"
    if out_name != undefined then
    {
        out_file = createfile out_name
        num_edges = polyOp.getNumEdges mysel
        print("numedges")
        print (num_edges)
        minx = mysel.min.x
        miny = mysel.min.y
        maxx = mysel.max.x
        maxy = mysel.max.x

        --format "minxy-maxxy,%,%,%,%\n" minx miny maxx maxy num_faces
            to:out_file
        for v = 1 to num_edges do
        {
            verts = polyOp.getVertsUsingEdge mysel v
            verts = verts as array
            p1index = verts[1]
            p2index = verts[2]
            p1 = polyOp.getVert mysel p1index
            p2 = polyOp.getVert mysel p2index
            x1 = p1.x
            x2 = p2.x
            x3 = p1.x
32 y1 = p1.y
33 x2 = p2.x
34 y2 = p2.y
35 format "%g,%g,%g,%g" v x1 y1 x2 y2 to:out_file
36 format "\n" to:out_file
37 )
38 close out_file
39 --edit out_name
40 )
41 )
42
43 fn buildvertlist savepath polysel =
44 (   mysel = convertToPoly(polysel)
45 out_name = savepath + "/2-VertexList.csv"
46 if out_name != undefined then
47 (   out_file = createfile out_name
48 num_verts = polyOp.getNumVerts mysel
49 for v = 1 to num_verts do
50 (   p1 = polyOp.getVert mysel v
51 x1 = p1.x
52 y1 = p1.y
53 format "%g,%g,%g" v x1 y1 to:out_file
54 format "\n" to:out_file
55 )
56 close out_file
57 --edit out_name
58 )
59 )
fn buildedgelist savepath polysel =
{
mysel = convertToPoly(polysel)
out_name = savepath + "3-EdgeList.csv"
if out_name != undefined then
{
  out_file = createfile out_name
  num_edges = polyOp.getNumEdges mysel
  print("nunedges")
  for v = 1 to num_edges do
    verts = polyOp.getVertsUsingEdge mysel v
    verts = verts as array
    p1index = verts[1]
    p2index = verts[2]
    format "%,%,%" v p1index p2index to:out_file
    format "\n" to:out_file
  }
  close out_file
--edit out_name
}
}

fn buildvertedgelist savepath polysel =
{
mysel = convertToPoly(polysel)
out_name = savepath + "4-VertEdgeList.csv"
if out_name != undefined then
out_file = createfile out_name
num_verts = polyOp.getNumVerts mysel
for v = 1 to num_verts do
  (v
    verts = polyOp.getEdgesUsingVert mysel v
    verts = verts as array
    format "%d" v to:out_file
    for i=1 to verts.count do
      (v
        format ",%d" verts[i] to:out_file
      )
    format 
  )
close out_file
--edit out_name
)
fn buildfreeverts savepath polysel =
  (v
    mysel = convertToPoly(polysel)
    out_name = savepath + "/5-FreeVertsList.csv"
    if out_name != undefined then
      (v
        out_file = createfile out_name
        num_verts = polyOp.getNumVerts mysel
        verts = polyOp.getVertSelection mysel
        verts = verts as array
        num_edges = polyOp.getNumEdges mysel
      )
    )
)
edgebits = #{1..num_edges}
polyop.setEdgeSelection mysel edgebits
mysel.SelectBorder()
mysel(ConvertSelection #Border #Vertex
for v = 1 to verts.count do
(format "\n" verts[v] to:out_file
(format "\n" to:out_file
)
close out_file
--edit out_name
)
)

suffix = "15"
mypath = getSavePath() -- get the folder to save the list files
if mypath != undefined then
(newpath = mypath as string + "/Pattern" + suffix
newdir = makeDir newpath
buildpattern newpath $[1]
buildvertlist newpath $[1]
bUILDEdgedelist newpath $[1]
bUILDevertedgelist newpath $[1]
bUILDfreeverts newpath $[1]
)
A.3.2 Relax Mesh Node

Relaxation Mesh Node Header

```cpp
#ifndef MANTIS_RELAXMESHNODE_H
#define MANTIS_RELAXMESHNODE_H
#include <genericgraphicitem.h>
#include "qocc.h"
#include "qoccinternal.h"
#include "mantis_surfacewrapnode.h"
#include "mantis_dynamicrelaxgraph.h"

#include <User_AIS.hxx>

class QGraphicsSceneHoverEvent;

class mantis_relaxmeshnode : public QObject, public genericgraphicitem
{
    Q_OBJECT
    
    public:
    mantis_relaxmeshnode(GraphWidget *graphscene);
    ~mantis_relaxmeshnode();

    TopoDS.Shape aShape;
    Handle(User_AIS) aisShape;
    QGraphicsScene* scene;
    mantis_dynamicrelaxgraph* RelaxGraph;
    mantis_surfacewrapnode* sw1;
```
private:
protected:
void hoverEnterEvent ( QGraphicsSceneHoverEvent * event );
void hoverLeaveEvent ( QGraphicsSceneHoverEvent * event );
void mouseDoubleClickEvent ( QGraphicsSceneMouseEvent * event );

public slots:

void setsw1(mantis_surfacewrapnode* currentsrf){
    sw1 = currentsrf;
    RelaxGraph->sw1 = currentsrf;
    connect(sw1,SIGNAL(shapechanged(void)),this,SLOT(updategeometry(void)));
}

void setRelaxSpeed (int val) {
    RelaxGraph->setspeed((double)val);updategeometry();
}

void linkSlider(QSlider* Input,int val)
{
    if (val == 1) connect(Input,SIGNAL(valueChanged(int)),
                        SLOT(setRelaxSpeed(int)));
}

void updategeometry();
void updatevisuals();
signals:
void geometrychanged(void);
}
Relaxation Mesh Node Implementation

```cpp
#include "mantis_relaxmeshnode.h"
#include <QGroupBox>
#include <QFormLayout>
#include <QLabel>
#include <QLineEdit>
#include <QDoubleSpinBox>
#include <QGraphicsProxyWidget>
#include <QGraphicsSceneMouseEvent>
#include <QObject>
#include <QoccController.h>
#include <graphwidget.h>
#include <QGraphicsRectItem>
#include <QDebug>
#include <QGraphicsScene>
#include <QWheelEvent>
#include <QProgressDialog>
#include <QStringList>
#include <QMapIterator>
#include <math.h>
#include <QDesktopServices>
#include <User_AIS.hxx>
#include <QCheckBox>

static const double Pi =
  3.14159265358979323846264338327950288419717;
```
static double TwoPi = 2.0 * Pi;

mantis_relaxmeshnode::mantis_relaxmeshnode(GraphWidget *graphscene) :
genericgraphicitem(graphscene)
{
    QGroupBox *groupBox = new QGroupBox("Point coord");
    QLabel *xlabel = new QLabel("Keep Moving");
    QCheckBox *allowmovement = new QCheckBox();
    allowmovement->setCheckState(Qt::CheckState::Checked);
    QFormLayout *layout = new QFormLayout;
    layout->addRow(xlabel, allowmovement);
    groupBox->setLayout(layout);
    proxyWidget = new QGraphicsProxyWidget(this);
    proxyWidget->setFocusPolicy(Qt::StrongFocus);
    proxyWidget->setWidget(groupBox);
    setFlag(ItemIsMovable);
    setFlag(ItemSendsGeometryChanges);
    setCacheMode(DeviceCoordinateCache);
    setZValue(+200);
    setNumberOfInputs(1);

    RelaxGraph = new mantis_dynamicrelaxgraph();

    connect( RelaxGraph, SIGNAL(geometrychanged(void)),
             his, SLOT(updatevisuals(void)) );

    connect( allowmovement, SIGNAL(toggled(bool)),
             RelaxGraph, SLOT(setKeepMoving( bool ))) ;
```cpp
mantis_relaxmeshnode::~mantis_relaxmeshnode()
{
    delete proxyWidget;
}

void mantis_relaxmeshnode::updategeometry()
{
    if (!sw1) return;
    if (sw1->aShape.IsNull()) return;
    if (sw1->aShape != RelaxGraph->getTopology())
        RelaxGraph->setTopology(sw1->aShape);
}

void mantis_relaxmeshnode::hoverEnterEvent(QGraphicsSceneHoverEvent *event)
{
    QoccController* vc = graph->myMainwindow->myController;
    if (!aisShape.IsNull())
        vc->getContext()->Hilight(aisShape, true);
    vc->update();
    genericgraphicitem::hoverEnterEvent(event);
}

void mantis_relaxmeshnode::hoverLeaveEvent(QGraphicsSceneHoverEvent *event)
{
    QoccController* vc = graph->myMainwindow->myController;
    if (!aisShape.IsNull())
        vc->getContext()->Hilight(aisShape, true);
    vc->update();
    genericgraphicitem::hoverLeaveEvent(event);
}
```
if (!aisShape.IsNull())
    vc->getContext()->Unhilight(aisShape, true);
vc->update();
genericgraphicitem::hoverLeaveEvent(event);
}

void mantis_relaxmeshnode::updatevisuals()
{
if (RelaxGraph->aShape.IsNull()) return;
aShape = RelaxGraph->aShape;
QoccController* vc = graph->myMainwindow->myController;
if (aisShape.IsNull()) aisShape = new User_AIS(RelaxGraph->aShape, vc->getContext());
if (aisShape->HasPresentation())
    {
aisShape->Set(RelaxGraph->aShape);
aisShape->SetHilightMode(1);
vc->getContext()->Deactivate(aisShape);
vc->getContext()->SetDisplayMode(aisShape, 1, Standard_False);
vc->getContext()->Redisplay(aisShape);
    }
else
    {
aisShape = new User_AIS(aShape, vc->getContext());
vc->getContext()->SetColor(aisShape,
    Quantity_NameOfColor::Quantity_NOC_RED4);
vc->getContext()->SetMaterial(aisShape, Graphic3d_NameOfMaterial::Graphic3d_NOM_PEWTER);
vc->getContext()->SetDisplayMode(aisShape, 1, Standard_False);
aisShape->SetHilightMode(1);
vc->getContext()->Display(aisShape);
}
A.3.3 Dynamic Relax Mesh Node

Dynamic Relaxation Mesh Node Header

```cpp
#ifndef MANTIS_DYNAMICRELAXNODE_H
#define MANTIS_DYNAMICRELAXNODE_H
#include <gp_Pnt.hxx>
#include <QMap>
#include <TopoDS_Shape.hxx>
#include <QString>

class mantis_dynamicrelaxgraph;
class mantis_dynamicrelaxnode
{
public:

mantis_dynamicrelaxnode(mantis_dynamicrelaxgraph *parentgraph, gp_Pnt startpoint);

~mantis_dynamicrelaxnode();

double getX() {return aPoint.X();}
double getY() {return aPoint.Y();}
double getZ() {return aPoint.Z();}
```
void setX(double val){aPoint.SetX(val);} 
void setY(double val){aPoint.SetY(val);} 
void setZ(double val){aPoint.SetZ(val);} 

bool isconstrained(){return constraintstatus;} 
void setConstraintMode(bool onoff){constraintstatus = onoff;} 

void movetoidealcenter(); 
gp_Pnt getIdealCenter(); 
double getPotentialEnergy(){updatePotentialEnergy();return potentialEnergy;} 
void updatePotentialEnergy(){potentialEnergy = aPoint.Distance(getIdealCenter());} 

void addNeighbour(int NeibourName) { 
if(!nodeNeighbours.contains(NeibourName)) 
{nodeNeighbours << NeibourName; }} 

void setIndex( int indexval) { Index = indexval;} 
int getIndex() { return Index;} 

gp_Pnt getPoint() { return aPoint;} 
bool isFixed; 
void setFix(bool val){isFixed = val;} 
bool isonlowCurve; 
bool isonhighCurve; 
bool isonSurface; 
void setSupport(int sel) 
{
if (sel == 1){isonlowCurve=true;}
if (sel == 2){isonhighCurve=true;}
if (sel == 3){isonSurface=true;}

int getneighbourcount(){return nodeNeighbours.length();}

protected:
gp_Pnt aPoint;
int Index;
mantis_dynamicrelaxgraph *graph;
QList<int> nodeNeighbours;
bool constraintstatus;
gp_Pnt idealCenter;
double potentialEnergy;
private:
};
#endif // MANTIS_DYNAMICRELAXNODE_H

Dynamic Relaxation Mesh Node Implementation

#include "mantis_dynamicrelaxnode.h"
#include "mantis_dynamicrelaxgraph.h"
#include "GeomAPI_ProjectPointOnSurf.hxx"
#include "GeomAPI_ProjectPointOnCurve.hxx"
#include <qDebug>

mantis_dynamicrelaxnode::mantis_dynamicrelaxnode
(mantis_dynamicrelaxgraph *parentgraph,
 gp_Pnt startpoint)
aPoint(startpoint),
11 graph(parentgraph),
12 isFixed(false),
13 isonlowCurve(false),
14 isonhighCurve(false),
15 isonSurface(false)
16 {
17 }
18
19 mantis_dynamicrelaxnode::~mantis_dynamicrelaxnode()
20 {
21 }
22
23 void mantis_dynamicrelaxnode::movetoidealcenter()
24 {
25 gp_Pnt pointonsupport;
26 pointonsupport = idealCenter;
27 if (isonSurface)
28 {
29 TopoDS_Shape aRefShape = graph->sw1->sl->aShape;
30 if (aRefShape.ShapeType() == TopAbs_FACE &&
31 !(aRefShape.IsNull()) )
32 {
33 TopoDS_Face F = TopoDS::Face(aRefShape);
34 Handle(Geom_Surface) aSurf = BRep_Tool::Surface(F);
35 GeomAPI_ProjectPointOnSurf Proj (pointonsupport, aSurf);
36 if (Proj.NbPoints() > 0)
37 {
38 pointonsupport = Proj.NearestPoint();
39 }
40 }
if (isonlowCurve)
{
    if ( !(graph->getlowcurve().IsNull()) )
    {
        Handle(Geom_Curve) thecurve = graph->getlowcurve();
        GeomAPI_ProjectPointOnCurve Proj (pointonsupport, thecurve);
        if (Proj.NbPoints() > 0)
        {
            pointonsupport = Proj.NearestPoint();
        }
    }
}

if (isonhighCurve)
{
    if ( !(graph->gethighcurve().IsNull()) )
    {
        Handle(Geom_Curve) thecurve = graph->gethighcurve();
        GeomAPI_ProjectPointOnCurve Proj (pointonsupport, thecurve);
        if (Proj.NbPoints() > 0)
        {
            pointonsupport = Proj.NearestPoint();
        }
    }
}
gp_Pnt newpoint1(0,0,0);
newpoint1.SetX(pointonsupport.X() - aPoint.X());
newpoint1.SetY(pointonsupport.Y() - aPoint.Y());
newpoint1.SetZ(pointonsupport.Z() - aPoint.Z());
newpoint1.SetX(newpoint1.X() * (0.01*graph->relaxspeed));
newpoint1.SetY(newpoint1.Y() * (0.01 * graph->relaxspeed));
newpoint1.SetZ(newpoint1.Z() * (0.01 * graph->relaxspeed));
newpoint1.SetX(aPoint.X() + newpoint1.X());
newpoint1.SetY(aPoint.Y() + newpoint1.Y());
newpoint1.SetZ(aPoint.Z() + newpoint1.Z());
aPoint = newpoint1;
}

gp_Pnt mantis_dynamicrelaxnode::getIdealCenter()
{
    double localx = 0, localy = 0, localz = 0;
    gp_Pnt newcenter;
    QListIterator<int> NeighbourIT(nodeNeighbours);
    while (NeighbourIT.hasNext())
    {
        int neighbourKey = NeighbourIT.next();
        if (graph->nodeMap.contains(neighbourKey))
        {
            localx += graph->nodeMap[neighbourKey]->getX();
            localy += graph->nodeMap[neighbourKey]->getY();
            localz += graph->nodeMap[neighbourKey]->getZ();
        }
    }
    if (nodeNeighbours.size() > 0)
    {
        localx /= nodeNeighbours.length();
        localy /= nodeNeighbours.length();
        localz /= nodeNeighbours.length();
        newcenter = gp_Pnt(localx, localy, localz);
        idealCenter = newcenter;
    }
A.3.4 Dynamic Relax Mesh Graph

Dynamic Relaxation Mesh Graph Header

```cpp
#ifndef MANTIS_DYNAMICRELAXGRAPH_H
#define MANTIS_DYNAMICRELAXGRAPH_H
#include <mantis_dynamicrelaxnode.h>
#include <mantis_surfacewrapnode.h>
#include <QObject>
#include <QString>
#include <TopoDS_Edge.hxx>
#include <QThread>
#include <mantis_shapefactory.h>

class mantis_dynamicrelaxgraph : public QThread , public QObject
{
    Q_OBJECT
public:
    struct indexedpoint
    {
```
QString XYZ;
gp_Pnt p1;
int index;
};

struct EdgePnt
{
QString p1;
QString p2;
 gp_Pnt gp1;
 gp_Pnt gp2;
TopoDS_Edge Edge;
QString Edgename;
};
mantis_dynamicrelaxgraph();
~mantis_dynamicrelaxgraph();

QMap<int, mantis_dynamicrelaxnode*> nodeMap;
QMap<QString,QList<QString>> NodeNeighborhood;
QMap<int,QList<QList<int>>> EdgeNeighborhood;
QMap<int,indexedpoint> VertexIndexMap;
QMap<int,gp_Pnt> VertexIndexMap2;
QList<EdgePnt> EdgeIndexMap;
QMap<QString,EdgePnt> EdgePointMap;
QMap<gp_Pnt,EdgePnt> EdgePointMap2;
mantis_surfacewrapnode *sw1;
QList<QString> uniquepointkeys;
QMap<int,TopoDS_Shape> EdgeShapeList;
QMap<QString, TopoDS_Shape> *supports;
```cpp
TopoDS_Shape aShape;
void updateShape();

const TopoDS_Shape& getTopology() { return Topology; }
void setTopology(const TopoDS_Shape &inshape){
  if(!inshape.IsNull())
  {
    Topology = inshape;
    buildTopology4(inshape);
  }
}

void run();
void addNode(int NodeName, gp_Pnt Coordinate);
void clearNodes(){nodeMap.clear();}
void clearSupports(){supports->clear();}
bool somethingmoved;
double convergence;
double relaxspeed;
bool keepmoving;
void setspeed(double val){relaxspeed = val;}
Handle(Geom_Curve) getLowCurve() { return lowercurve; }
Handle(Geom_Curve) getHighCurve() { return uppercurve; }

mantis_shapefactory* hsf;

public slots:
void setKeepMoving(bool val);

private:
protected:
TopoDS_Shape Topology;
```
Dynamic Relaxation Mesh Graph Implementation

```cpp
#include "mantis_dynamicrelaxgraph.h"
#include <QMapIterator>
#include <QMap>
#include <Precision.hxx>
#include "TopTools_IndexedMapOfShape.hxx"
#include "TopTools_IndexedDataMapOfShapeListOfShape.hxx"
#include "TopExp.hxx"
#include "TopTools_ListOfShape.hxx"
#include <TopTools_ListIteratorOfListOfShape.hxx>
#include <gp_Pnt.hxx>
#include <BRep_Tool.hxx>
#include <TopoDS.hxx>
#include <QDebug>
#include <TopExp_Explorer.hxx>
#include <TopoDS_Edge.hxx>
#include <TopoDS_Vertex.hxx>
#include <QMultiMap>
#include <BRepBuilderAPI_MakeEdge.hxx>
#include <TopoDS_Compound.hxx>
#include <BRep_Builder.hxx>
#include <BRepAlgoAPI_Fuse.hxx>
#include <QTime>
```
#include <QStringList>
#include <ShapeAnalysis.hxx>
#include <QCoreApplication>

static const double Pi = 355/113;

static double TwoPi = 2.0 * Pi;

inline bool operator<(const gp_Pnt &e1, const gp_Pnt &e2)
{
    if (e1.Distance(gp_Pnt(0,0,0)) != e2.Distance(gp_Pnt(0,0,0)))
        return -1;
    return 0;
}

mantis_dynamicrelaxgraph::mantis_dynamicrelaxgraph()
{
    hsf = new mantis_shapefactory();
}

mantis_dynamicrelaxgraph::~mantis_dynamicrelaxgraph()
{
}

void mantis_dynamicrelaxgraph::setKeepMoving(bool val)
{
    keepmoving = val;
}

void mantis_dynamicrelaxgraph::run()
{
    somethingmoved = true;
}
int updatecount = 0;
while (somethingmoved )
{
    if(keepmoving == false) return;
    somethingmoved = false;
    QMapIterator<int, mantis_dynamicrelaxnode*> iT(nodeMap);
    int nodeupdatecount = 0;
    while (iT.hasNext())
    {
        nodeupdatecount +=1;
        mantis_dynamicrelaxnode* currentNode = iT.next().value();
        if(!currentNode->isFixed)
        {
            if (currentNode->getPotentialEnergy() > Precision::Confusion())
            {
                somethingmoved = true;
                currentNode->movetoidealcenter();
            }
        }
        if (nodeupdatecount == 8)
        {
            QCoreApplication::instance()->processEvents();nodeupdatecount =0;
        }
        updatecount +=1;
    }
    if (updatecount > 10) {
        updateshape();updatecount =0;
    }
} // end of node update
} // end of something moved
exec();

} // end of function

void mantis_dynamicrelaxgraph::addNode(int NodeName, gp_Pnt Coordinate)
{
nodeMap.insert(NodeName, new mantis_dynamicrelaxnode(this, Coordinate));
}

void mantis_dynamicrelaxgraph::updateshape()
{
TopoDS_Compound aRes;
BRep_Builder aBuilder;
aBuilder.MakeCompound (aRes);
EdgeShapeList.clear();
QMapIterator<int,QStringList> edgeIT(sw1->Edgelist);
while (edgeIT.hasNext())
{
QStringList curedge = edgeIT.next().value();
int curedgeindex = edgeIT.key();
int plindex = curedge.at(0).toInt();
int p2index = curedge.at(1).toInt();
if (nodeMap.contains(plindex) && nodeMap.contains(p2index))
{
    gp_Pnt gP1 = nodeMap[plindex]->getPoint();
gp_Pnt gP2 = nodeMap[p2index]->getPoint();
if ( (gP1.Distance(gP2) > 0.1))
{
try
{ 
  TopoDS_Shape edge1 = BRepBuilderAPI_MakeEdge(gP1, gP2).Shape();
  aBuilder.Add(aRes, edge1);
  EdgeShapeList.insert(curedgeindex, edge1);
}

catch(...)
{
}

if (aRes.IsNull()) return;
this->aShape = aRes;
emit geometrychanged();

A.4 SCRIPTING INTERFACE

This section will show 3 key classes of the OpenShapeFactory system created for this thesis. The full source code for this project can be downloaded from: https://code.google.com/p/openshapefactory/

The project used or extended the following Libraries:

1. OpenCascade 6.7.0

2. Qt 4.7

3. QScintilla

4. NVoronoi
Sample Scripts  These sample scripts are two examples of using the scripting system.

Script Widget  The script widget is the text editor where the user writes the code and where the user evaluates the code.

HybridShape Interface  This class represents the binding code which maps the C++ class Shapefactory into QtScript functions which can be called from the Script Widget.

ShapeFactory  Wrapper Class which wraps open CASCADE operations.

The following code snippet is only the first 200 lines of code of each file.

### A.4.1 Sample Scripts

Kilian Roof Sample. This sample is based on the Axel Kilian’s Roof Sample which was given with Generative Components as a Sample of a parametric model. This Sample builds a series of bspline arcs, lofts a surface between them, and panelises the resulting surface with a paneling function.

```python
1 minval = -300
2 maxval = 300
3 x1 = getval(0,minval,maxval)
4 y1 = getval(1,minval,maxval)
5 z1 = getval(2,0,maxval)
6 x2 = getval(3,minval,maxval)
```
y2 = getval(4,minval,maxval)

z2 = getval(5,0,maxval)

x3 = getval(6,minval,maxval)

y3 = getval(7,minval,maxval)

z3 = getval(8,0,maxval)

x4 = getval(9,minval,maxval)

y4 = getval(10,minval,maxval)

z4 = getval(11,0,maxval)

p1 = makepoint(x1,y1,0)

p2 = makepoint(x2,y2,0)

p3 = makepoint(x3,y3,0)

p4 = makepoint(x4,y4,0)

splineset = makepointlist(p1,p2,p3,p4)

spline1 = makebspline(splineset)

vis(spline1)

crs1 = crossection(spline1,0,z1,z1)

crs2 = crossection(spline1,0.25,z2,z2)

crs3 = crossection(spline1,0.75,z3,z3)

crs4 = crossection(spline1,1,z4,z4)

loftlist= makeshapelist(crs1,crs2,crs3,crs4)

loft1 = makeloft(loftlist)

panelize(loft1,10,10,mypanel)

function crossection(spline,percent,width,height)
{
    ratio = percent
    width = (width/2) +1
    height = height + 1
    up = makevector(0,0,1)
    p1 = makepointoncurve(spline1,ratio)
Kilian Roof Sample with Iges Import. This sample imports a surface and panelises it with a paneling function.

```plaintext
path = "c://thesurface.iges"
surface = importigs(path) //import igs surface
panelize(surface,10,10,mypanel) //utility function
function mypanel(p1,p2,p3,p4) {
    diag1 = makelineptpt(p1,p3)
    diag2 = makelineptpt(p2,p4)
    midpl = makepointoncurve(diag1,0.5)
    vis(diag1)
    vis(diag2)
}
```
diag2 = makelineptpt(p2,p4)
vis(diag1);vis(diag2);
}

Build Grid Shell geometry from comma separated values (CSV) files. This sample loads the node coordinates of a gridshell stores them in a hashmap and builds beam swept surfaces from a CSV file indicating the start node index and end node index.

filename1 = "H:\DEV\NodeCoordinates.csv" // node_index,x,y,z
filename2 = "H:\DEV\EdgeTopology.csv" // edge_index,
node_index_1, node_index_2

csv1 = readcsv(filename1)
csv2 = readcsv(filename2)
rowcount1 = getcsvrowcount(csv1)
rowcount2 = getcsvrowcount(csv2)
mypmap = makemap()

//create points from csv
for (i = 1; i < rowcount1; i++) {
    pointname = getcsvrow(csv1,i,1)
    x = getcsvrow(csv1,i,2)
    y = getcsvrow(csv1,i,3)
    z = getcsvrow(csv1,i,4)

    p1 = makepoint(x,y,z)
    mapinsert(mypmap,pointname,p1)
    vis(p1)
}
//create edges from csv map
for (i = 1; i < rowcount2; i++) {
    edgename = getcsvrow(csv2,i,1)
    name1 = getcsvrow(csv2,i,2)
    name2 = getcsvrow(csv2,i,3)

    if (mapcontains(mypmap,name1) &&
        mapcontains(mypmap,name2) ) {
        p1 = mapgetvalue(mypmap,name1)
        p2 = mapgetvalue(mypmap,name2)
        l1 = makelineptpt(p1,p2)
        makebeam(l1)
        vis(l1)
    }
}

function makebeam(arc)
{
    p1 = makepointoncurve(arc,0)
    v1 = makevectortangenttocurve(arc,0)
    mylen = getcurvelength(arc)
    c1 = makecircle(p1,v1,mylen/20)
    sweep = makesweep(arc,c1)
    vis(sweep)
A.4.2 Script Widget

Script Widget Code

```cpp
#include "scriptwidget.h"
#include "shapefactory.h"
#include "ui.h"
#include "User_AIS.hxx"
#include "SGMGUI_COMMON.h"
#include <QBoxLayout>
#include "AIS_Gauss.hxx"
#include "parametricsfordummies.h"
#include <QScriptable>
#include <QFile>
#include <QTextStream>
#include <Qsci/qsciscintilla.h>
#include "Qsci/qscilexerjavascript.h"
#include <qsciapis.h>
#include <QFont>
#include <QFontMetrics>
#include <QTextBlock>
#include <QAbstractItemModel>
#include <QStringListModel>
#include <QSettings>
#include <QShortcut>

scriptwidget::scriptwidget(QWidget *parent)
   : QWidget(parent)
```
ui.setupUi(this);
ui.tab_2->setLayout(ui.thetextlayout);
ui.tab->setLayout(ui.slider_tab_layout);
this->setLayout(ui.verticalLayout);
connect(ui.evalbutton, SIGNAL(pressed()), this,
    SLOT(evaluatetext()));
connect( appui::getInstance()->getWindowController() ,
    SIGNAL(clickEvent(occviewport*, QMouseEvent*)),
    this, SLOT (clickEvent(occviewport*, QMouseEvent*)) );
connect( appui::getInstance()->getWindowController(),
    SIGNAL(selectionChanged()) ,
    this, SLOT (onSelectionChanged()) );
myeditor = new QScriptEdit(0); // addtexteditor found it in QT
debugger
this->seteditor();
QString folder = QCoreApplication::applicationDirPath();
readcodefile();
hsfapi = new HsfScriptingInterface() ;
hsfapi->setparentwidget(this);
myengine.importExtension("qt.core");
myengine.importExtension("qt.gui");
QScriptValue myglobal = myengine.newQObject(hsfapi);
myglobal.setPrototype(myengine.globalObject());
myengine.setGlobalObject(myglobal);
QScriptValue frontend = myengine.newQObject(ui.tab);
myengine.globalObject().setProperty("gui", frontend);

int childcount = ui.sliderset->count();
for(int i=0;i<childcount;i++){
    QSlider* widgetitem =
        qobject_cast<QSlider*>(ui.sliderset->itemAt(i)->widget());

    if(widgetitem){
        QString objname = widgetitem->objectName();
        QString obj2name = objname + QString("txt");

        connect(widgetitem,SIGNAL(sliderMoved(int)),this,SLOT(evaluatetext()));
        frontend.setProperty(objname,myengine.newQObject(widgetitem));
    }
}

void scriptwidget::makeinteractive_text(bool value)
{
    if (value)
    {
        connect(myeditor, SIGNAL(textChanged()), this,
               SLOT(evaluatetext()));
    } else {
        disconnect(myeditor, SIGNAL(textChanged()), this,
                   SLOT(evaluatetext()));
    }
}
void scriptwidget::on3dSelectionChanged()
{
    evaluatetext();
}

void scriptwidget::moveEvent( occviewport* widget,
    QMouseEvent* e )
{
    hsfapi->setmousepos(widget->getPoint());
    evaluatetext();
}

void scriptwidget::clickEvent( occviewport* widget,
    QMouseEvent* e )
{
    hsfapi->setmousepos(widget->getPoint());
    evaluatetext();
    QString posmsg("widgetpos:" + e->pos().x() + tr("," ) + 
                  e->pos().y());
    hsfapi->print(QScriptValue(posmsg));
}

QAbstractItemModel* scriptwidget::modelFromFile(const QString&
    fileName,QCompleter* completer)
{
    QFile file(fileName);
    if (!file.open(QFile::ReadOnly))
        return new QStringListModel(completer);
QApplication::setOverrideCursor(QCursor(Qt::WaitCursor));
QStringList words;
while (!file.atEnd()) {
    QByteArray line = file.readLine();
    if (!line.isEmpty())
        words << line.trimmed();
}
QApplication::restoreOverrideCursor();
return new QStringListModel(words, completer);

void scriptwidget::seteditor()
{
    textEdit = new QsciScintilla();
    QShortcut* shortcut_ctrl_space = new QShortcut(QKeySequence("Ctrl+Space"),textEdit);
    connect(shortcut_ctrl_space, SIGNAL(activated()),
            textEdit,SLOT(autoCompleteFromAll()));
    QsciLexerJavaScript* jscript = new QsciLexerJavaScript(textEdit);
    QFont font;
    font.setFamily("arial");
    font.setFixedPitch(true);
    font.setPointSize(12);
    font.setWeight(300);
    font.setStyleStrategy(QFont::StyleStrategy::PreferQuality);
    QFontMetrics fm = QFontMetrics(font);
    jscript->setFont(font);
    jscript->setColor(QColor("#BDAF9D"),QsciLexerCPP::Default);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::Default);
jscript->setDefaultColor(QColor("#BDAF9D"));
jscript->setDefaultPaper(QColor("#2A211C"));
jscript->setColor(QColor("#FF3A83"),QsciLexerCPP::Number);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::Number);
jscript->setColor(QColor("#37A3ED"),QsciLexerCPP::Keyword);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::Keyword);
jscript->setColor(QColor("#BDAE9D"),QsciLexerCPP::Identifier);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::Identifier);
jscript->setColor(QColor("#00FF40"),QsciLexerCPP::DoubleQuotedString);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::DoubleQuotedString);
jscript->setColor(QColor("#80FF00"),QsciLexerCPP::SingleQuotedString);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::SingleQuotedString);
jscript->setColor(QColor("#666666"),QsciLexerCPP::Comment);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::Comment);
jscript->setColor(QColor("#666666"),QsciLexerCPP::CommentLine);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::CommentLine);
jscript->setColor(QColor("#FFFF80"),QsciLexerCPP::CommentDoc);
jscript->setPaper(QColor("#2A211C"),QsciLexerCPP::CommentDoc);
textEdit->setMarginsForegroundColor(QColor("#E5C138"));
textEdit->setMarginsBackgroundColor(QColor("#2A211C"));
textEdit->setAutoFillBackground(true);
textEdit->setCaretLineVisible(true);
textEdit->setCaretWidth(5);
textEdit->setCaretForegroundColor(QColor("#E5C138"));
textEdit->setCaretLineBackgroundColor(QColor("#2A211C"));
textEdit->setFoldMarginColors(QColor("#E5C138"),QColor("#2A211C"));
textEdit->setMarginWidth(0, fm.width("0000" ));
setTextStyle->setMarginLineNumbers(0, true);
setTextStyle->setEdgeMode(QsciScintilla::EdgeLine);
setTextStyle->setEdgeColumn(0);
setTextStyle->setEdgeColor(QColor("green"));
setTextStyle->setLexer(jscript);
setTextStyle->setFolding(QsciScintilla::FoldStyle::BoxedTreeFoldStyle, 2);
setTextStyle->setIndentationGuides(true);
setTextStyle->setAutoCompletionSource(QsciScintilla::AutoCompletionSource::AcsAll);
setTextStyle->autoCompleteFromDocument();
setTextStyle->setAutoIndent(true);
setTextStyle->setBraceMatching(QsciScintilla::BraceMatch::SloppyBraceMatch);
setTextStyle->setCallTipsStyle(QsciScintilla::CallTipsStyle::CallTipsContext);
setTextStyle->show();
setTextStyle->setAutoCompletionShowSingle(false);
setTextStyle->autoCompleteFromAll();
setTextStyle->autoCompletionFillupsEnabled();
setTextStyle->autoCompletionReplaceWord();
setTextStyle->annotationDisplay();
setTextStyle->zoomIn(2);
ui.thetextlayout->addWidget(textEdit);
setTextStyle->setCallTipsStyle(QsciScintilla::CallTipsContext);
}

QString scriptwidget::gettextbyline (int linenumber)
{
    if (myeditor->isVisible())
        { 
QString lineat =
            myeditor->document()->findBlockByLineNumber(linenumber).text();

    }
A.4.3 HybridShape Interface

The HybridShape Interface binding code which maps the shapefactory class into QtScript commands.

```cpp
#include "HsfScriptingInterface.h"
#include "shapefactory.h"
#include "ui.h"
#include "User_AIS.hxx"
#include "SGMGUI_COMMON.h"
#include "QHBoxLayout"
#include "AIS_Gauss.hxx"
#include "QFileDialog"
#include "scriptwidget.h"
#include "QRadioButton"
#include "QoccInputOutput.h"
#include "Prs3d_Presentation.hxx"
#include "Qsci/qsciscintilla.h"
```
#include <QLabel>
#include <BRepClass3d_SolidClassifier.hxx>
#include <Handle_MeshVS_Drawer.hxx>
#include <MeshVS_Mesh.hxx>
#include <MeshVS_DrawerAttribute.hxx>
#include <MeshVS_MeshPrsBuilder.hxx>
#include <MeshVS_TextPrsBuilder.hxx>
#include <MeshVS_Drawer.hxx>
#include <XSDRAWSTLVRML_DataSource.hxx>
#include <Graphic3d_MaterialAspect.hxx>
#include <Handle_Prs3d_BasicAspect.hxx>
#include <Graphic3d_MaterialAspect.hxx>
#include <Quantity_Color.hxx>

// voronoi needs
#include <iostream>
#include <math.h>
#include <algorithm>
#include <time.h>
#include "Voronoi.h"
#include "VPoint.h"
// end voronoi

#include "QSqlQuery"
#include "QSqlRecord"
#include "gradients.h"

class MeshVS_Drawer;
// this is for the use of the KMlocal library for kmeans clustering
// experiment starting on august 30 2011
#include <cstdlib> // C standard includes
#include <iostream> // C++ I/O
#include <string> // C++ strings
#include "KMlocal.h" // k-means algorithms
using namespace std; // make std:: available
// execution parameters (see KMterm.h and KMlocal.h)
KMterm term(100, 0, 0, 0, // run for 100 stages
0.10, 0.10, 3, // other typical parameter values
0.50, 10, 0.95);

QScriptValue HsfScriptingInterface::importigs()
{
    TopoDS_Shape curimport;
    if(context()->argumentCount() == 1)
    {
        QString filename = context()->argument(0).toString();
        QFileInfo curfile(filename);
        Handle(TopTools_HSequenceOfShape) importedsequence =
            io_man->importIGES(filename);
        if (importedsequence->Length() > 0)
        {
            curimport = importedsequence->Value(1);
            LastImportShape = curimport;
        } // end of check something inside file
        LastImportFilename = filename;
    } // end of check filename
} // end of check argument
return engine()->toScriptValue(curimport);

QScriptValue HsfScriptingInterface::panelize()
{
    if(context()->argumentCount() == 4) {
        TopoDS_Shape surface1 =
            context()->argument(0).toVariant().value<TopoDS_Shape>();

        int x = context()->argument(1).toNumber();
        int y = context()->argument(2).toNumber();
        QScriptValue panelfunc = context()->argument(3);
        if (panelfunc.isFunction()) {
            TopoDS_Compound folder;
            BRep_Builder B;
            B.MakeCompound(folder);
            int viscount = 0;
            QMap<QString, QVariant> plist =
                HSF::BuildPointGridonSrf(surface1, x, y);
            QMapIterator<QString, QVariant> i(plist);
            while (i.hasNext()) {
                i.next();
                QString currentname = i.key();
                QString epn = HSF::GetNextUvName(currentname, 1, 0);
                QString sepn = HSF::GetNextUvName(currentname, 1, 1);
                QString swpn = HSF::GetNextUvName(currentname, 0, 1);
                if (plist.contains(epn) && plist.contains(sepn) &&
                    plist.contains(swpn)) {
                    gp_Ax1 plval = i.value().value<gp_Ax1>();
                }
            }
        }
    }
}
gp_Ax1 p2val = plist.value(epn).value<gp_Ax1>();
gp_Ax1 p3val = plist.value(sepn).value<gp_Ax1>();
gp_Ax1 p4val = plist.value(swpn).value<gp_Ax1>();
gp_Pnt p1 = p1val.Location();
gp_Pnt p2 = p2val.Location();
gp_Pnt p3 = p3val.Location();
gp_Pnt p4 = p4val.Location();
gp_Vec v1 = p1val.Direction();
gp_Vec v2 = p2val.Direction();
gp_Vec v3 = p3val.Direction();
gp_Vec v4 = p4val.Direction();
QScriptValue pp1 = engine()->toScriptValue(hsf::AddNewPoint(p1));
QScriptValue pp2 = engine()->toScriptValue(hsf::AddNewPoint(p2));
QScriptValue pp3 = engine()->toScriptValue(hsf::AddNewPoint(p3));
QScriptValue pp4 = engine()->toScriptValue(hsf::AddNewPoint(p4));
QScriptValueList args;
args << pp1 << pp2 << pp3 << pp4;
QScriptValue result;
QScriptValue thepanel = panelfunc.call(thisObject(), args);
QString type = thepanel.toVariant().typeName();
QList<TopoDS_Shape> panelshape =
    thepanel.toVariant().value<QList<TopoDS_Shape>>();
if (!panelshape.isEmpty()) {
    for(int j=0; j<panelshape.count(); j++)
    {
        B.Add(folder, panelshape.at(j));
viscount++;  

if (viscount>0){  
TopoDS_Shape resultshape = folder;  
return engine()->toScriptValue(resultshape);  
}  

else { return engine()->toScriptValue(false);  
}  

void kmeanscluster(int familycount,QList<panelinstance> &mypanels)  

test =  

k = familycount;  
int dim = mypanels.at(0).parameters.count();  
int nPts = mypanels.count();  
KMdata dataPts(dim, nPts);  
KMpointArray pa = dataPts.getPts();  
for (int i = 0; i < nPts; i++) {  
panelinstance curpanel = mypanels.at(i);  
for (int d = 0; d < dim; d++) {  
double curparm = curpanel.parameters.at(d);  
pa[i][d] = curparm;  
}  
}  

myarr = dataPts.getPts();

76
KMpoint curval = myarr[1,1];
dataPts.buildKcTree(); // build filtering structure
KMfilterCenters ctrs(k, dataPts); // allocate centers
// run the algorithm
KMlocalLloyds kmAlg(ctrs, term); // repeated Lloyd’s
ctrs = kmAlg.execute(); // execute
// print number of stages
cout << "Number of stages: " << kmAlg.getTotalStages() << "\n";
// print average distortion
cout << "Average distortion: " << ctrs.getDist()/nPts << "\n";
ctrs.print(); // print final centers
KMctrIdxArray closeCtr = new KMctrIdx[dataPts.getNPts()];
double* sqDist = new double[dataPts.getNPts()];
ctrs.getAssignments(closeCtr, sqDist);
for (int i = 0; i < dataPts.getNPts(); i++) {
    int pointindex = i;
    //panelinstance &curpanel = ;
    mypanels[i].panelindex = pointindex;
    int closestcenterindex = closeCtr[i];
    double distancetocenter = sqDist[i];
    mypanels[i].centerindex = closestcenterindex;
    mypanels[i].distancetocenter = distancetocenter;
}
delete [] closeCtr;
delete [] sqDist;
HsfScriptingInterface::HsfScriptingInterface()
{

Handle_AIS_InteractiveContext ic =
    appui::getInstance()->getWindowContext();
viscount = 0;
setuprunonce();
io_man = new QoccInputOutput();
needstofitall = false;
filewatch = new QFileSystemWatcher( this ); // pass this
    (QObject) as parent
objectcache = new QMap<QString,QVariant> ;
}
HsfScriptingInterface::~HsfScriptingInterface()
{

}

A.4.4 ShapeFactory

The Shape Factory class. This class wraps openCASCADE operations into a
modular library for geometry creation with distinct functions.

#include <shapefactory.h>
#include <stdio.h>
#include <time.h>
#include <ui.h>
#include <Poly_Polygon3D.hxx>
#include <BOPAlgo_BOP.hxx>
#include <BOPAlgo_PaveFiller.hxx>
#include <ShapeConstruct_ProjectCurveOnSurface.hxx>

Q_DECLARE_METATYPE(gp_Ax1)

static const double Pi = 3.1415926535897932384626;
static double TwoPi = 2.0 * Pi;
static Standard_Boolean fixParam(Standard_Real& theParam);

gp_Pnt globalorigin; // glocal point used in the sorting function

gp_Pnt globalhbcenter; // center of current hb

gp_Pnt globalrefedgemidp; // center of current hb

gp_Pln globalhbplane; // plane of hyperbola

//distance sorting function for using with QMap
bool sortbydistance(const gp_Pnt p1, const gp_Pnt p2)
{
  double dis1 = globalorigin.Distance(p1);
  double dis2 = globalorigin.Distance(p2);
  return dis1 < dis2;
}

bool sortbyangle(const gp_Pnt p1, const gp_Pnt p2)
{
  gp_Pnt2d p12d =
      HSF::Get2dPntonSurfacefromPoint(globalhbplane,p1);
  gp_Pnt2d p22d =
      HSF::Get2dPntonSurfacefromPoint(globalhbplane,p2);
  gp_Lin
      refline(globalhbcenter,gp_Vec(globalhbcenter,globalrefedgemidp));
  gp_Lin 11(globalorigin,gp_Vec(globalorigin,p1));
  gp_Lin 12(globalorigin,gp_Vec(globalorigin,p1));
double angle1 = refline.Angle(l1);
double angle2 = refline.Angle(l2);
if (p12d.X() < 0 && p12d.Y() < 0 ) angle1 += 180;
if (p22d.X() < 0 && p22d.Y() < 0) angle2 += 180;
return angle1 < angle2;
}

TopoDS.Shape HSF::BooleanGlue(QList<TopoDS.Shape> argumentlist)
{
  TopoDS.Shape Result;
  try
  {
    BOPAlgo_BOP aBOP;
    if(argumentlist.count() > 2)
    {
      for(int i=1;i< argumentlist.count();i++)
      {
        aBOP.AddArgument(argumentlist.at(i));
      }
    }
    else
    {
      aBOP.AddArgument(argumentlist.at(1));
    }
    aBOP.AddTool(argumentlist.at(0));
    aBOP.SetOperation(BOPAlgo_FUSE);
    aBOP.Perform();
    Result = aBOP.Shape();
  }
  catch(...)
  {
  }
qDebug() << "boolean crashed";
}

return Result;
}

TopoDS_Shape HSF::BooleanCommon(TopoDS_Shape Stock, TopoDS_Shape Tool)
{
    int count = 0;
    if(Stock.IsNull()) count++;
    if(Tool.IsNull()) count++;
    if(count > 0) return TopoDS_Shape();
    TopoDS_Shape Result ;
    try
    {
        BOPAlgo_BOP aBOP;
        aBOP.AddArgument(Stock);
        aBOP.AddTool(Tool);
        aBOP.SetOperation(BOPAlgo_COMMON);
        aBOP.Perform();
        Result = aBOP.Shape();
    }
    catch(...) 
    { 
    }
    return Result;
}

TopoDS_Shape HSF::BooleanSubstract(TopoDS_Shape
TopoDS_Shape Result;

try
{
  if (!Stock.IsNull() && !Tool.IsNull())
  {
    Result = BRepAlgoAPI_Cut(Stock, Tool);
  }
}
catch(Standard_Failure)
{
}

return Result;

const gp_Pln& HSF::AddNewPlane(TopoDS_Shape SupportSurface,
                                 TopoDS_Shape SupportEdge, Standard_Real uRatio)
{
  const TopoDS_Face& aFace = TopoDS::Face(SupportSurface);
  const TopoDS_Edge& aEdge = TopoDS::Edge(SupportEdge);
  Handle(Geom_Surface) aSurf = BRep_Tool::Surface(aFace);
  gp_Pnt normalpoint = AddNewPointOnCurve(SupportEdge, uRatio);
  gp_Vec normalVector =
    getVectorNormalToSurfaceAtPoint(SupportSurface, normalpoint);
  gp_Vec TangentVector =
    getVectorTangentToCurveAtPoint(SupportEdge, uRatio);
  gp_Vec PerpendicularVector =
    normalVector.Crossed(TangentVector);
gp_Dir PerpendicularDir(PerpendicularVector);

gp_Pln aPlane(normalpoint,PerpendicularDir);

return aPlane;

}

const gp_Pnt & HSF::AddNewPointonCurve(TopoDS_Shape SupportEdge, Standard_Real uRatio)
{
    gp_Pnt p1;
    if (SupportEdge.IsNull())
        return p1;
    const TopoDS_Edge & aEdge = TopoDS::Edge (SupportEdge);
    Standard_Real aFP, aLP, aP;
    Handle(Geom_Curve) aCurve = BRep_Tool::Curve(aEdge, aFP, aLP);
    aP = aFP + (aLP - aFP) * uRatio;
    p1 = aCurve->Value(aP);
    return p1;
}

const TopoDS_Shape & HSF::Get2dPntonSurfacefromPoint(TopoDS_Shape SupportSurface, gp_Pnt point)
{
    const TopoDS_Face & aFace = TopoDS::Face (SupportSurface);
    Handle(Geom_Surface) aSurf = BRep_Tool::Surface(aFace);
    Standard_Real u1, u2, v1, v2;
    BRepTools::UVBounds(TopoDS::Face(aFace), u1, u2, v1, v2);
    Handle(ShapeAnalysis_Surface) aSurfAna = new ShapeAnalysis_Surface (aSurf);
gp_Pnt2d pUV = aSurfAna->ValueOfUV(point,
   Precision::Confusion());

double newx = hsf::map(pUV.X(), u1, u2, 0, 1);

double newy = hsf::map(pUV.Y(), v1, v2, 0, 1);
pUV.SetX(newx);
pUV.SetY(newy);
return pUV;

const gp_Pnt2d& HSF::Get2dPntonSurfacefromPoint(gp_Pln pln1,
   gp_Pnt point)
{
   gp_Pnt projp = HSF::ProjectPoint(point, pln1);
   Handle_Geom_Plane myplane = GC_MakePlane(pln1);
   Handle(Geom_Surface) aSurf = myplane;
   Handle(ShapeAnalysis_Surface) aSurfAna = new
      ShapeAnalysis_Surface(aSurf);
   gp_Pnt2d pUV = aSurfAna->ValueOfUV(projp,
      Precision::Confusion());
   return pUV;
}

const gp_Vec&
   HSF::getVectorNormaltoSurfaceatPoint(TopoDS_Shape
   SupportSurface, gp_Pnt point)
{
   if(SupportSurface.ShapeType() != TopAbs_ShapeEnum::TopAbs_FACE)
   {
      return gp_Vec(0, 0, 0);
   }
gp_Pnt projp = HSF::ProjectPoint(point, SupportSurface);

const TopoDS_Face& aFace = TopoDS::Face(SupportSurface);

// Get 2d UV data
gp_Pnt2d pUV = Get2dPntonSurfacefromPoint(SupportSurface, projp);
BRepAdaptor_Surface aSurface(aFace);
gp_Vec ut, vt;
gp_Pnt pt;
aSurface.D1(pUV.X(), pUV.Y(), pt, ut, vt);
gp_Vec V = ut.Crossed(vt);
Standard_Real mod = V.Magnitude();
if (mod < Precision::Confusion()) qDebug() << "Vector has no Magnitude";

// consider the face orientation
if (aFace.Orientation() == TopAbs_REVERSED || aFace.Orientation() == TopAbs_INTERNAL) {
    V = -V;
}
return V;

const gp_Pnt& HSF::ProjectPoint(gp_Pnt p1, TopoDS_Shape Surface)
{
    gp_Pnt resultpoint;
    if (Surface.ShapeType() != TopAbs::TopAbs_FACE)
    {
        return resultpoint;
    }

    TopoDS_Face aFace = TopoDS::Face(Surface);
Handle_Geom_Surface aSurf = BRep_Tool::Surface(aFace);
GeomAPI_ProjectPointOnSurf Proj (p1, aSurf);
if (Proj.NbPoints() > 0)
{
    resultpoint = Proj.NearestPoint();
}