FEDERATION MODELER: A TOOL FOR ENGAGING
CHANGE AND COMPLEXITY IN DESIGN

by

Davis Mota Marques
Diploma of Technology, British Columbia Institute of Technology, 1993
Bachelor of Architecture, Southern California Institute of Architecture, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Interactive Arts and Technology

© Davis Mota Marques 2007
SIMON FRASER UNIVERSITY
Summer 2007

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.
APPROVAL

Name: Davis Mota Marques

Degree: Master of Science

Title of thesis: Federation Modeler: A Tool for Engaging Change and Complexity in Design

Examinining Committee: Dr. Chris Shaw
School of Interactive Arts and Technology
Simon Fraser University
Chair

Dr. Robert F. Woodbury, Senior Supervisor
School of Interactive Arts and Technology
Simon Fraser University

Dr. John C. Dill, Supervisor
School of Engineering
Simon Fraser University

Dr. Axel Kilian, External Examiner

Date Approved: MAY 03 2007
Declaration of Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada

Revised: Summer 2007
STATEMENT OF ETHICS APPROVAL

The author, whose name appears on the title page of this work, has obtained, for the research described in this work, either:

(a) Human research ethics approval from the Simon Fraser University Office of Research Ethics,

or

(b) Advance approval of the animal care protocol from the University Animal Care Committee of Simon Fraser University;

or has conducted the research

(c) as a co-investigator, in a research project approved in advance,

or

(d) as a member of a course approved in advance for minimal risk human research, by the Office of Research Ethics.

A copy of the approval letter has been filed at the Theses Office of the University Library at the time of submission of this thesis or project.

The original application for approval and letter of approval are filed with the relevant offices. Inquiries may be directed to those authorities.

Bennett Library
Simon Fraser University
Burnaby, BC, Canada
Abstract

Increasing change and complexity in the task environment impedes the ability of designers to explore and develop designs. We hypothesize that by focusing on the management of interactions that realize a design, rather than on the specification of the design itself, the capacity of designers to deal with change and complexity can be increased. Our exploratory research aims at a parametric modeling application that aids designers in managing the interactions of parts that realize a design. We describe the system design, report on its current state of implementation, and present an initial evaluation.

Keywords: Computer Aided Design, Design, General System Theory, Organizational Management, Parametric Modeling
For Nikki, with love.
“... intellectuals should no longer be asked to erect themselves as master thinkers or providers of moral lessons, but to work, even in the most extreme solitude, at putting into circulation tools for transversality.”

— Felix Guattari, Chaosmosis, An Ethico-Aesthetic Paradigm, 1992
Acknowledgments

That this work has been able to mature from speculation to the level of an implemented system is a testament to the efforts of those who have supported me, those who have taught me, and those who have contributed directly to its making. I wish to thank the following individuals for their continued support:

My parents Adelino and Rosa Marques, and partner Nikki Chen who have encouraged and made possible this extended adventure, without question and with great sacrifice. My senior advisor, Dr. Robert Woodbury, without whose contributions, mentorship and friendship this work would remain only speculation. Dr. John Dill, who provided critical feedback on its development. Thom Mayne and George Yu who offered their time to discuss this research and share ideas, and whose own work provided much inspiration to its development. Dr. Halil Erhan for his critical commentary and suggestions. Victor Chen, Maryam Maleki, Cheryl Qian, Roham Sheikholeslami, Dr. Axel Kilian and Dr. Chris Shaw, who participated in its examination.

Ted Ngai, with whom I spent many long nights in studio sharing and debating ideas on architecture that ultimately led to this work. My teachers at the Southern California Institute of Architecture — Karl Chu, Michael Dobry, Linda Hart, Ron Golan, Aris Janigian, Eric Kahn, Perry Kulper, Michael Rotondi, Michael Speaks and Russell Thomsen — who guided my early development as an Architect.

Ashok Shah, Hector Lo, Douglas MacLeod, Shilpi Rao, Homer Perez, Jinsil Seo, Edward and Sandi Palushock; the Chesterton, Reed, Sharma, and Tarbet families; former colleagues at McCall Design Group and peers at Simon Fraser University for their friendship.

These individuals have left an indelible mark in my life and on this work. Thank-you.
Contents

Approval ii
Abstract iii
Dedication iv
Quotation v
Acknowledgments vi
Contents vii
List of Tables x
List of Figures xi
List of Programs xviii
Preface xix

1 Introduction 1
  1.1 Engaging Change and Complexity: A Motivating Example 3
    1.1.1 IFCCA Design Process 5
    1.1.2 A System of Interacting Parts 7
  1.2 Systems Theory and Practice 14
    1.2.1 General System Theory 14
    1.2.2 Managing Systems 22
  1.3 Cognitive Models of Designer Behavior 25
1.3.1 The Mind as Object of Scientific Inquiry .................. 26
1.3.2 The Emergence of Computing and Artificial Intelligence ........ 28
1.3.3 Theory of Information Processing .......................... 29
1.3.4 Cognitivism and Early Research in Computer Aided Design .... 31
1.4 Tools for Amplifying Designer Action .......................... 40
   1.4.1 Parametric CAD Systems ................................ 40
   1.4.2 Design Space Explorers ................................ 43

2 Exploratory Research ............................................. 48
   2.1 An Exemplary Problem in Parametric Modeling and Design .... 49
   2.2 Research Proposal ........................................... 50
      2.2.1 A System to Support Adaptive Parametric Modeling .... 50
      2.2.2 System Interactions and Modeling Idiom ............... 56
   2.3 Modeling State and Process .................................. 62
      2.3.1 The Systolic Array ..................................... 63
      2.3.2 Propagation System Design .............................. 65
   2.4 Managing Changing Relationships ............................... 67
      2.4.1 Explicit and Implicit Relations ......................... 67
      2.4.2 Studies in Implicit Relational Modeling ............... 69
      2.4.3 Behavior and IContext ................................. 72

3 Federation Modeler ................................................. 76
   3.1 System Organization ......................................... 76
      3.1.1 Model Interfaces ....................................... 78
      3.1.2 Model Classes .......................................... 80
   3.2 Graphical User Interface Studies and Implementations ......... 88
      3.2.1 Dependency Graph Viewer ............................... 88
      3.2.2 3D Model Viewer ....................................... 92
      3.2.3 Stack Viewer ........................................... 92
      3.2.4 Abstract Tree Explorer ............................... 94
      3.2.5 PropertySheet ......................................... 94

4 Evaluation of Current Research .................................. 96
   4.1 Attacking the Simplicity of a Single System ................... 97
List of Tables

1.1 Taxonomy of Independent Components. © 1998, Morphosis Architects/George Yu, by permission. .................................................. 8
List of Figures

1  Representations of the body. a) The symbolic body. Jehan Cousin. Livre de
pourtraiture. Paris: Jean Leclerc, 1608. © 2007, National Institute of Health,
work in the public domain. b) The physical body. Albrecht von Haller,
© 2007, National Institute of Health, work in the public domain. c) The
process body. Metabolic pathway for respiration in S. cerevisiae. Adapted
from http://biolab.cin.ufpe.br/tools/grnp.htm

2  Searching for architecture from the bottom up. a) Chronomorphology. Davis
Marques, William McNaughton, Rene Rodriguez, Karl Chu (Instructor).

3  Superposition of systems G7-1.3, G8-3.2a, and G8-4.3. Library for the Infor-

4  Interacting organizations. Machinic Assemblages. Davis Marques, Karl Chu

1.1  IFCCA Competition Site, New York City. The site is bounded on the west
by the Hudson River, east by 8th Avenue, north by 30th Street, and south by
34th Street.

1.2  Noodles and Linkers. Interactions between Independent Components lead
to ‘formal mutations’ and their adaptation to local circumstances. © 1998,
Morphosis Architects/George Yu, by permission.

1.3  Scenario 33. Low density scenario with large open park area, waterfront
Conquistador. © 1998, Morphosis Architects/George Yu, by permission.
1.4 Scenario 15. Medium density scenario with relocated Madison Square Gardens, waterfront recreational facilities. © 1998, Morphosis Architects/George Yu, by permission. .......................................................... 10

1.5 Scenario 28. High density scenario with relocated Madison Square Gardens, waterfront recreational facilities and Conquistador. © 1998, Morphosis Architects/George Yu, by permission. .......................................................... 11

1.6 Analytical Procedure. Knowledge of an object or phenomena of interest is acquired by decomposing it into its constituent parts. Knowledge of the parts and their relations should then, in theory, enable the recomposition of the phenomena of interest. .......................................................... 16

1.7 Weaver distinguishes three classes of problems in the sciences. a) Problems of Simplicity, involving one to three variables in relation. b) Problems of Organized Complexity, involving small to large numbers of variables with identifiable relations. c) Problems of Unorganized Complexity, involving very large numbers of variables with no discernible governing relation. ............ 17

1.8 Information Processing System. Adapted from Newell and Simon, 1972. .... 31

1.9 Spreadsheets are an example of an application of the Systolic Array. a) To add a column of numbers in the spreadsheet, the user enters values and expressions in the matrix of cells. b) The system creates an underlying symbolic model that represents the network of dependencies between cells. Cell values are propagated through the network to perform calculations. .................. 41

1.10 Generative Components, Bentley Microsystems Inc. In Generative Components, models are represented in both symbolic (left) and 3D geometric (right) views. The symbolic view represents the process that realizes the geometric model. The 3D geometric view represents the result of that process. ............. 42

1.11 Conceptual diagram of Design Space. The Design Space is the set of realized or potential designs for a given problem. Represented designs comprise the explicit design space. The set of undiscovered designs comprise the implicit design space. When designs are related, we say a path exists between them. Adapted from 2006, Woodbury and Burrow. ................................. 44
2.1 An Assembly is a composition of one or more elements that together form a 'whole'. The relations that define an Assembly vary along a continuum. At one end of the spectrum, a) an Assembly may take the form of a collection of elements with no interrelations, and at the other b) an Assembly may comprise parts with strong relations. In the latter case, an Assembly can be interpreted as both a parametric object and a system. ........................................ 51

2.2 Behavior enables a model object to respond to changes in its environment. a) The Designer specifies a set of target objects or conditions that the Behavior should watch for in the host object’s context. b) When the condition occurs, the Behavior triggers some action that enables the host object to respond to those changes. ......................................................... 53

2.3 Assembly, Behavior, Scenario interactions. Directed edges represent the possible relations or interactions between the three principal elements of the system. Following from those relations, Scenario plays the role of the top level object. Assembly is a part of Scenario and may itself interact with sub-Assemblies recursively. Behaviors mediate contingent relations between instances of Assembly and Scenario. .................................................. 56

2.4 Scenario Graph. Scenarios interact by inheriting elements from other Scenarios. Transactional elements are depicted in khaki; contextual elements inherited from other Scenarios are ghosted in white. The network of relations between Scenarios reflects the Designer’s explorations and the changing context of design activity. ......................................................... 59

2.5 Masking. a) Scenarios may inherit one or more elements of another Scenario as contextual elements. b) Making a change to a contextual object in a Scenario generates a local copy of the object. The copy becomes part of the Scenario’s collection of transactional elements, and hides or masks the original contextual object. This enables the designer to override prior design decisions without compromising other Scenarios that inherit that object. ... 61
2.6 Characteristic patterns of change between Scenarios. a) Development occurs when transformations are enacted on one Scenario to produce another new Scenario that is closer to the design goal state. b) Variation occurs when transformations are enacted on one Scenario to produce two or more new Scenarios that differ from the original Scenario. c) Composition occurs when two or more Scenarios are merged to create a new Scenario.

2.7 Systolic Array. The Systolic Array is comprised of Processing Elements and Links. Complex computations can be performed by sequencing individually simple Processing Elements in the Array.

2.8 Rule Based Selector in Generative Components. The Rule Based Selector filters a set of objects to find those that match a set of conditions specified by the Designer. The interface provides three distinct views of the model (clockwise from top left): a) symbolic graph, b) model view, and c) property editor. The Designer enters selection rules through the property editor. In this example, a subset of points is selected in the point grid. A line is drawn from each selected point to another point to the top right of the grid.

2.9 Selection rules can be used to make complex object selections with relatively little work. Here, the Rule Based Selector is employed to select two subregions of a grid of points. The large dots represent the selected points; the statements at the bottom of the image are the corresponding selection rules.

2.10 Observer pattern. The Observer registers its interest in the Observable object. The Observable object notifies the Observer when a change occurs to it.

2.11 Proxy pattern. The Proxy acts as a surrogate for the Target object.

2.12 Dynamic Proxy pattern. The Client registers its interest in a set of Target objects with the Proxy. The Proxy searches a specified Context for objects matching the Target properties whenever a change occurs in that Context. When a match is found, the Proxy notifies the Client.

3.1 Model-View-Controller pattern.

3.2 Composite pattern. IContext is a subinterface of INamed, and contains elements that implement INamed.
3.3 Dependency Graph, conceptual study of Scenario level view. Thumbnail images depict Scenarios and their corresponding state; links between Scenarios represent inheritance relationships. Text annotations enable the Designer to record her thoughts and label groups of Scenarios. The graph, as a whole, passively captures aspects of the overall design process.  

3.4 Dependency Graph, conceptual study of Assembly level view. Nodes in the graph represent Components, solid links between nodes represent explicit relations, and dashed links represent implicit relations. Behaviors are represented using two nodes. Here, 'BendLine' represents the action that is activated when an object matching the conditions specified by 'TargetObject1' is found.  

4.1 Sketch of a hybridized parking organization for a project in Las Vegas. a) Parking diagram for a residential block. b) Parking diagram for a commercial or office block. c) A hybridized parking block that combines the performance criteria of both housing and commercial programs. Adapted from sketch by Mayne. (2007)  

4.2 Symbolic graph in Generative Components. Complex objects become difficult to understand through a graph based representation. Encapsulation of subgraphs within nodes of the graph, or alternative representations such as text may provide means to make complex models more comprehensible.  

A.1 Model interface hierarchy. All Federation model objects implement the INamed interface. This enables them to be identified and addressed by their name. Objects implementing IContext contain other addressable subobjects. Objects implementing IReadable provide 2D and 3D graphic representations. Objects implementing IGraphable may have dependancies on other model objects. Objects implementing IUpdatable have states that are transient or may change over time.  

A.2 Model class hierarchy.  

A.3 ProcessingElement and Behavior class hierarchy. ProcessingElements may be extended to create more specialized objects in the model. ProcessingElements and Behaviors are subparts of container-like objects that implement the IContext interface.
A.4 Systolic Array update sequence. The Systolic Array gathers the dependencies for each ProcessingElement, then performs a topological sort on the graph to order the update sequence from independent elements to dependant elements. The update method on each element that implements IUpdateable is then called in order to update the state of the Systolic Array.

A.5 New Behavior construction and initialization sequence. Behaviors are ProcessingElements instantiated as subparts of an IContext. When given a selection rule, the Behavior observes a target context for changes.

A.6 Behavior update sequence. When the Behavior receives an update notification from the target context, it searches the context for objects of interest and then notifies the model object to which it is bound to update.

A.7 Dependency Graph Viewer, 'container' icon study. a) The icon depicts the current state of the object and its subparts. b) If a Behavior is associated with the object, a gray bar appendage appears at the top of the icon. c) In this study, we envision that double-clicking the Behavior bar opens it to reveal the list of Behaviors bound to that element.

A.8 Dependency Graph Viewer, 'component' icon study. The component has no accessible or representable subparts. Consequently, the icon provides only a symbolic representation of the Component.

A.9 Scenario Graph, conceptual study, June 2005. The Scenario Graph represents the iterative and incremental development of designs over time.

A.10 Modeler, conceptual study, June 2005. The Modeler provides an interface for creating and editing geometric objects. Prior model states that inform the current state are depicted schematically at the bottom of the modeling view.

A.11 Scenario annotation, discussion and markup, conceptual study, June 2005. The interface allows Designers to collect references to external data that describes the context, record thoughts about that context, and annotate the model.

A.12 Scenario Graph Viewer, interface study.

A.13 Model Viewer, interface study.

A.14 Model Viewer with Scenario Graph overlay, interface study.

A.15 Model Viewer with single model object, interface study.

A.16 Graph Viewer showing graph for a single model object, interface study.
A.17 Code Editor, interface study. .................................................. 152
A.18 Stack Viewer, conceptual study depicting a flat listing layout. The Stack Viewer depicts the inheritance chain for the current Scenario. The listing is ordered from farthest to nearest predecessor. a) The current Scenario is highlighted at the bottom of the stack. Predecessors at the same depth in the inheritance tree (ex. Scenario3, Scenario4) are grouped in the listing. b) Alternative means of differentiating the current Scenario from the predecessors were explored. .................................................. 153
A.19 Stack Viewer, conceptual study depicting a pseudo-tree layout. The pseudo-tree layout is intended to aid the Designer in understanding the structure of inheritance relations. However, the representation is uneconomical in terms of the screen space it requires and cannot scale to represent large, complex inheritance chains. .................................................. 153
A.20 Stack Viewer, current implementation. ....................................... 154
A.21 Abstract Tree Explorer, current implementation. The Abstract Tree Explorer provides a hierarchical view of the model. ........................................... 154
A.22 Model Viewer, current implementation. The Model Viewer is the principal application interface. It enables the Designer to see and manipulate geometric objects in the model. ........................................... 155
A.23 Property Sheet viewer, as implemented. The Property Sheet enables the Designer to inspect and edit model object properties. ............... 156
# List of Programs

<table>
<thead>
<tr>
<th>B.1</th>
<th>IContext.java</th>
<th>157</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.2</td>
<td>IGraphable.java</td>
<td>158</td>
</tr>
<tr>
<td>B.3</td>
<td>INamed.java</td>
<td>159</td>
</tr>
<tr>
<td>B.4</td>
<td>IUupdateable.java</td>
<td>160</td>
</tr>
<tr>
<td>B.5</td>
<td>IViewable.java</td>
<td>161</td>
</tr>
</tbody>
</table>
The human body is composed of cells that are born, live, and die in a cycle that renews the body continuously until death. That the body changes is no great mystery. But, that it changes continuously and in great proportion is perhaps not as evident. Consider for a moment that every two weeks the entire surface of your skin is shed and replaced; every 120 days, all your red blood cells; each year, your liver; every ten years, your entire skeleton; and every fifteen years, your digestive tract and nearly all the muscle tissue in your body. [45]. To our senses, the body appears stable and complete. But, zooming in closer, the stability of parts dissolve, and expanding our horizon of time, we begin to see the metamorphosis of the whole and its interplay with the world around it. Such a transformation belies the apparent persistence of the body, and unravels the image of the body as an object.

Throughout history, conceptions of the body have provided inspiration to the arts. The symbolic body found its sense in the idea of an immutable and eternal universe. The physical body began to reveal the logos of its organization in the chaos of form. Ideas of order and organization in the body where interpreted by artists and architects and translated into the artifacts of everyday existence.

The notion of the body as process provokes new questions about the way we think about and realize designs in the world. To paraphrase Ackoff, one does not cure a headache by applying aspirin directly to the head. Instead, we take the path of least resistance by swallowing the aspirin and allowing the body’s metabolic processes to transport the appropriate compounds to the centers of pain itself. Ackoff’s parable is fitting for this period of increasing change and complexity in human affairs. However, achieving this level of simplicity in design will necessitate not only new ideas and approaches, but new tools that can aid designers in thinking about the whole while working on the parts.

**An Evolving Conception of Design**

The work presented here is the culmination of an interest in architecture and computing that has occupied me to varying degrees over the last decade. The influences on this work have been many and the integration of ideas slow and piecemeal. I offer this short interlude into my history with these ideas as a means of situating for the reader the context of their development.

My early work as an architecture student was influenced primarily by Karl Chu, my teacher and mentor for many years at SCI-Arc. Karl was invested in the idea of using artificial life systems to generate architectural form; his interest soon became mine. I developed my first computational form generating system with the help of two classmates (Figure 2). The exercise was both fruitful and frustrating. Creating a system ex nihilo to generate architectural form is an incredibly difficult and opaque task. In my mind, the work was a provocative and productive failure. The organic qualities of the resulting forms struck me deeply. However, the effort to realize those forms made it abundantly clear that we were taking random walks through a design space so vast that developing a purposeful design in this way seemed inconceivable.

The Library for the Information Age competition afforded a second opportunity for research. Taking my prior work as a point of departure, I developed eight geometry generating
systems, each sharing a common underlying logic but with different geometric manifestations. The work soon stalled at the chasm between random form and architecture. For better or worse, the pressure of deadlines keeps one from lingering too long on any one thing. From my collection of system output, I cut those chunks that appeared to me to have interesting architectural properties and pasted them unconsciously together in one model (Figure 3) with the hope of assembling a design in time for my review. The resulting superposition of forms was surprising. Something useful had occurred but I wasn't sure what it was. Some days later my confusion began to clear and what developed was a recognition that architecture is by and large an art of composites and that single systems can not address the complex and faceted conditions that characterize architectural design. I conjectured that by superimposing and editing different systems together, I had effectively combined them to make a more complex whole that could address a wider range of differences more readily.

The competition design provided fodder for my thesis research. It was clear that the approach that was developing involved multiple systems coming together to form a composite system and, from their interactions, architectural form would arise. I took inspiration from notions of computational agents and the body as a kind of process-organization. The increasingly central role of algorithms in the work began to transform my conception of designing from that of specifying objects to that of creating organizations that would interact to realize a design. I was led to the notion of Assembly, as I termed it, as a composite, agent-like, form-generating system; and Behavior as a form of agency that would enable Assemblies
to change and adapt to their circumstances over time. In my thesis project (Figure 4), I suggested that Assembly and Behavior could be used together to create combinations of systems that could be employed as designs for specific purposes. However, while I felt that Assembly and Behavior were a conceptual step forward over my prior work, I recognized that the design space for this approach remained vast and the connection to architectural program and context tenuous. Something was clearly still missing.

My employment with George Yu following SCI-Arc was fortuitous. George had spent the prior summer in collaboration with Morphosis Architects on an entry for the IFCCA Competition for the Design of Cities, and showed me work from their project. With some explanation on his part, the overlap between my own research and what they had created in their competition entry became apparent to me. However, it was also clear that their work had answered many questions that mine had not. Not only had they created a clear working process around a strategy of designing through interactions but, their entry made clear an approach to the treatment of the ‘parts of the design’ and the use of scenarios to manage exploratory action. Most powerful for me was their use of scenarios because they provided not only a connection between the context and program of the design, and the design itself but, they afforded a means to control the size and scope of the design space during exploration.
Not long afterward, we employed some part of the IFCCA approach in a design project for a large shopping mall in Japan. Working with George over the course of a number of months, we designed and redesigned aspects of the shopping mall repeatedly. The Assemblies of our design, or ‘Independent Components’ as they were termed in the IFCCA project, retained their identity throughout the process despite constant reworking of their forms. The robustness of these elements in the face of continual change convinced me of their viability for more general use. That we had used this same strategy at the scale of a building also suggested that the approach had relevance for other types of design work smaller than that of a city block.

Since returning to school two and a half years ago, I have endeavored to take up this line of inquiry again. The research presented here attempts to trace some part of the context around these ideas, and to resolve in concrete terms a system that can not only support this particular approach to design but, can do so in a potentially useful and relevant way. Much remains to be done. I offer the work presented here as a first sketch, toward a greater whole.
Chapter 1

Introduction

Design is distinguished in many respects from other intellectual endeavors. Simon [37, p.58] explains that, “The natural sciences are concerned with how things are. ... Design, on the other hand, is concerned with how things ought to be, with devising artifacts to attain goals.” Designers interpret design problems and formulate design objectives. They explore opportunities and represent possibilities, reflect on prior designs and communicate design proposals to others. Designs are contingent upon the contexts in which they arise. [6]. When a Designer creates a design, she does so in light of her current understanding of what is to be accomplished and the opportunities available to her at that time.

Contingency presents a twofold challenge to Designers. First, change is a prominent and growing force in our time. Peter Drucker argues, “No century in recorded history has experienced so many social transformations and such radical ones as the twentieth century.” [14]. Traditional design approaches focus on the specification of the object to be realized. When the context changes, the contingent relations upon which the design are based break and the design must be revised to regain its persuasiveness. [6]. Because Designers are constrained by limitations of resources and time, change presents a significant impediment to exploration and development of design work in a traditional approach. Second, the context of design and the objects of design themselves are more complex now than ever before. While industrialization has significantly expanded our capacity to produce, it has not led to a similar increase in our ability to address both the interrelatedness and particularity of phenomena that characterize design problems. Complexity challenges Designers to be more cognizant of the unique circumstances of each project. However, human cognition is limited in its ability to deal with complexity, and constraints typical to professional practice impede
the exploration and thorough study of contexts that inform design work.

Understanding and devising means for dealing with change and complexity in design is a beneficial and timely area of research. Simon [37] argues that the challenge of research in design, "is to show how empirical propositions can be made at all about systems that, given different circumstances, might be quite other than they are." The proper means of studying design, he suggests, is to observe the way in which means are adapted to environments, "and central to that is the process of design itself." [37, p.58].

Our research has benefited greatly in this regard from an exceptional example of design process. The Morphosis Architects/George Yu IFCCA competition project provides a concrete and intelligible model of how change and complexity can be addressed through design. Our contention is that the approach exhibited therein renders the design as a system of interacting parts. The systemic nature of the process and resulting artifact presents clear opportunities for description and analysis in terms of systems, accords with accounts of Designer cognitive behavior, and creates apparent opportunities for computational support. Drawing on systems models of organizational management, we hypothesize that the capacity of Designers to deal with change and complexity can be increased by focusing on the management of interactions that realize the design, rather than on the specification of the design itself. To this end, we have conducted exploratory research aimed at the development of a parametric modeling application intended to facilitate change in design through the management of interactions.

Our work is presented here in four parts. In Chapter 1, we describe the Morphosis/Yu project and the properties that make it relevant for our research. We situate our observations in the domains of General Systems Theory, Organizational Management, Cognitivist research in Designer behavior, parametric modeling, and a class of design support systems known as Design Space Explorers. We outline major concepts found in each of these research areas and discuss their relevance to the issues at hand. In Chapter 2, we recast the Morphosis/Yu project as an exemplary research problem in parametric modeling and describe particular objectives for research in this area. We propose a novel propagation-based, parametric modeling application intended to facilitate change and development in design. The proposed application is organized around the interactions of three computational entities: Assembly, Behavior and Scenario. Underpinning these entities is a parallel computing architecture known as the Systolic Array. We describe how Behaviors manage interactions
and facilitate change in the model over time through an implicit relational modeling approach that takes the form of the Dynamic Proxy pattern. Chapter 3 provides a detailed description of our implemented system. We outline in detail the high level organization, particular concepts and interactions that characterize the system. We describe our user interface studies and vectors of inquiry, both past and ongoing, in the development of the application. Having developed a demonstrable system, we presented the system to experts in the domain of architectural design and parametric modeling. In Chapter 4, we give an account of our discussions with these individuals. They provide an initial evaluation of the application, our working hypothesis and approach. We conclude with a summary of the current status of system development and recommendations for further research.

1.1 Engaging Change and Complexity: A Motivating Example

In 1998, the International Foundation for the Canadian Centre for Architecture (IFCCA) held an ideas competition called the *Competition for the Design of Cities*. The challenge of the competition was to create a speculative design for a sixteen block area of Manhattan (Figure 1.1). The IFCCA invited a coterie of internationally recognized architects to participate. Among that group was the team of Morphosis Architects and George Yu.

![Figure 1.1: IFCCA Competition Site, New York City. The site is bounded on the west by the Hudson River, east by 8th Avenue, north by 30th Street, and south by 34th Street.](image)

Morphosis Architects is an internationally recognized architectural design firm. Its principal, Thom Mayne, is a winner of the Pritzker Prize in architecture, a founding member of
the Southern California Institute of Architecture and a studio instructor at the UCLA School of Architecture. George Yu is the principal of George Yu Architects, formerly Design Office. He is a recipient of the Canadian Rome Prize, a former project architect with Morphosis Architects, and studio instructor at the Southern California Institute of Architecture.

The Morphosis/Yu team immediately recognized the challenge presented by the enormous site. Designing and managing the development of 16 city blocks of dense urban space would require perhaps more than ten years of work. Within that period, the project would pass through numerous political, social and economic cycles, and the very context upon which the design would be predicated in 1998 would change significantly before its completion. A norm of architectural design is that a project should be integrated in its site. Achieving that degree of integration in a changing context necessitated an approach that would enable both adaptation of the design to change and engagement of the complex and particular conditions of the site. [30]. The history of cities provided inspiration for a novel approach to the challenges of the project:

“The traditional city depended on the stability of its economic and productive structure, a relative uniformity of social composition and the concentrated political power of oligarchies. These conditions were translated into a stable, homogeneous, hierarchic spatial organization.

It is increasingly difficult to experience the city as a linear process or to operate by limiting variation to reinforce a given geometrical structure. It is precisely the instability of the regimes of flexible economic accumulation and the increasing pace of change that bring the variable conditions of new urban structures to the forefront, putting into question traditional city building strategies.

Therefore: a dynamic process with multi Scenario responses (no fixed solutions) to varied forces, the site is to be organized by a set of independent components influenced by formal mutations arising from their interaction, ..., the emerging urban construct is no longer linked to a singular city typology, but a composite of historical and emerging city typologies, simultaneously Cairo, Tokyo, the medieval village, New York City, and Los Angeles, and the new next city.” [29].
1.1.1 IFCCA Design Process

The Morphosis/Yu IFCCA design approach was organized around three conceptual elements: 1) Independent Components, 2) interactions that created 'formal mutations' of these Independent Components, and 3) multi-scenario responses. Independent Components were geometric objects suggestive of an architectural use or intent. Mayne and Yu (Appendix A.1) characterize these components as 'morphological characters' or 'types'. Coupled with each Independent Component was a notion of how the element might be inhabited, how it would transform as a consequence of an interaction with other Components, and the kinds of conditions the element would 'seek'. Independent Components were named so as to capture these traits. For example, 'Noodles' were proportionately long, slender elements that would bend and move between other elements as required to accommodate particular conditions. 'Linkers' were long, thin and relatively small elements that would criss cross between other elements, creating interconnections. Other Components included Warp, Snake, Conquistador, Missle, Bar, Pug, Floater, Bit, Passage, Pod, Hold, and Display (Table 1.1).

Designs were created by bringing Independent Components together in a site model, having them interact with one another, and then organizing their interactions into or around scenarios. Following rules specified in advance, designers would place Components in the site model then adjust and integrate them into their local conditions. Figure 1.2 illustrates a group of Noodles interacting with one another and elements of the site model. Linkers were deployed to cross connect Noodles where opportunities appeared. Independent Components change shape and configuration – what the architects termed 'formal mutations' – as a result of these interactions. The aggregate configuration of these components was the design. Designers could change the 'mixture' of Components to achieve different design objectives.

Scenarios guided the interactions of Components and the development of designs. The concept of scenario as employed in the IFCCA project is related to the concept of version. However, where a version simply makes the distinction that some artifact is different from and related to another, scenarios capture the contingent relation between the context of design and the artifacts that arise in response. In so far as scenarios are interpretations of a context as it exists at some point in time, or speculations about how a context may develop in the future, scenarios can be said to capture or reflect the relation between the subjective state of the designer and the design that arises in response. Scenarios organize design activity into focused explorations, often centered around design narratives and key
factors. Narratives are heuristic devices employed by designers to guide the development of design activity over time. For example, a designer may devise a scenario around the notion of 'growth' of an element over time. The design that follows would then employ this notion to guide design work, and emphasize those characteristics that make growth behavior apparent in the design. Factors are properties that give a scenario identity.

Figure 1.2: Noodles and Linkers. Interactions between Independent Components lead to 'formal mutations' and their adaptation to local circumstances. © 1998, Morphosis Architects/George Yu, by permission.

George Yu (Appendix A.1) explains that the starting point for the IFCCA design was the need for a public park in Manhattan. The design team decided early on to situate the park on an elevated plinth. Subsequently, each design scenario took the plinth as its starting point. Three major design scenarios, identified as scenarios 33, 15 and 28, were developed. Each scenario explored design possibilities based on different levels of investment in building cost, floor area ratio, total daytime population, and total floor area for the development.
Scenario 28 (Figure 1.5) had the highest total floor area, followed by 15 (Figure 1.4) and then 33 (Figure 1.3). In scenarios 28 and 15, a large cluster of Noodles occupy the south edge of the site adjacent to the Chelsea district. The existing Madison Square Gardens building is replaced with a high density tower Component called a Conquistador, and then a new sports facility is designed in a site further west. The waterfront in both of these scenarios is occupied by recreational buildings and artificial beaches. Scenario 33, increases the open park area over scenarios 15 and 28, and the high density cluster of Noodle components is moved farther East.

Yu recalls that one of the original motivations for taking on the project was to experiment with new parametric modeling software. At the time, Alias/Wavefront had just released the first version of Maya, which offered a scriptable modeling environment that could potentially be used to prototype a parametric approach. However, as Yu explains, learning to use Maya while executing the project proved to be too difficult. The early conceptual work of the project was initiated in physical models. Following initial physical studies, the remaining design work was produced using form-Z. Given that form-Z had no scripting support, the team could only suggest through imagery their design intent. As such, they produced a series of Flash animations that were used to communicate the idea of parametric exploration by showing the transformation of the designs based on changes to the four primary factors.

1.1.2 A System of Interacting Parts

The effect of the Morphosis/Yu approach was to render the design as a system of interacting parts. Independent Components were morphological types defined through geometric properties and relations. Components interacted with other Components in the site, leading to formal mutations and an increasing specificity of the Components to their local conditions. Interactions between Components were guided and organized into scenarios that could be presented as contingent designs. This approach supported change and responsiveness to the conditions of the site in a number of ways.

Independent Components were systemic objects with geometric organization and identity. Noodles for example, were proportionately long and slender. When a designer modified an instance of a Noodle, they would stretch and bend it in the CAD model, understanding that it should remain identifiable as a long and slender object. Second, rules or behaviors associated with the Component guided and localized its interactions with other Components. For example, given that Components were intended as building-like objects, in most
# CHAPTER 1. INTRODUCTION

<table>
<thead>
<tr>
<th>Independent Component</th>
<th>Morphological Traits</th>
<th>Programmatic Affinities</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1. Warps</td>
<td>Variable plane, surface continuity (landscapes of intersection, a place of intersection, between public and private, and the exploitation of the vertical dimension as a public realm).</td>
<td>Cultural, institutional, open space and greenery, recreation, service, commercial.</td>
</tr>
<tr>
<td>M3. Snake</td>
<td>Continuous, multi-directional linear volume, large, serpentine, air rights over Lincoln Tunnel access.</td>
<td>Incubator, office, residential, work live lofts, service, commercial.</td>
</tr>
<tr>
<td>M4. Conquistadors</td>
<td>Vertical volume, large profit, power, ambition, infinite.</td>
<td>Commercial, mostly office, partly hotel, partly high end residential, some cultural and institutional.</td>
</tr>
<tr>
<td>M6. Bars</td>
<td>Horizontal volume, small to medium to large, complex adjacencies.</td>
<td>Office, educational, residential, institutional, work live lofts.</td>
</tr>
<tr>
<td>M7. Linkers</td>
<td>Fasteners to the existing city, mimetic transitions, nostalgic of other cities and places.</td>
<td>Office, educational, residential, institutional, work live lofts.</td>
</tr>
<tr>
<td>M8. Pugs</td>
<td>Unique form, strong attractors.</td>
<td>Ecolarium (ecology &amp; aquarium) of the Hudson River system, boathouse, botanical garden, Madison Square Garden, Penn Station, beach, hockey and ice skating.</td>
</tr>
<tr>
<td>M10. Bits</td>
<td>Point grids, X Y Z, too numerous to be counted.</td>
<td>Cafes, information, commercial service, vents for rail yard heat.</td>
</tr>
<tr>
<td>M11. Passages</td>
<td>Solid or void volumetric directional lines providing linkages, thruways, connections, voyeur.</td>
<td>Bridges, pedestrian and visual thruways.</td>
</tr>
<tr>
<td>M13. Holds</td>
<td>Voids in ground surface, subterranean - where inside and outside are concepts that lose their meaning.</td>
<td>Institutional, public space, service, commercial.</td>
</tr>
</tbody>
</table>

Table 1.1: Taxonomy of Independent Components. © 1998, Morphosis Architects/George Yu, by permission.
Figure 1.3: Scenario 33. Low density scenario with large open park area, waterfront Conquistador. © 1998, Morphosis Architects/George Yu, by permission.
Figure 1.4: Scenario 15. Medium density scenario with relocated Madison Square Gardens, waterfront recreational facilities. © 1998, Morphosis Architects/George Yu, by permission.
Figure 1.5: Scenario 28. High density scenario with relocated Madison Square Gardens, waterfront recreational facilities and Conquistador. © 1998, Morphosis Architects/George Yu, by permission.
circumstances they would need to maintain minimum clearances from one another to ensure proper access to light and air. As such, placing a Component next to another Component would effectively trigger the 'setback' rule, and prompt the designer to modify the Component and adjacent Components to resolve the condition. As Independent Components 'interacted' with one another, their forms became more specific to their local circumstances. The local focus of interactions meant that Components could be added and removed from the model without necessitating redesigns of the whole. Notions of agency coupled with the Components guided their deployment and interactions in the model. When situated in a particular condition, a Component might seek to improve its state, or respond to changes in the context in some particular fashion. The role of the designer was to execute those responses in light of the identity of the Component.

Scenarios complemented the exploratory facilities provided by Components by constraining and focusing design activity, providing guiding narratives, breaking the design process into chunks, and enabling reflection. Yu (Appendix A.1) explains that there were four principal factors underpinning the three design scenarios — levels of investment in building cost, floor area ratio, total daytime population, and total floor area for the whole development. The multi-scenario approach organized the exploration of design alternatives on these four specific factors. Particular themes emerged in these scenario explorations: low, medium and high density variations, alternative locations of Madison Square Gardens, waterfront recreational facilities, and so on. We might characterize this emergence of themes as an effective 'clustering' of design variations or factor values. Scenario designs and their corresponding design themes evolved over time. This evolution could be conceived of in terms of a continuous process or an incremental movement (ie. in 'steps') between different conceptualizations. Regardless, the scenario provides a discrete unit of conceptualization or design that can be extended and developed further. The multi-scenario approach is also an overt exploratory strategy that affords increased opportunities for comparison between designs, reflection and learning. By representing scenarios in different states and at different times, the designer is provided opportunities to learn by making comparisons across scenarios, and to make connections between the state of the design and their actions.

The design approach distinguished different strata or areas of concern within the overall process and within the design itself. The development of Independent Components, for example, was a different concern and a different process than the management of interactions between Components and the site, or the development and evaluation of scenarios. Such a
separation of concerns would not preclude integrated development of these different areas but, afforded the opportunity to reduce the interdependencies between activities such that they could be executed asynchronously. Such a separation of concerns also allowed designers to focus on more narrow tasks without having to worry about their immediate interrelation with the larger process or design.

Finally, taken as a whole, the approach facilitated change and adaptation through its focus on process rather than on the product of design activity. In providing Independent Components with identity and rules governing their interactions with other Components, designers would not have to rethink their individual forms when changes occurred. By organizing designs around the basis of interactions, the design could be continually changed and modified without having to think about the design compositionally. By making explicit the process for executing interactions, the design could be realized faster and changes made with less effort.

**Grounding the IFCCA Approach**

The Morphosis/Yu IFCCA project is multi-faceted, and responds to concerns located both within the competition and within the discipline of architecture itself. In attempting to understand what we find most compelling about the work, we have approached it from various perspectives and sought to situate our discussion and analysis in appropriate theoretical grounds. Our readings have led us to a number of research areas that are relevant to the work at hand: General System Theory [43], the systems approach to organizational management, empirical research in designer cognitive behavior, and parametric modeling.

We have argued that the IFCCA approach renders the design as a system of interacting parts. General System Theory provides us with a general model of systems and their properties, and heuristic methods that enable their study. The systems approach to organizational management suggests strategies for improving the performance and adaptability of systems. The cognitive sciences provide insight into the limitations of human cognitive faculties. Designer cognitive behaviors develop from those limitations and, in response, designers have developed characteristic strategies for reducing cognitive complexity when dealing with design problems. The IFCCA approach employs hierarchy and discrete choice throughout the design process, affording means for reducing the cognitive complexity of design tasks. Finally, parametric CAD systems enable the modeling of both objects and relations; in other words, systems. There is a natural affinity between the IFCCA approach,
which we argue renders the design as a system of interacting parts, and parametric tools that facilitate the modeling of systems.

In the following sections, we provide a high level overview of each of these research domains, then revisit the IFCCA approach through the lens of each. In Chapter 2, we draw key themes and principles from those domains to guide the development of our proposed parametric modeling system. Our analysis also serves to locate opportunities for further research.

1.2 Systems Theory and Practice

In the sciences, the notion of system has played an important role in enhancing understanding of complex phenomena. Two areas of development in this regard are critical for our current research. First, General System Theory is a scientific theory thatformulates, "principles that are valid for 'systems' in general, whatever the nature of their component elements and relations or 'forces' between them." [43]. Second, organizational management is a discipline focused on creating means for organizing, executing, measuring, and improving the effectiveness of human work activities. The systems approach [2] to organizational management, also known as the social systemic approach, builds upon General System Theory. It describes human organizations as systems and analyzes the effects that different forms of human organization have on our ability to execute work and adapt to change.

1.2.1 General System Theory

General System Theory emerged in the first half of the twentieth century from the research of scholars working in a variety of scientific disciplines, including biology, psychology, sociology and economics. It developed as a recognition of the structural similarities and isomorphisms of phenomena encountered in different domains and in response to the, "problem of the limitations of analytical procedures in science." As a scientific theory of systems, it aspires to be, "a useful tool providing, on the one hand, models that can be used in, and transferred to, different fields, and safeguarding, on the other hand, from vague analogies which often have marred the progress in these fields." [43, p.34].
CHAPTER 1. INTRODUCTION

From Mechanism to System

While traces of the principal concepts of General System Theory can be found in the work of a number of scholars of the 1920s, including that of mathematician Alfred Lotka and Gestalt psychologist Wolfgang Köhler, Ludwig von Bertalanffy is generally credited as its main proponent. von Bertalanffy explains, “While in the past, science tried to explain observable phenomena by reducing them to an interplay of elementary units investigatable independently of each other, conceptions appear in contemporary science that are concerned with what is somewhat vaguely termed ‘wholeness’, i.e. problems of organization, phenomena not resolvable into local events, dynamic interactions manifest in the difference of behavior of parts when isolated or in a higher configuration, etc.; in short, ‘systems’ of various orders not understandable by investigation of their respective parts in isolation.” [43, p.36].

von Bertalanffy recounts the evolution of General System Theory through his own personal history as a biologist. He explains that in the 1920s, biologists were immersed in a debate between mechanistic and vitalistic conceptions of the organism. The mechanistic conception held that the organism was a complex aggregation of parts that could be understood by decomposing it into subparts. Once the totality of parts were understood, a picture of the whole that accounted for the behaviors and properties of the organism would emerge. The vitalistic conception, by contrast, claimed that the functions and properties of living organisms were due to meta-physical forces within the organism, and were therefore ultimately outside the grasp of science. On the one hand, von Bertalanffy and his colleagues were aware that the mechanistic conception was unable to explain key properties of living organisms – organization, maintenance of the organism, evolution, and so on. However, they were unwilling to concede to a vitalistic conception that would attribute these phenomena to, “soul-like factors hovering in the cell or the organism.” [43, p.89]. In this context, von Bertalanffy and others were led to what was termed the organismic paradigm. [33, p.19].

Advocates of the organismic paradigm contended that organisms were best understood as open and hierarchical systems. von Bertalanffy defined systems as, “sets of elements standing in interaction”. [43, p.38]. Organisms were ‘open’ systems because they exchanged matter and energy with their environments, and ‘hierarchical’ because they were composed of identifiable parts and sub-systems. This reconceptualization of the organism as system had two important implications. First, relations or interactions between the parts of the organism were held as critical to understanding behavior. Consequently, it became necessary
Figure 1.6: Analytical Procedure. Knowledge of an object or phenomena of interest is acquired by decomposing it into its constituent parts. Knowledge of the parts and their relations should then, in theory, enable the recomposition of the phenomena of interest.

to, “not only study parts and processes in isolation, but also to solve the decisive problems found in the organization and order unifying them, resulting from dynamic interaction of parts, and making the behavior of parts different when studied in isolation or within the whole.” [43, p.31]. Second, the openness of the organism suggested that it could no longer be considered separately from its environment. As Rapaport [33, p.21] states, “The function (better said, functioning) of a living system is reflected in the way it reacts to inputs from the environment.” It followed from the definition of the organism as an open system, and from the definition of behavior as arising through interactions, that interactions with the environment should also be considered as important determinants of behavior and necessary to understanding function.

The jump from a theory of the organism as system to a more general theory of systems was not a large one. The sciences are replete with mathematical models of phenomena encountered within different research domains. von Bertalanffy [43, p.13] and others observed that, “The structural similarity of such models and their isomorphisms in different fields became apparent; and just those problems of order, organization, wholeness, teleology, etc., appeared central which were programmatically excluded in mechanistic science.” Simultaneously, other developments in the sciences — Cybernetics, Information Theory, Game Theory, Decision Theory, Topology, Factor analysis — each contributed both directly and indirectly to the possibility and emergence of a general theory of systems. [43, p.90]. In 1937, von Bertalanffy gave a lecture at the University of Chicago where he summarized the
developments to date and presented the program for a new interdisciplinary field of scientific research called ‘General System Theory’. He characterized General System Theory as a general science of ‘wholeness’ whose purpose, “is the formulation and derivation of those principles which are valid for ‘systems’ in general.” [43, p.32]. The aim of this new field would be the integration of research and education in the natural and social sciences through unifying principles, and the development of exact theories in the non-physical research fields.

Limitations of the Analytical Method

General System Theory’s contribution to the sciences was not simply a descriptive one. It pointed to a limitation within the analytical method itself. In a 1948 article, Warren Weaver [47] made a number of distinctions that further our understanding of the context of these developments.

Figure 1.7: Weaver distinguishes three classes of problems in the sciences. a) Problems of Simplicity, involving one to three variables in relation. b) Problems of Organized Complexity, involving small to large numbers of variables with identifiable relations. c) Problems of Unorganized Complexity, involving very large numbers of variables with no discernible governing relation.

Weaver describes the physical sciences as having been organized historically around three classes of problems: problems of simplicity, problems of unorganized complexity, and problems of organized complexity. Problems of simplicity he argued occupied the physical sciences up until 1900. These were the two, at most three or four, variable problems that effectively taught scientists how to deal with variables. Newton’s Laws of Motion are emblematic of this class of problems. Following 1900, scientists began to develop new analytical techniques for dealing with problems of a significantly larger magnitude; problems
with, “two billion variables” as Weaver described it. Weaver discusses this transformation through the example of a billiard game:

“The classical dynamics of the nineteenth century was well suited for analyzing and predicting the motion of a single ivory ball as it moves about on a billiard table. ... But, as soon as one tries to analyze the motion of ten or fifteen balls on the table at once, as in pool, the problem becomes unmanageable, not because there is any theoretical difficulty, but just because the actual labor of dealing in specific detail with so many variables turns out to be impracticable.”

The introduction of Probability Theory and Statistical Mechanics enabled scientists to approach the problems of unorganized complexity, of the interactions of large populations of disorganized entities, and find within them orderly and analyzable average properties. However, Weaver points out that in going from problems of simplicity to problems of unorganized complexity, the sciences had effectively left from one end of the problem spectrum to the other, leaving a vast range unaccounted for. That middle space dealt with problems of organized populations of entities, that is, “problems which involve dealing simultaneously with a sizable number of factors which are interrelated into an organic whole. They are all, in the language here proposed, problems of organized complexity.”

Weaver’s claims begin to shed light on the limitations of the analytical method. von Bertalanffy explains that the analytical procedure is dependent upon two operating conditions. The first is that, “the interactions between ‘parts’ be non-existent or weak enough to be neglected for certain research purposes. Only under this condition, can the parts be ‘worked out’ actually, logically, and mathematically, and then be put together.” [43, p.19]. The second condition is that the relationships between parts be linear such that the equations that describe the behavior of parts also describe the behavior of the whole. Systems are characterized by the, “existence of ‘strong interactions’ or interactions which are ‘nontrivial’, i.e., nonlinear.” [43, p.19]. As such, neither of the necessary conditions for the application of analytical procedures holds for problems of organized complexity.

Natural phenomena found in the world are often too complex to be studied and understood as they are encountered. Scientists simplify their objects of study by focusing on particular items of observation, taken in isolation from the environments in which they occur. The analytical method, as we have seen, works through a process of reduction to components whereby the object of inquiry is decomposed into its constituent elements and
explained through linear causal sequences. As Laszlo and Laszlo [23] explain, “The heuristic of ‘reduction to components’ has led to the accumulation of vast storehouses of information about specific entities and the interactions among them. ... But this type of knowledge proved deficient in one important respect: it did not disclose how complex things behave when exposed to a complex set of influences.” They argue that, “another heuristic became necessary, capable of simplifying unmanageably complex phenomena by reduction to dynamics instead of to components.”

Problems of organized complexity presented serious practical challenges to the science of von Bertalanffy and Weaver’s day. Weaver [47] remarks that organized complexity, “can not be handled with the statistical techniques so effective in describing average behavior in problems of disorganized complexity.” However, Weaver is hopeful and points to developments that he believes provide ‘evidence’ that these problems can be challenged. The first is the development of ‘electronic computing devices’. “These devices are, in flexibility and capacity, more like a human brain than like the traditional mechanical computing device of the past. ... This combination of flexibility, capacity and speed makes it seem likely that such devices will have a tremendous impact on science.” The second development he points to is that of, “the ‘mixed-team’ approach of operations analysis.” Taken together, computing devices and operations analysis permit empirical research into the organizational properties of systems.

Definitions and Properties of Systems

According to Rapaport [33], scientific literature shows a wide divergence in the definition of system, criteria of classification, and evaluation of the goals and benefits of the systemic approach. The root of that divergence, he suggests, lies partly in epistemological differences, partly in differences of value orientation. Rapaport maps the various definitions of system onto a two dimensional classification, with one axis representing an analytic-holistic dimension and the other a descriptive-normative dimension. The analytic-holistic dimension characterizes the means by which knowledge about systems is acquired. The descriptive-normative dimension characterizes the answers to questions, such as ‘How?’ and ‘What for?’, that such knowledge provides.

The analytic method is intended to provide knowledge of some phenomena by examining its constituent parts. The term ‘method’, Rapaport argues, already implicitly suggests the analytic approach because, “the description of a method is tantamount to an analysis of
knowledge acquiring processes.” [33, p.2]. Rapaport suggests that it is inappropriate to speak of a holistic method. Instead, he suggests that the “holistic approach” to knowledge is based on an act of recognition, that is, on an identification of the phenomenon of wholeness. Consequently, definitions of system accordant to an analytic conception focus on the state of the system as the principal object of interest. Definitions pertaining to a holistic conception focus on the identity preserving aspects of a system, that is, what enables a system to retain its wholeness, its principal characteristics. The descriptive-normative axis connotes a value orientation toward systems. At the normative end of the axis are descriptions of systems that exist for some purpose, “and it is these goals that are at the center of interest.” At the descriptive end of the axis are characterizations of systems that are non-instrumental, that reflect the continued existence of the system as an end in itself. Rapaport argues that, “far from being incompatible, these views are complementary, revealing different aspects of a unified approach to system theory.” [33, p.6].

Rapaport [33] singles out identity, organization, and goal directedness as important thematic properties of systems. He draws a parallel between identity and the notion of invariance, which he argues underscores both the analytic and holistic conceptions of systems. Invariance, he says, is not the absence of change, “Rather it is the preservation of constancy in certain respects in the context of change.” [33]. Invariance enables us to recognize systems as such, and their identification is a principal endeavor in scientific pursuits. He notes, “All operational definitions of physical concepts entail theoretical statements that assert invariance amid change.” Recognition of invariance also enables scientists and non-scientists alike to employ analogy to communicate and relate knowledge about phenomena in one domain through established models and concepts in another. Rapaport describes analogy as, “a perceived invariance of a relationship.” [33, p.14].

Organization is qualified by the presence of relationships between the parts that comprise a system. “It is the totality of relations (not the nature of the elements) that defines organization and identifies the concept of system with that of an organized whole.” [33, p.134]. Rapaport distinguishes between a systemic theory of ‘organization’ and a theory of ‘organizations’. Theory of organization attempts to describe the general properties of “organized wholes”, most often in mathematical terms. Rapaport discusses at length an information theoretic approach that attempts to quantify degrees of organization and organizational properties of systems. The theory of organizations is directed toward descriptions of goal oriented behavior. Rapaport states, “Those who study organizations are interested
CHAPTER 1. INTRODUCTION

not so much in properties of those systems that make them easy to describe but rather in the properties that permit them to pursue certain goals.” The theory of organization and the theory of organizations can be said to intersect based on the degree to which, “the ‘interest’ of the members of an organization overlap.”

Rapaport discusses goal directedness in terms of Aristotelean efficient and final causes. Efficient causes are those that precede and generate an event. Final causes are those that follow the event, to which it is said the event is directed. Rapaport presents the example of a furnace that supplies heat to a house and is sometimes automatically turned on and off. An explanation of the furnace in terms of efficient causes, called an analytic-causal explanation, would describe the system comprising the house, ambient air temperature inside the house, ambient temperature and weather conditions outside the house, thermostat, electrical system, and furnace; and then describe the interactions between these elements that lead to triggering the furnace to switch on and off. An explanation in terms of final causes, also referred to as a teleological explanation, would suggest that the furnace is turned on and off to keep the house at room temperature, for the comfort of the occupants.

System Properties of the IFCCA Approach

The Morphosis/Yu IFCCA design process, the Independent Components that comprise the design, and the resultant design itself are characterized by the presence of ‘elements standing in interaction’. Consequently the IFCCA approach exhibits system properties such as identity, organization and goal directedness. Independent Components, in particular, embody these traits. Noodles for example are morphological characters with long, slender geometries. Their identity is organized around their proportions. The organization of the Noodle is founded on its rectilinear cross section, and proportional ratios of length to width and height between the surfaces that delimit its volume. When Independent Components are brought together, they ‘interact’ with one another by forming relations. Such relations accrue into groups, or higher level systems. The identity, organization and goals of these systems develop from the defining attributes of the scenario of which they are a part. For example, Yu suggests that the three scenarios developed for the IFCCA project were based on different levels of investment in building cost, floor area ratio, total daytime population, and total floor area for the whole development. Interactions in each of these scenarios were organized around variations on these four variables.
1.2.2 Managing Systems

Organizational management is a discipline focused on creating means for organizing, executing, measuring, and improving the effectiveness of human work activities. The systems approach to organizational management, also known as the social systemic approach, conceives of human organizations as systems and analyzes the effects that different forms of human organization have on our ability to execute work and adapt to change. [2].

Russell L. Ackoff is a proponent of the social systemic approach and a widely recognized scholar within the domain of organizational management. Ackoff cites an empirical study by Arie de Geus [13] that finds the average lifespan of corporations in the Northern Hemisphere to be less than twenty years. In the study, de Geus remarks that, “The high corporate mortality rate seems unnatural. No living species suffers from such a discrepancy between its maximum life expectancy and the average span it realizes. And few other types of institution – churches, armies, or universities – have the abysmal record of the corporation.” The source of the problem, Ackoff argues, is that the prevailing culture of research and practice in business is organized around an analytic approach that deals with human organizations as aggregations of independent parts whose properties and performances are the sum of the individual properties and performances of subparts. This leads organizations to focus on organizational units and products (i.e. the components of the organization) in their efforts to improve performance and adapt to change. What follows is an increase of individual performances, but no direct correlation with performance on the whole. Ackoff discusses the car as an analog of the corporation, and points to a source of discorrelation between the analytical approach and that which it tries to explain:

“If I brought an automobile in here, ... and we disassembled it, and every single part of the automobile were retained in this room, we would no longer have an automobile because the automobile is not the sum of its parts – its the product of their interactions. ... The automobile loses its essential properties when it is taken apart. That’s true of any system, because the properties of the system derive from the interactions of the parts.” [3].

Ackoff reconceptualizes human organizations as systems. He revisits the definition of system [2, p.5] and defines it as a whole comprised of two or more parts that satisfy five conditions:
1. *The whole has one or more defining properties or functions.* The defining function of a car, for example, “is to transport people on land.”

2. *Each part in the set can affect the behavior or properties of the whole.* A fuel pump that delivers too much or too little fuel to the engine impedes the car’s ability to accelerate or decelerate, and may cause problems for other parts, such as the exhaust system or transmission.

3. *There is a subset of parts that is sufficient in one or more environments for carrying out the defining function of the whole; each of these parts is necessary but insufficient for carrying out this defining function.* “These parts are essential parts of the system; without any one of them, the system cannot carry out its defining function.” He describes the engine, battery and steering wheel of a car as essential parts. Continuing, Ackoff states, “Most systems also contain nonessential parts that affect its functioning but not its defining function.” A car radio, ash tray and floor mats are examples of nonessential parts of a car. “A system that requires certain environmental conditions in order to carry out its defining function is an open system. This is why the set of parts that form an open system cannot be sufficient for performing its function in every environment. A system that could carry out its function in every environment would be completely independent of its environment and, therefore, be closed.” He continues: “The environment of a system consists of those things that can affect the properties and performance of that system, but over which it has no control. That part of its environment that a system can influence, but not control, is said to be transactional. ... That part of a system’s environment that can neither be influenced nor controlled is said to be contextual.”

4. *The way that each essential part of a system affects its behavior or properties depends on (the behavior or properties of) at least one other essential part of the system.* “Put another way, the essential parts of a system form a connected set, that is, a path can be found between any two parts.” In addition, “The essential parts of a system necessarily interact, either directly or indirectly.”

5. *The effect of any subset of essential parts on the system as a whole depends on the behavior of at least one other such subset.* The properties of a system derive from the interactions of its parts, not the actions of parts taken separately. When a system is
taken apart, it loses its defining functions and properties, and so do its parts. Like the individual parts of a system, no subset of parts has an independent effect on the whole. For example, the function of a car is not a property of the wheels nor the engine. By disassembling a car, its essential function and properties are lost.

Ackoff [2, p.11] suggests that *synthesis* is a means of understanding systems that is complementary to analysis. He states that both involve three basic steps:

1. In analysis, something that we want to understand is first taken apart. In synthesis, that which we want to understand is first identified as *a part of one or more larger systems*.

2. In the second step of analysis, an effort is made to understand the behavior of each part of a system taken separately. In the second step of synthesis, an effort is made to understand the function of the larger system(s) of which the whole is a part.

3. In analysis, the understanding of the parts of the system to be understood is then aggregated in an effort to explain the behavior or properties of the whole. In synthesis, the understanding of the larger containing system is then disaggregated to identify the role or function of the system to be understood.

Analysis provides us with insight into the structure and particular behaviors or properties of a system. "It provides the knowledge required to make it work efficiently and to repair it when it stops working. Its product is know-how, knowledge, not understanding." Synthesis, by contrast, provides us with knowledge about the function a system serves in the larger system of which it is a part.

Ackoff concludes that, "*when the performances of the parts of a system, considered separately, are improved, the performance of the whole may not be (and usually is not) improved.*"

Organizational management should consequently be directed toward the *management of the interactions of parts*, and therefore what we need are organizations that can facilitate such management.

**Managing Interactions in the IFCCA Approach**

We take Ackoff's arguments as instructive for our research in parametric modeling and design. Designing and modeling are interrelated activities. Designing is characterized by
a process of change and development. The Designer models a design and in the process of designing, recognizes new opportunities that direct the Designer to change the design further still. Similarly, changes in the context of designing prompt changes to the design. The effort to model a design is not simply the aggregate effort of modeling or designing individual parts on their own. When changes occur to the design, not only must individual components be replaced but they must be reintegrated into their contexts as well. We argue that an analogy can be drawn between a parametric model conceived as a system of interacting parts and Ackoff’s social systemic model of human organizations, and speculate that the capacity of the design process vis-à-vis modeling activity may be improved by focusing on the management of interactions which realize the design. Therefore, what we need are design tools that aid the Designer in managing the interactions that realize the design.

### 1.3 Cognitive Models of Designer Behavior

Building on the methodological practices of the natural sciences, psychologists of the early twentieth century began to study the cognitive abilities of human beings. Their empirical research slowly brought light to distinct limitations in human cognitive abilities. Simultaneously, research spurred on by the needs of two world wars, made the development of general purpose computers possible. By the mid 1950s, it had become clear to researchers in both psychology and computing that they held common aspirations for understanding human ‘problem solving behavior’. Their exchanges lead to a new theory of mind thereafter known as Cognitivism.

The emergence of Cognitivism as an articulated theoretical position coincides with the emergence of computer aided design as a field of research. Design researchers saw the Cognitivist model and its supporting scientific methodologies as a means to understand the cognitive behaviors of human Designers. The aspiration of Cognitivist research in design was not only to understand the basis of human creative processes but, to develop theories of designing that would aid them in devising tools to assist Designers in creating better designs, or relieve them of certain design responsibilities.
1.3.1 The Mind as Object of Scientific Inquiry

At the turn of the twentieth century, the discipline of psychology was in its infancy. Researchers of the period employed introspective methods of observation and analysis in an attempt to elicit clues to the workings of the mind. However, this approach was fraught with problems and the credibility of the practice quickly diminished.

Drawing inspiration from the progress of research being made in the study of animal behavior, John B. Watson, a professor in the School of Experimental Psychology at the University of Chicago, published a 1907 paper entitled, “Studying the Mind of Animals.” [48]. Watson wrote, “The possibility of learning more about the mental life of animals becomes a probability when we consider that our knowledge of the mental processes of infants, children and defective individuals is obtained almost entirely without the aid of language. The moment we take this broader point of view, that the behavior of man expresses his psychology, and are willing to admit that we can scientifically study his behavior, it follows at once that we can build up an animal psychology, because we can study the behavior of animals just as scientifically as we can study the behavior of man.” [46]. The paper was significant in its reframing of the object of psychological inquiry from internal states to external behaviors, and in its proposal for adopting the methodological practices of the natural sciences. Behaviorism, as it was termed, provided a new basis for conducting research on the human mind.

However, over time, scientists began to recognize that they could in fact recognize particular limitations of the brain, and from those limitations could begin to gain insight into the inner workings of the mind. From the late 1930’s forward, empirical research brought new insights into the structure of memory, the mind’s ability to recall information and the limitations on the accuracy of recall. George Miller’s [31] 1956 paper, “The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information,” provided empirical evidence for innate limitations on the brain’s capacity to make accurate judgments of quantities. Across a range of experimental research, Miller observed that the “channel capacity” of individuals, that is their capacity to relay information or make distinctions within a continuum of choices, was limited to seven discrete choices, plus or minus two. Miller recalls,

“In experiments by Kaufman, Lord, Reese, and Volkmann random patterns of dots were flashed on a screen for 1/5 of a second. Anywhere from 1 to more
than 200 dots could appear in the pattern. The subject's task was to report how many dots there were. The first point to note is that on patterns containing up to five or six dots the subjects simply did not make errors. The performance on these small numbers of dots was so different from the performance with more dots that it was given a special name. Below seven the subjects were said to suitize; above seven they were said to estimate.”

Miller called this limitation the span of absolute judgment. He suggests that this effect can be compensated against through other means: “a) to make relative rather than absolute judgments; or, if that is not possible, b) to increase the number of dimensions along which the stimuli can differ; or c) to arrange the task in such a way that we make a sequence of several absolute judgments in a row.”

Claude Shannon [36], a computer scientist rather than a psychologist, wrote in 1950 that constructing computer programs to play chess, while of limited practical importance, could bring theoretical insight into more substantial research areas including the creation of machines for designing various types of electrical and mechanical components, automatic routing of telephone calls, symbolic mathematical operations, strategic decision making in military operations, and the orchestration of melodies. In his view, chess provided an ideal environment for study because, “1) the problem is sharply defined both in allowed operations (the moves) and in the ultimate goal (checkmate); 2) it is neither so simple as to be trivial nor too difficult for satisfactory solution; 3) chess is generally considered to require ‘thinking’ for skillful play; a solution of this problem will force us either to admit the possibility of a mechanized thinking or to further restrict our concept of ‘thinking’; 4) the discrete structure of chess fits well into the digital nature of modern computers.” Shannon’s proposal is notable not only for its characterization of chess as a ‘problem solving’ activity but for its framing of thinking as a potentially mechanical, or more appropriately, computational phenomenon. While Shannon’s interests lay primarily in achieving technical goals, the proposal bought the interests of computer scientists together with psychologists in the evaluation of expert chess players — the model for effective computer chess programs.

Scientists began to pursue this line of inquiry. Among them was Adriaan de Groot who, in 1965, published “Thought and Choice in Chess”, where he relayed the findings of his studies on the cognitive differences between master and amateur chess players. One of the key arguments arising from his study was that memory played an extremely important role
in the capacity of a chess player to perform well. de Groot found that one of the primary
differences between master and amateur players was that masters had a greater capacity
to recall the configuration of game pieces at various stages of game play. This capacity
was enabled by the brain’s ability to associate two or more game pieces as a group, and to
later recognize these configurations of pieces thereby speeding recall. de Groot called these
mental groupings *chunks*. As players gained expertise, the size or complexity of the chunks
would increase correspondingly. Hence, the key difference between master and amateur
players lay in the size and complexity of mental chunks.

Subsequent studies recreated the original study with some variation and corroborated
most of the results, although questions existed as to the relationship between short term and
long term memory in chunking behavior. Gobet and Simon revisited both de Groot’s original
work, and Simon’s own earlier work of 1973 in memory and recall, to reexamine mental
chunking. They proposed that mental chunks that recur often in chess masters’ practice
and study evolve into larger and more complex data structures called *templates*. [17, p.228]
In addition to holding information about patterns of pieces, templates possess slots that
hold information for specific instances of entities, and that enable information about chunks
to be recursively encoded. Templates would develop through normal learning processes, and
encompassed a wider range of memory structure than chunks including symbols representing
plans, moves, strategic/tactical concepts and other templates.

1.3.2 The Emergence of Computing and Artificial Intelligence

Parallel to the developments taking place in the field of psychology, research in computing
systems was being spurred on by the military needs of two World Wars. The result was a
veritable blossoming of the field.

Early work in computing led to the realization of rudimentary computing machines and
new theories of computation. Alan Turing was seminal in this regard. In 1936 he published
“On Computable Numbers, with Applications to the Entscheidungsproblem,” wherein he
argued for the theoretical limitations of the kinds of functions that could and could not be
considered computable, introduced the notion of a logical computing machine (thereafter
known as a Turing Machine) and argued that such a machine would be capable of computing
the values of a function whenever there existed an effective or mechanical means of realizing
the machine — in other words, the Turing Machine was a ‘universal’ computing device.

By 1945, the sophistication of computing systems had reached a point where notable
scientists such as Vannevar Bush argued for an expanded vision of their relevance in human endeavors. Bush wrote, “The repetitive processes of thought are not confined however, to matters of arithmetic and statistics. In fact, every time one combines and records facts in accordance with established logical processes, the creative aspect of thinking is concerned only with the selection of the data and the process to be employed and the manipulation thereafter is repetitive in nature and hence a fit matter to be relegated to the machine.” [10]. Bush’s words were suggestive not only in their implication of human thought as an act of computation, but in their suggestion that humans could be replaced in non-creative endeavors.

In 1956, John McCarthy [21] coined the term artificial intelligence to describe computing machines with such an ability, and with a number of other notable researchers, inaugurated the first conference on the subject. At that conference, Newell, Shaw and Simon demonstrated the first working artificial intelligence application called the Logic Theorist, which could provide mathematical proofs for the first thirty-eight of fifty-two mathematical theorems in Whitehead and Russell’s Principia Mathematica. One year later, Newell and Simon returned to demonstrate the General Problem Solver, a system that employed a heuristic strategy of problem decomposition and iteration to successively approach a goal state.

Other dimensions of research in the computing fields were equally powerful. Ivan Sutherland’s dissertation, “Sketchpad: A man-machine graphical communication system,” was the first system to support interactive drawing and manipulation of vector graphics and, by that measure, is considered to be the first Computer Aided Design application. Sketchpad was also the first example of a computer program that employed a graphical user interface, a rudimentary object model, and constraints. With Sketchpad, Sutherland demonstrated the interactive construction of geometric figures, their replication as instances of a parent figure, interactive manipulation of figures and maintenance of geometric relations during manipulation. [39]. His research work would provide a model for many computer aided design applications that followed.

1.3.3 Theory of Information Processing

It may be argued that the culmination of a number of separate threads of intellectual development came together in Allan Newell and Herbert A. Simon’s 1972 book, Human Problem Solving. As they explain in the opening pages of the book, they sensed that a general change had occurred in 1956 that moved the emphasis of research in psychology,
 CHAPTER 1. INTRODUCTION

linguistics and computer science toward, "the exploration of complex processes and the acceptance of the need to be explicit about internal, symbolic mechanisms." [32, p.4]. For the authors, those changes had culminated in the Theory of Information Processing as a model for representing, measuring and understanding a human being's 'problem solving behavior.' In the opening passages of Human Problem Solving, Newell and Simon [32, p.5] remark,

"An information processing theory is not restricted to stating generalities about Man. With a model of an information processing system, it becomes meaningful to try to represent in some detail a particular man at work on a particular task. Such a representation is no metaphor, but a precise symbolic model on the basis of which pertinent specific aspects of man's problem solving behavior can be calculated. This model of symbol manipulation remains very much an approximation, of course, hypothesizing in an extreme form the neatness of discrete symbols and a small set of elementary processes, each with precisely defined and limited behavior. This abstraction, though possibly severe, does provide a grip on symbolic behavior that was not available heretofore. It does, equally, steer away from physiological models, with their concern for fidelity to continuous physiological mechanisms, either electrical, chemical or hormonal."

Four important themes emerge in this passage: 1) Information Processing Theory is a theory about the cognitive abilities of human beings; 2) human cognition is the act of manipulating symbolic representations, and therefore distances itself from Behaviorist conceptions of human cognition; 3) cognitive activity is organized around tasks; and, 4) Information Processing Theory is inclined toward discreet, logical models of cognition. They continue by proposing that Information Processing Systems, that is systems which best exemplify the Theory of Information Processing, consist of, "a Memory containing symbol structures, a Processor, Effectors and Receptors." Memory contains knowledge and data about the problem in the form of symbols and tokens that represent entities in the Environment, other objects and their relations. The Processor is, "a symbol manipulator that: a) converts the information provided by the Receptors into a code that is internally consistent with the symbol structures of the system, b) transforms internal symbols and their relations, and c) converts internal symbols into code that can be transmitted to the external world or the environment by Effectors." [32]. The Environment is that part of the world outside the
realm of the individual and to which Effectors and Receptors push and pull information.

![Information Processing System](image)

**Figure 1.8: Information Processing System. Adapted from Newell and Simon, 1972.**

### 1.3.4 Cognitivism and Early Research in Computer Aided Design

Research in computer aided design finds its origins in the early 1970's with empirical studies of designing conducted separately by Eastman, Foz, Henrion and Krauss and Myer. [5, p.4]. These studies approached design activity from a number of perspectives but, each found its methodological basis in the research practices of the disciplines of psychology and artificial intelligence. Early empirical research began to identify those operational differences that distinguished design from the model problem solving domains of chess, engineering and science.

**The Psychology of Architectural Design**

Ömer Akin's *The Psychology of Architectural Design* is a major contribution of empirical research on the cognitive processes of designers. Akin [5] describes the objective of the book as being the development of a descriptive model of the creative process that can act as a foundation for education, practice and design research. Akin writes, “The premise of these studies is that mechanisms responsible for the behaviors of problem-solvers, such as designers, managers, and organizations, can be understood by studying their behaviors and by designing information-processing models that account for such behavior.” [5, p.3]. To this end, the book provides an account of empirical studies of amateur and professional designers; proposes an Information Processing theoretic model of designing, and a concomitant Information Processing System.
CHAPTER 1. INTRODUCTION

Akin begins by characterizing the nature of design activity. He argues that design activity can be differentiated into categories of routine design and creative design. Routine design is defined by the application of standardized or vernacular design solutions to problems that are likewise constrained to fit into normative patterns. Creative design by contrast deals with atypical requirements, the invention of non-standard designs, and their realization through what is most often a consciously planned and executed process. In practical terms, routine design presumes little consideration of the particularity of a design problem because the expectation is that the designer or laborer are capable of resolving conflicts and atypical situations in terms of the vernacular as they are encountered, and the specifics of a problem are not so unique that they can not be addressed through standardized solutions. Creative design, by contrast, necessitates forethought and consideration to enable the realization of specific solutions to specific problems. Akin consequently focuses his research on creative design because it provides a more general basis for understanding design activity.

Creative design brings particular challenges. Creative design implies a process that creates changes in and development of a design over time. Because of the difficulty of analyzing and representing continuous change, researchers often approximate continuity as a sequence of discrete events. The notion of Problem Space is introduced to model the evolution of designs through steps. Problem Space may be defined as the, “totality of all information relevant to the problem-solving process and available to the Information Processing System at any give instance ... the basic entity in a problem space representation is a state.” [5, p.14]. In a game of chess, for example, the problem space may be considered as the set of game rules, the game board and game pieces, and the moves available to a player given the current state of the game and the goal of winning. The game state is reflected in the current player turn (ie. white or black), the particular configuration of game pieces on the board, and whether that configuration matches the conditions for winning the game. In the case of a design problem, the problem space may be described by the various choices about dimensions, materials, and construction methods for a given design element given some intended purpose, client budget and laws governing the design of that element. “Alterations of the contents of a state within the state space of a problem are called state transformations.” [5, p.15]. Moving a game piece during a chess game, or altering the dimensions or location of a design element both constitute state transformations. The objective of designing is to locate a state in the Problem Space that fulfills the needs of the design problem.
In *The Sciences of the Artificial*, Simon [37] contrasts problem solving in the sciences against problem solving in the context of design. Simon proposes that design problems are *ill defined* because they typically present no clear goals or methodologies for their solution. Or, as Akin describes, "Ill-defined problems are those which inherently have little structure in terms of their operational parameters: goals, legal operations, alternatives to be considered, and evaluation functions." [5]. Simon proposes that to deal with ill-defined problems, designers must decompose large problems into smaller problems of more limited scope because smaller problems can be more readily addressed. Recomposing small design solutions back together yields a total design that can then be compared to the goal. By repeating the process of problem decomposition, solution recomposition, and comparison of the design against the goal state, a designer can incrementally approach a solution to the design problem. Simon coined the term *satisficing* to denote a design solution which approaches a solution state, or in other words, addresses the problem conditions sufficiently to be deemed acceptable. Akin extends Simon's argument for problem decomposition, as it applies to the intended product of designing, by arguing that the process by which a design is negotiated and realized must also be decomposed, "to make possible the contributions of a large number of participants." [5]. In the case of architectural design, Akin notes the important contributions of engineers, landscape architects, contractors and the client in realizing a design and that those more specialized aspects of the total design solution must be negotiated over time.

Developing and communicating designs necessitates some representation of the problem space and state spaces of the design. Architects employ technical drawings, illustrations, physical models and the like to study and develop designs because of the relative ease and speed by which they can be produced. These representations enable the Architect to study the particular conditions surrounding some design element without having to actually build the element and its surroundings. However, consideration must be given to the effect of the representation on the process itself. The designer needs to understand how the representation impacts their understanding of what is being designed, what the correspondence is between the representation and the intended end result, and how that representation is interpreted and executed upon by each of the participants within the design process. Akin [5, p.15] argues that the utility of a design representation is in direct relation to its capacity to facilitate state transformations in the world. In practical terms, this may be interpreted as being a measure of whether a particular representation makes clear the choices available
to her, whether the transformations required to realize those choices are made clear, or whether the representation enables her to gain the client's confidence in the design, that is to change their understanding of what is being proposed so that they see how it relates to their needs.

Akin summarizes his operational assumptions:

1. There is clear phenomenological evidence that all design takes place through individual decisions that reinforce and build upon each other to achieve a total comprehensive design proposal. This proposal is the objective of the design process. Then, design is a form of problem-solving where individual decisions are made toward the fulfillment of objectives.

2. A fundamental argument stemming from the central role of cognition and the structured nature of cognitive activity is that goal driven systems and their cognitive behaviors can be explained through causal relationships. Consequently, our products in design are not random responses, but clearly articulated products based on thought. Then, the designed product is a direct consequence of the preceding cognitive activity and not some arbitrary process that is independent of such activity.

3. Depending on the knowledge acquired by the cognitive system and the context of the problem at hand, different circumstances will lead to radically different behaviors. ... Then, although designers' knowledge and behaviors may vary, their basic information-handling capabilities such as encoding, manipulating, and recall of information, are essentially similar to the capabilities observed in other task contexts. ... The temporal interface between the ever-changing context in a problem and the knowledge of the problem-solver account for the unique behaviors that result each time a problem is encountered.

Design as Search

Within the Information Processing System (IPS) model of design, the ill-structured nature of design problems means that designers must explore the Problem Space in order to locate states that satisfice goals. The essential challenge of search is to reduce the Problem Space to a manageable set of states that can be visited with less effort in order to determine those states that are satisficing. Akin frames the challenge in terms of finding the transformations
CHAPTER 1. INTRODUCTION

required to reach the goal state: "For each problem state the IPS has the choice of applying one of a multitude of legal transformations. The success with which a goal state is reached depends on the application of the correct sets of transformations. Random selection can make an otherwise trivial problem practically impossible. ... Goal-directed search can drastically cut down the number of necessary transformations." [5, p.16]. Akin introduces the Problem Description Graph (PDG) as a means to model design problems hierarchically, in terms of tokens and attributes with Part-Whole relationships, and to understand the designer's search behavior. He states, "The PDG provides a precise codification of the decomposition of the designed object and the sequence in which the subject focuses on each part."

Akin differentiates global and local search strategies. The distinction he says is important because, "methods that reside over the total design process have different characteristics from those applied only locally." [5, p.90]. Global search strategies are those that work to resolve a wide range of constraints and which, "can only be detected through the examination of the complete protocol." Global search is further differentiated into depth-first and breadth-first search patterns, which characterize the precedent placed on refining a search versus considering alternative search choices. Depth-first suggests that one considers an option in detail first before considering alternative options. Or, in terms of the PDG, "Depth-first search can be characterized as the allocation of designer's attention to the siblings of a parent node before moving to the next parent node of the same depth in a tree-like search space." [5, p.91]. Breadth-first is the complement of depth-first and suggests that one consider a range of alternatives first before considering any in greater detail. "To eliminate segments of the search space not likely to contain a solution, it is necessary to compare the potential success of each path in the PDG. This is tantamount to considering alternative approaches, or traversing the problem space, in breadth." [5]. Akin notes that depth-first and breadth-first search applied singularly present problems for reconciling varying levels of constraints. It is more realistic to consider that these patterns are always used together in varying degrees.

Local search methods apply in circumstances where the requirements and constraints of a problem are more defined. "Often local methods contain systematic techniques of alternative generation and well-articulated evaluation functions. These methods are suitable for producing transformations in limited representation domains." [5, p.97]. Akin proposes four families of local goal directed search: Generate-and-Test, Hill Climbing, Heuristic Search
CHAPTER 1. INTRODUCTION

(Means-Ends-Analysis), and Induction.

Generate-and-Test is a two part search method that couples an algorithm for generating a design state with an algorithm that tests the state against the goal. If the state successfully meets the goal conditions then some action is taken. The distinguishing characteristic of Generate-and-Test is that the algorithm generates choices based on the criteria that comprise the goal. "In this way, each new instance generated is guided by one (or more) criteria, developing it from a manageable set of instances rather than a much larger exhaustive set of possibilities." [5, p.100].

Hill-Climbing builds on the Generate-and-Test method. "Each newly generated solution is accepted if and only if it represents an improvement over the best solution developed so far. The difference between this and GAT is in the test applied to evaluate the solutions generated. In Generate-and-Test, the solution is accepted if it satisfies the goal of the designer. In the case of Hill Climbing, only the solution better than the best solution developed so far is acceptable." [5, p.100]. In this way, Hill Climbing locates progressively better design solutions than Generate-and-Test.

Heuristic search refers to a family of approaches that employ knowledge about the input data to reduce the Problem Space and thereby facilitate faster, more effective search. An example of a heuristic operator is Divide-and-Conquer, where one decomposes a problem into its constituent parts and then develops solutions for each problem component separately. Heuristic search does not necessarily produce better results, nor more reliable results than what may be produced by other search methods. Newell suggests that one should do a ‘Means-Ends-Analysis’ prior to applying a heuristic operator in order to judge whether it is likely to produce better results than may be obtained through other methods. Akin lists three criteria for determining the suitability of an heuristic operator to a problem: “a) the result obtained in the previous application of the operator, b) the comparison between the improvement expected in the problem state and the operations available to bring them about, c) the appropriateness of the operator in accomplishing this relative improvement.” [5, p.101]. Akin enumerates other heuristic operators, such as Singular Solution, Time Sharing Between Solutions, Decision Hierarchy, Conflicting Unequal Constraints, Most Constrained First, Representation Selection, Solution Testing, Search in Uncertainty, Simulation of Constraint, Hedging the Bets, Sequential Processing, Constraint Hierarchy, Least Promising First, and First Idea.

Akin [5, p.108] remarks that in Newell’s research on problem-solving methods, he defines
the power and generality of heuristic methods as being inversely related.

"As the power of a method increases, he states, it becomes more specialized and its generality of application therefore becomes limited. On the other hand, the most general methods, by virtue of their non-specificity, apply to many different problem situations. ... This suggests that the intuitive design process, as observed in the protocol we analyzed, reply primarily on general methods. The more powerful methods lacking in generality are reserved for special circumstances and consequently not utilized as much as the other two. Following this train of thought, one can speculate that if designers were able to bring well-defined metrics for measuring the success of their partial solutions to bear on the design process, then processes like hill-climbing would be utilized more frequently. Similarly a richer and more powerful kernel of heuristic operators may increase the frequency of use of means-ends-analysis."

Finally, Induction is a method for deriving information from particular facts or instances. Induction may be thought of in lay terms as “If-Then” rules, which require, “a) a mapping between the given data and the predicted data, and b) a given form for this mapping which conforms to the data.” [5, p.108].

**Representation in Design**

Representations are the primary currency of design activity. Akin argues that there are two fundamental modes of representation available to a designer: verbal-conceptual and visual. [5]. The verbal-conceptual mode, “refers to all schemata that make up a representation and that have single specific visual equivalents.” He gives the example of the symbol ‘chair’, which may have multiple visual representations associated with it. Akin then reviews three ‘representational paradigms’ supported by the verbal-conceptual and visual mode distinctions he has proposed: productions, conceptual inference structures, and chunks.

Productions are control structures that represent ‘If-Then’ type condition- action pairings. Productions are typically represented in the form of $A \rightarrow B$, where $A$ is the condition and $B$ is the action. The arrow implies that when the condition exists, or is met, then the action must be taken. When a number of productions such as these are grouped to form more complex patterns of behavior, the group is known as a Production System. In addition to Productions, a Production System may also have rules which govern the flow of control
between individual Productions. Newell and Simon, as cited in Akin, note that, "Production systems, because of their modular structure, lend themselves to parsimony in the design of control structures and have been also demonstrated to accommodate the double memory functions of humans: Long Term Memory and Short Term Memory." Akin argues that the Production System has some correspondence to Long Term Memory in human beings, and can approximate the temporal nature and 'limited span' of Short Term Memory.

Conceptual Inference refers to a broader set of inference operations which do not necessarily follow classical logical but are considered correct nonetheless. Akin notes that Conceptual Inference is applied by human beings in routine tasks such as natural language, problem solving and syllogistic reasoning. Akin explains the structure of Conceptual Inference: "When a conceptual inference is made, this is equivalent to matching the implication of a predicate (an IF-THEN structure) whose truth is known; and the premise of another predicate whose implication is to be asserted. For example, given the IF-THEN structure:

\[
\text{if a balloon is punctured (P1),} \\
\text{then the balloon will pop (I1)}
\]

and the event E ('the balloon has been popped with the pin'), then, the truth of a new implication, I2 (‘then the balloon has been punctured’), can be asserted." [5]. Riger as cited in Akin notes that Conceptual Inferences have a number of properties that make them appropriate for use in design problem solving: they have little goal direction until particular criteria are matched, they bear strong resemblance to the associative machinery of the brain, and their ability to acclimate inconsistent or contradictory relationships and claims. For these reasons, Akin argues that they are, “a suitable tool for use with ill-defined and informal problems.” [5].

Chunks, as we may recall from de Groot, are memory structures that speed matching and recall of patterns with related attributes. This form of human memory structure groups together tokens that have one or more common relationships. “The associative links, sometimes spatial and at other times symbolic, are always stronger between units of information belonging to the same chunk than between information belonging to different chunks.” [5].

Taken individually, each of these representational types — production rules, conceptual inferences, and chunks — model a different functional aspect of human memory. “Production systems model the control mechanisms necessary to simulate goal-directed behavior. Conceptual inferences allow low-level spontaneous information-processing capabilities.
CHUNKS organize the input and output of information and its maintenance in memory.” [5]. However, Akin notes that inconsistencies arise if one attempts to unify these individual representations into a singular memory model. Instead, he proposes to take the representations singularly as they are and apply them for specific functional purposes.

Akin’s empirical research gives insight into how architectural representations are encoded and recalled from memory. He conducts two experiments: the first to examine the nature of chunking in architectural representations, and the second in inference applied during the design process and its effect on learning. With respect to chunking, Akin makes three key observations. The first is that chunks that were recorded represented the lowest level architectural elements represented in the experiment — steps, corners, walls, furnishings — or other identifiable features such as structure or access paths. His second observation is inferred: given that buildings are comprised of parts that are larger than the lowest level elements represented, he postulates that larger memory structures than these low level chunks must exist. Akin gives the example of rooms and clusters of rooms as organizers of inter-chunk relationships, which appear when one aggregates together a number of lower level chunks. Finally, “the accuracy of recall was directly correlated to the type of learning task which immediately preceded recall. The trace task facilitated recall least, whereas the interpret task facilitated recall most. In other words, the accuracy of encoding and decoding in the LTM seems to be a function of the learning task.” [5].

Akin’s second experiment deals with inferential knowledge in architectural design. He examines the application of verbal-conceptual knowledge in the design process and proposes that rewriting rules play an important role in this regard. Akin explains that Rewriting Rules, “a term selected for brevity and from a precedent in cognitive psychology and artificial intelligence literature, are descriptive of the more general IF-THEN structures reviewed earlier. That is, in contrast to procedural information, as in the case of production systems, they simply define equivalency of descriptions or attributes of schemata. For example, if we know that A is equivalent to B and B to C, then we can write B in place of A, and C in place of B or A.” [5]. The modularity of rewriting rules enables them to be chained together to form larger collaborations of rules. In addition, new rules can be learned or generated using the Rewriting Rule pattern. “If we consider the space of all rewriting rules as a pool of premises (P1,P2,...Pn) and implications (I1,I2,...In) then the rules themselves (R1,R2,...Rn) could be presented as causal associations between these premises and implications.” [5]. Akin describes this facility provided by Rewriting Rules as akin to discovery.
1.4 Tools for Amplifying Designer Action

Computer aided design (CAD) refers to a family of computing systems intended to facilitate design activity and amplify Designer action. Specifically, CAD systems may provide functions that enable Designers to develop and model design requirements; conceptualize, document and communicate design ideas; or transmit design data to other systems for further development and use. Schmidt and Wagner [35] argue that CAD models are of central importance in contemporary architectural practice because, “they incorporate, as an ensemble, a project’s trajectory from draft to implementation; they absorb and reflect all decisions taken and changes made, as plans are gradually detailed and modified.”

Ivan Sutherland is generally credited as having developed the first CAD application. Sutherland’s [39] 1963 dissertation, entitled “Sketchpad: A Man-Machine Graphical Interface”, documented the development of a system that enabled the operator to use a light-pen to draw and interact with rudimentary line drawings on a vector graphic display. CAD systems have evolved considerably since Sutherland’s Sketchpad. Early commercial systems of the 1970s through 1990s developed as surrogates for traditional manual drafting approaches, focusing on the specification of the designed artifact through the production of two dimensional vector drawings. These systems introduced new functions such as layers and object snapping to ease drawing production. AutoCAD and Microstation are exemplary in this generation of systems, and are still in wide use today. In the 1990s, 3D visualization began to play an increasingly significant role as the cost of computing decreased and the power of graphics processing systems increased. 3DStudio, form-Z, and Maya for example combine 3D modeling tools with photo realistic rendering and animation capabilities. These applications afford the Designer means to create complex geometric objects quickly, and communicate designs through visually compelling imagery. More recent applications such as Architectural Desktop and Revit have attempted to blend 2D and 3D design production in an effort to speed the production of design documentation. Here again, the focus of this generation of systems lies in the description or specification of the object being designed. Current industry focus is on the development of parametric CAD applications.

1.4.1 Parametric CAD Systems

Parametric CAD systems enable Designers to create geometric models that describe objects and relationships between objects, often driven by parameters. Aish and Woodbury [4]
argue that parameterization is positive in the sense that it can reduce the time and effort needed to make changes, facilitate design exploration, increase the specificity of designs, and aid the Designer in clarifying the conceptual structure of what is being designed. Conversely, parameterization is negative in the sense that it can increase complexity in the modeling task by necessitating a greater level of description and control, compounding the effort needed to simultaneously think through the design.

![Figure 1.9: Spreadsheets are an example of an application of the Systolic Array.](image)

There are two general classes of parametric modeling systems: propagation based and constraint based. [49]. Spreadsheets are an example of a common use of a propagation based parametric system. A spreadsheet comprises a matrix of cells that contain either a value or an expression. Expressions generate values or relationships to other cells in the spreadsheet. We say that a cell that refers to another cell is dependent upon that other cell. A cell that provides a value to another cell is an independent cell. When a change occurs in the spreadsheet, the computer updates the spreadsheet by working from the independent cells with known values to the dependent cells with unknown values. The effect of the update operation is to make data propagate or ‘flow’ through the network of cells. In a spreadsheet, we encounter this when we create a sum for a column of numbers, for example. The values which comprise the column are the independent cells. The cell that totals the values is the dependent cell. When an independent cell value changes, the system recalculates the total by propagating the new values to the dependent cell, thus causing the cell value to update.

Constraint based systems, by contrast, specify the properties of a solution to be found. For example, a Designer may create a line and then apply a constraint to it that specifies
that the line’s length may only fall within a particular value range. When the Designer stretches the line beyond the specified length range, the constraint system is triggered to return the line back to an acceptable value. The constraint system employs discrete and continuous equations to determine which length value the line should return to, and then sets the line to that value.

Figure 1.10: Generative Components, Bentley Microsystems Inc. In Generative Components, models are represented in both symbolic (left) and 3D geometric (right) views. The symbolic view represents the process that realizes the geometric model. The 3D geometric view represents the result of that process.

Examples of both propagation and constraint based systems can be found in commercial production environments. Generative Components [4] is an example of a commercial propagation based parametric CAD system. In Generative Components, Designers model designs by creating instances of ‘Feature’, an application primitive that typically takes the form of a geometric object. Features are organized in a graph that models the process that realizes the design computationally. Nodes in the graph are Features, and links are relations between Features. Relations are created by means of Expressions. Like a spreadsheet, Expressions create dependencies between Features. The system updates the design by working from independent Features with known values, to dependent Features with unknown values.
Geometry that describes the design is generated as a result of the update process.

SolidWorks is a commercial constraint based parametric CAD system. In SolidWorks, designers model designs by directly specifying geometric objects and relations, that is constraints, between objects. As the designer develops the model over time, the relations they add begin to circumscribe the range of valid states for the object.

Parametric models imply ranges of designs. By changing the parameters or inputs that drive a model, different designs are realized. We call the collection of realized and potentially realizable designs the design space. A domain of research known as design space exploration looks at principles and mechanisms for guiding computationally assisted exploration of the design space.

1.4.2 Design Space Explorers

A design space is the set of realized and potential designs for a given design problem. Woodbury and Burrow [51] explain the concept of design space exploration at length, and describe it as, "the idea that computers can usefully depict design as the act of exploring alternatives." Figure 1.11 is a conceptual representation of a design space. Represented designs define the explicit design space. Outside of this region is the implicit design space. When designs are related, we say that a path exists between those designs. Exploration of design space entails representing designs and their interrelationships in a network like structure, and then searching both explicit and implicit spaces for opportunities.

Parametric models are tangible approximations of design spaces. In a parametric model, one describes objects and relations between objects that are governed by parameters. The permutations of objects, relations and parameter values over the model constitute a design space within which the Designer may search. By changing these aspects of the model, different designs within the space are realized.

Woodbury and Burrow argue that there are three primary arguments for the relevance of design space exploration. First, exploration is an effective model for design because it aligns with Designer cognitive behaviors, and therefore suggests an approach for aiding them. Second, computers can provide effective support for exploratory actions and can thus enhance the ability of Designers to represent problem spaces, create and search for appropriate designs. Third, such computing support is feasible and there are effective representations and algorithms that can be used to support design space exploration.

Empirical research reveals that Designers employ a pattern of breadth first, depth next
search when developing designs. [5] Development of designs entails the creation of representations that describe design states. By creating a representation of a design, the Designer makes some part of the design space explicit. Good design representations are both partial and intentional. Partiality means that having a representation of a design does not equate to knowing or saying everything about that design. Designers ‘add and subtract’ from representations as a routine part of their work and, thus, representations convey facets of designs as they exist at particular moments in time. Intentionality suggests that representations, and ultimately designs, are always about something. Designer intent arises in relation to the particular context of a design. As the context changes over time, representations reflect changes in what is being investigated. The history of representations, embodied in the design space path, and the ‘alternatives forsaken’ provide the best explanation of the evolution of Designer intent. “Thus, a move in the design space can be interpreted as seeking a new solution to a particular problem arising in some previous design, where in general the

Figure 1.11: Conceptual diagram of Design Space. The Design Space is the set of realized or potential designs for a given problem. Represented designs comprise the explicit design space. The set of undiscovered designs comprise the implicit design space. When designs are related, we say a path exists between them. Adapted from 2006, Woodbury and Burrow.
representation of the problem and its solution may remain implicit in both the antecedent and consequent designs.” Woodbury and Burrow add that, “Narratives of design intent provided by replay of a thread, and embellished by commentary, themselves provide a basis for comparison and reflection on design possibilities. ... Representations enable comparison to occur, revealing oversights and intangibles that cannot be considered formally in symbolic representations.”

Representations are used in ‘distinct ways’ that make them weak or strong. Strong representations afford the ability to make algorithmic inferences about their referents. Spreadsheets, for example, enable the Designer to model particular relationships between different values, say the cost of materials and labor, and the final price of a product. By changing the cost of labor, the Designer can then determine the consequent cost of the final product. Weak representations aid the Designer by facilitating recall, setting the context of design activity or suggesting new design directions. Scrapbooks and sketchbooks are good examples of weak representation. In them, the Designer accumulates fragments of ideas, interesting findings, and partial designs that they consult over time to inform new work, recollect past thoughts, and reference objects of inspiration.

Finally, representations must aid the Designer in coping with the vastness of design space. Regardless of their specific content, representations are important because they stake out explicit terrain from which further exploration can take place. “This is a corollary of the fact that accessibility is the measure of possibility: designs without physical interpretation or with poor qualities may be the basis for other realizable designs.” [51, p.67]

**CAD Systems as Design Space Explorers**

CAD systems can provide effective support for exploratory actions and can thus enhance the ability of Designers to represent problem spaces, create and search for appropriate designs. Most CAD applications have ‘informal representational qualities’ that make them useful for design exploration. For example, Woodbury and Burrow suggest that solid modeling systems afford Designers tools with enough representational prowess to model and render interesting designs but not enough to overburden them with details. Solid models capture no meaning beyond their solidity and, consequently, Designers are free to interpret the connection between the modeled forms and the objects they represent.

Through the process of designing, Designers reflect on their work and, over time, codify aspects of their working processes. Many CAD systems provide built in scripting languages,
such as Visual Basic or AutoLisp, that enable the Designer to encode design procedures algorithmically. Designer actions can then be replayed in new contexts, reenacting paths and state transformations through the design space to generate new designs.

Constructing new paths is the primary means of exploring design space. Exploration, from any location in the design space or at any point along a path, is informed by knowledge of both the current design state and the states that can be accessed from the path. We say that the implicature of a design state is the set of other design states and their implicatures that can, in turn, be accessed from that state. CAD representations aid the Designer in making inferences about the implicature of a state and, consequently, in determining what other states can or cannot be accessed from that state. Accessing a new state thus generates or extends the path through the design space.

Humans use limited backup strategies due to cognitive limits. As a consequence, Designers typically consider only a small number of alternatives before making a decision. Backups reduce the cost of recovery from mistakes and thereby provide support to Designer exploratory activity. Commercial CAD systems offer limited backup in the form of Undo. Undo enables the Designer to step backward through a historical list of state transformations to recent design states. Similar to the notion of backup is that of recall. Whereas backup is concerned with reverting the current state back to a recent state, recall is concerned with both representing and accessing prior distant states. Designer exploratory behavior, we have argued, builds on known states to create or locate unknown states. Recall affords the Designer with precedents upon which new states may be founded. Because the probability of reusing an entire design is less than the probability of being able to use a small portion of one, recall facilities should be able to work on both wholes and parts of designs. Once a design or portion of a design of relevance has been recalled, it can be applied to the current state. Copy-paste is a well understood metaphor for selecting a portion of one existing set of data and adding it to another. The natural analog to copy-paste in a generative system is ‘copy and apply path’, where a path is a sequence of design states related by rule application or other moves to a start state in a design space. Copying a path copies not the design states, but the moves that generate the state sequence from the start state. Applying a copied path replays the rule applications from a different start state.

Despite the apparent potential of design space exploration, research in this area is relatively scarce. Most research is centered on the design state and on making Designer actions explicit. By contrast, the primary objects of interest in design space exploration are the
space itself, and the paths that realize particular designs within that space. Woodbury and Burrow argue that displays and interactions involving multiple states are the principal elements from which interfaces to design space explorers will be built. Further research is needed in this area.
Chapter 2

Exploratory Research

Increasing change and complexity in the task environment presents a significant challenge to designers. The Morphosis/Yu IFCCA competition entry provides an exemplary model of design process addressing change and complexity in design. We have argued that the IFCCA approach renders the design as a system of interacting parts. Independent Components are morphological types defined through geometric properties and relations. Independent Components interact with other Components in a site, leading to 'formal mutations.' Interactions between Components are guided and organized into scenarios.

The cognitive sciences provide empirical evidence of the limitations of human cognitive faculties. Designer cognitive behaviors develop from those limitations and, in response, Designers have developed characteristic strategies for reducing cognitive complexity when dealing with design problems. The IFCCA design approach employs hierarchy and discrete choice to order the design process and resulting artifact, thereby affording means to reduce cognitive complexity. The design process had different levels of concern: Independent Components; interactions between Components themselves; interactions between Components and their context; and, Scenarios. The different levels of concern create a hierarchy within both the design process and representation that reduces the scope of design tasks and enables them to be executed asynchronously. Scenarios organize the production of interactions around particular factors. As such, they act as an exploration support mechanism, thereby supporting Designer search behavior and making explicit the contingent nature of designing. Finally, there is a clear affinity between the IFCCA design approach that renders the design as a system of interacting parts, and a propagation based parametric CAD system that models objects in interaction and the processes that realize designs.
Design is a process of making proposals for change and design activity is itself characterized by a process of change and development. Supporting and managing that change is of vital importance in aiding designers to realize better design outcomes. Existing parametric CAD applications enable designers to model the objects and relations that comprise designs. However, modeling and maintaining relationships in a parametric model compounds the work of the Designer. Not only does the task of modeling relationships increase the complexity of designing but, when major changes in design direction take place, the work invested in the modeled relations is lost.

A parametric tool that shifts the focus of modeling from the specification of objects to the management of interactions that realize a design would do much to further design exploration while increasing the ability of designers to engage change and complexity in the task environment. The Morphosis/Yu IFCCA design provides us with a clear operational model that can be used as the starting point for research on such a tool.

2.1 An Exemplary Problem in Parametric Modeling and Design

Good problems abstract and simplify complex objects of study from the particular contexts in which they occur. The ‘Cornell Box’ [12], for example, has provided a stable testing ground for more than a generation of research in computer generated lighting and rendering techniques, and enabled means for comparing and discussing very different research approaches around the same problem space. We argue that the Morphosis/Yu project presents a model problem for design. It poses multi-faceted challenges. We approach it through our interest in computer aided design, and propose that its design process can be interpreted as an exemplary problem in parametric modeling and design.

We begin by reconceptualizing the parametric model as a system of interacting parts. The interactions of parts realize a design. Interactions occur within, and are informed by, a particular context or situation. Contexts are interpretations of some part of the total design environment at particular points in time. A design is contingent upon the context within which it occurs. If the context of a design changes, then the design itself may change. The corollary of this rule is that different contexts or situations may lead to different designs. The goal of the system is to aid the designer in not only creating designs but in managing the interactions that realize designs, representing those interactions, and assisting the designer
in her efforts to reflect on the state of interactions and processes leading to designs.

Good problems present multiple research questions. Three broad research themes are relevant to the problem at hand: interaction, search, and learning. Interactions entail both relations and processes that lead to designs. Management of interactions requires the development and evaluation of mechanisms for defining requisite conditions for relations, automatic establishment and maintenance of relations, and the representation of conditions and relational instances in the model. Search is related to interaction. However, where interaction is concerned with the mechanisms enabling relations, search employs interaction to produce variations in relations for the purpose of exploring the design space. Research in search may be directed toward the development of mechanisms for producing variations on designs, specification of goal conditions, selection of designs from sets of variations, and documentation of the contexts informing search activity. Finally, learning builds on search by aiding the designer in reflecting on prior design states, their means of discovery, context and rationale. These themes serve as a point of departure for the research.

2.2 Research Proposal

Extrapolating from systems approaches to organizational management, we hypothesize that the capacity of Designers to deal with change and complexity can be increased by focusing on the management of interactions that realize the design, rather than on the specification of the design itself. We propose a program of exploratory research aimed at the development of a parametric modeling application intended to aid the designer in engaging change and complexity by managing the interactions that realize a design. The goal of the current research is to develop an implementation of such an application that can be used to conduct empirical research toward the evaluation of the working hypothesis.

2.2.1 A System to Support Adaptive Parametric Modeling

We begin by proposing a top-down system design comprising three conceptual and computational entities called Assembly, Behavior, and Scenario; that work together to facilitate design exploration and enable an adaptive parametric modeling process. In the following, we describe Assembly, Behavior and Scenario; then characterize the interactions and associated modeling idiom that realize the system.
CHAPTER 2. EXPLORATORY RESEARCH

Figure 2.1: An Assembly is a composition of one or more elements that together form a 'whole'. The relations that define an Assembly vary along a continuum. At one end of the spectrum, a) an Assembly may take the form of a collection of elements with no interrelations, and at the other b) an Assembly may comprise parts with strong relations. In the latter case, an Assembly can be interpreted as both a parametric object and a system.

Assembly

An Assembly is a composition of one or more elements that together form a 'whole'. Its name is intended to suggest an object that is 'put together' and consequently composite in nature. The role of Assembly is to model the parts of a design. The relations between the parts of an Assembly may vary. At one end of the spectrum, Assemblies may comprise parts that have no interrelationships. In this case, the Assembly is simply a grouping of autonomous elements. At the other end of the spectrum, Assemblies comprise parts that are highly interrelated. In this case, the Assembly can be described as a parametric object, and by extension a system. As such, Assemblies with interrelated parts exhibit system properties such as identity and organization.

It is often helpful to think about or organize parts of a design through some system of classification. In such circumstances, we effectively concern ourselves with the identity of the objects we are modeling. Commercial parametric CAD applications often provide the Designer with collections of prototypical objects, such as walls, doors, windows and the like. Objects in the collection are classified on the basis of their type. Type is tantamount to a specification of properties and structure designating a class of entities. For example, a door
may comprise parts such as frame, panel, hinges, and door knob. The parts of the door may have attributes such as dimension, material and surface finish. An object is a door if it has all of these elements and properties. The system makes a door object by instantiating the type, that is, the abstract specification of the door.

A problem arises though when we start to create doors with new attributes, or when we have doors that are ‘hybridized’ with other classes of entities. For example, a Dutch door is a door that is divided into an upper and a lower panel so that the top can be opened while the bottom is closed. Structurally, it is very similar to a regular door. But, functionally, we might interpret a Dutch door as both a door and a window of sorts. Or, what about a wall or a floor tilted 45 degrees from the ground plane? Is it still a wall, a floor? Or, is it now some other new type of object? In the process of designing, it is not uncommon to design and then redesign parts repeatedly. If, at each iteration, the parts that comprise the door, wall or floor change then our type based approach will continually break. Similarly, if our system includes only strongly typed objects, then its descriptive capacity will be limited, or we will be forced to create unwieldingly large collections of prototypical entities.

An alternative approach to classification can be found in an area of knowledge representation known as Description Logics. In Description Logics, the object and the classification that defines the object’s membership in some class of entities are held separately. On one side is an object of interest to us, and of arbitrary or unknown composition; on the other is a specification that defines various classes of entities. When we want to determine what class of entities the object in question belongs to, the system evaluates the object to determine whether it has the properties that meet the requirements of the class definition, regardless of what other properties the object may have. By separating the notion of type from the implementation of the object, opportunity is provided for the object to change and for it to become more complex, that is, to be a member of multiple classes of entities.

Assembly, as we have described, is intended to be recognized as a composite object, and as such should not be thought of as a type. However, classification is of benefit to Designers. In order to classify Assemblies, and other parts of the parametric model, the system will require two features. First, Assembly instances require either system inspectable metadata that records the instance’s affiliation with one or more classes of entities, or an interface that enables the system to inspect the object at runtime to determine what classes it is a member of. Second, the system must maintain a model describing the entities comprising the design domain; in other words, an ontology. When the Designer queries an object to determine its
type, the system can then inspect the object and compare it against the domain model to determine the object’s class membership. Description Logics are noted here as an example of the strategy of decoupling type from implementation, rather than as a proposition for use of a particular technology. We pose the problem of knowledge representation here to disambiguate the character and purpose of the Assembly class.

Figure 2.2: Behavior enables a model object to respond to changes in its environment. a) The Designer specifies a set of target objects or conditions that the Behavior should watch for in the host object’s context. b) When the condition occurs, the Behavior triggers some action that enables the host object to respond to those changes.

Behavior

A Behavior is a form of agency assigned to an object that enables it to respond to changes that occur in its environment. The role of Behavior is to manage the contingent relations of a design. Ackoff [1, p.664] classifies different forms of system changes, and describes behavior as:

“a system event(s) which is either necessary or sufficient for another event in that system or its environment. Thus behavior is a system change which initiates other events. Note that reactions, responses, and actions may themselves constitute behavior. Reactions, responses, and actions are system events whose antecedents are of interest. Behavior consists of system events whose consequences are of interest. We may, of course, be interested in both the antecedents
and the consequents of system events.”

The operation of Behaviors necessitates four parts: a *host object* that the Behavior is assigned to, a *context* containing elements and conditions to which the Behavior responds, and which the Behavior can query to determine if those elements or conditions exist; a *rule* that describes the set of objects or conditions that the Behavior responds to in that context, and an optional *action* to take when one or more of those objects or conditions are found. Once instantiated, the Behavior observes the context for changes. When a change occurs, the Behavior searches the context for the presence of target objects or conditions that match the specified rule. If targets are found, the Behavior executes the action using the set of matching targets as input.

Behaviors have two effects of interest on the host object that they are bound to. First, we say that Behaviors enable a *loose coupling* between the host and target objects because relations are established by the system only when the rule is satisfied. Likewise, when the rule is no longer satisfied, the relation is removed. The benefit of loose coupling between objects is that it reduces the effort required by the Designer to maintain relations when the context is changing, and similarly, allows the Designer to move the host object to different contexts1 without loss of work in the modeled relations. Second, Behaviors attend to, and implicate some part of the environment in the host object. von Bertalanffy [43, p.228] explains that,

> “the organizational and functional plan of a living being determines what can become ‘stimulus’ and ‘characteristic’ to which the organism responds with a certain reaction. According to von Uexküll’s expression, any organism, so to speak, cuts out from the multiplicity of surrounding objects a small number of characteristics to which it reacts and whose ensemble forms its ‘ambient’ (*Umwelt*). All the rest is non-existent for that particular organism.”

Thus, the target objects specified by the matching rule of the Behavior constitute the *ambient* or *Umwelt* of the Behavior, and by extension, the ambient of the host object to which the Behavior is bound. That the host object begins to incorporate and respond to aspects of its environment also suggests that Behavior transforms the host object from a *closed system* to an *open system*.

---

1Including different parametric models.
Scenario

A Scenario is the combination of a conception of a situation and a collection of elements in interaction, responding to that situation. The role of Scenario is to organize the interactions of elements comprising a design. We say that the environment is the totality of elements and conditions in which design activity occurs. Scenarios are interpretations of that design environment at particular points in time. Scenarios comprise collections of contextual and transactional elements. Contextual elements are that part of a Scenario that can, "neither be influenced nor controlled." [2, p.5]. Transactional elements comprise that part of a Scenario that can be influenced by other contextual and transactional elements, but not controlled.

For example, an Architect may be commissioned to design a new hotel in Vancouver. The environment of the project comprises the social, economic, and political conditions of Vancouver and the world as it exists in 2007. Recognizing that only some of those conditions have relevance to the problem at hand, and in order to simplify the problem of designing, the Architect filters the environment to arrive at some subset of factors that the design can respond to. The Architect may decide that the view from the site, public transportation lines to and from the airport, and local amenities may constitute a relevant context for design work. The Architect then creates designs that explore and respond to those factors; we call each design variant a Scenario.

In one Scenario, the Architect may emphasize exploration of the view from the site, and organize the design so that it optimizes conditions for viewing. In another Scenario, the Architect may study means for making connections between the hotel and public and private transportation more effective. Or, given some development of the project, the Architect may begin to study small portions of the design in detail, looking only at the design of particular spaces or elements within the overall project.

Scenarios may develop top-down, by specifying the properties of the Scenario in advance and then implementing them through a design, or bottom-up, by bringing parts into interaction and then characterizing what is most important about the resultant configuration. Regardless, the Scenario captures some sense of the contingency between the design, that is the particular interactions and configuration of parts, and the context to which the design responds or in which it occurs.

The contextual elements of the Scenario in this example are the 'givens' that can not
be changed — the large scale social, economic and political conditions; the transportation system to and from the airport, and any prior designs that a Scenario under consideration builds upon. The transactional elements of the Scenario are those elements that can be changed or influenced in the current Scenario — the plot of land where the project is built, and the parts of the design modeled by the Designer.

2.2.2 System Interactions and Modeling Idiom

The interactions of Assembly, Behavior and Scenario (Figure 2.3) together facilitate the modeling and realization of designs. The interactions between these three elements are constrained. Scenarios may interact with Assemblies or Behaviors. Assemblies may interact with Scenarios, Assemblies or Behaviors. Behaviors may interact with Assemblies and Scenarios. Following from these basic relations, Scenario assumes the role of top level object. Assembly is a part of Scenario and may itself interact with sub-Assemblies recursively. Behaviors are intermediaries between instances of Assembly and Scenario.

![Figure 2.3: Assembly, Behavior, Scenario interactions. Directed edges represent the possible relations or interactions between the three principal elements of the system. Following from those relations, Scenario plays the role of the top level object. Assembly is a part of Scenario and may itself interact with sub-Assemblies recursively. Behaviors mediate contingent relations between instances of Assembly and Scenario.](image)

The principal source of change in the system is user interaction. Two primary user roles are responsible for changes: Designer and Developer. The Designer is an individual who uses the system to model designs. The Developer is an individual who creates new classes of model objects or extensions to the system for use by Designers. Designer interactions with the system are characterized by four high level activities: 1) creating and managing projects, 2) modeling Assemblies, 3) managing interactions between Assemblies within the context of Scenarios, and 4) managing interactions between Scenarios. Developer interactions with
the system are directed primarily toward the creation of new model object classes, adding and removing model object libraries from the system, and extending the system itself. At the application level, Developers may also be engaged in development of graphical user interface components and interactions. Developer interactions are not addressed in the current research.

Context, as we have argued, plays a crucial role in the creation of designs. Appropriately then, a principle of our system design is that model objects must always be instantiated within, or bound to a particular context. In our system, a context in its most elementary form is a model object that can contain other model objects. To model a design, the Designer begins by creating a context for design work. A customary unit of design work is the project. Consequently, the top level context and the top level model object in our system is called Project. The context of a Scenario is always a Project. The context of an Assembly or Behavior is either a Scenario or an Assembly. Given some context, the Designer may then proceed to model the 'parts' of the design.

Assemblies are the principal focus of design work, and the primary elements of a design. Modeling an Assembly entails first instantiating a new Assembly inside an existing context. Assemblies may be empty, with no parts, or may comprise collections of parts and other sub-Assemblies. When we speak of 'parts', we do so to focus on the hierarchical relation between a set of elements and its parent context; in this case, an Assembly. At times, we also refer to the parts of an Assembly as components, to both focus on their contribution of properties to the Assembly and to highlight the fact that such elements, unlike Assemblies, have no subparts. For example, we might describe a Point or a Line as components if we were discussing their properties, but would refer to them as parts when describing their role in the larger Assembly. The Developer creates components and makes them available for use by the Designer in the system. Components may include geometric objects such as points, lines, planes and solids; and non-geometric entities such as boolean set operators and file import/export filters. The Designer models Assemblies by creating instances of components and relating them together. For example, the geometry of a geodesic dome can be constructed from point, line and plane components. To realize the dome, components are added in succession starting from the center point, to radii, to spars, to the surfaces of the dome. That the center point, for example, must be established before the surfaces of the dome can be defined underlines the fact that relations between components make explicit the process that realizes the Assembly.
In the context of a Scenario, Assemblies may be either contextual or transactional. Contextual elements, as we have described, are those elements in a Scenario that can be neither influenced nor controlled. Transactional elements are those elements of a Scenario that can be influenced but not controlled. In computing terms, we might think of contextual and transactional elements as \textit{read-only} and \textit{read/write} respectively. The system should provide means to allow the Designer to designate which elements are contextual and which elements are transactional within a particular Scenario.

Designs are realized by bringing model objects into interaction. We take the term \textit{interaction} here to be consonant with the concept of \textit{relation}. However, we interpret the notion of relation as conferring a 'structural' bias that suggests stasis or permanence in the relation itself. During the design process, relationships are in many cases both contingent and transient. When the context of the design changes, or when parts of the design change, relationships may be modified, added or deleted. Consequently, interaction is a more appropriate term because it suggests a variability in the duration of a relation, from the instantaneous to the long term.

As we have described, interactions within the system occur at three levels. The first level is that of interactions that realize the parts of the design, that is, Assemblies. Within the context of an Assembly, the Designer creates relations between Components in order to model or realize geometric forms. In some cases, relations are driven by parameters and imply ranges of forms rather than a single end state. The second level is that of interactions between the parts of the design. The operation of a car, for example, entails the connection of certain parts to one another, such as the motor and the transmission, or the gas tank and the fuel pump. Realizing those connections necessitates not only adaptation of parts to one another, but consideration of those connections in light of the particular circumstances of the design context. In the example of the car, connecting the motor to the transmission requires not only that the mechanical parts of the motor and transmission interconnect physically, but that the pair of motor and transmission be able to fit within the physical space constraints of the car frame. Similarly, connecting the gas tank and the fuel pump can be accomplished by running a flexible pipe from one part to the other. But, that pipe must be routed through or around other parts in the car. The interactions of parts consequently lead to adjustments in the individual parts — what was termed 'formal mutations' in the language of the IFCCA project — and an increasing specificity of parts to one another and the particular circumstances of the context. Finally, the third level is that of interactions
Figure 2.4: Scenario Graph. Scenarios interact by inheriting elements from other Scenarios. Transactional elements are depicted in khaki; contextual elements inherited from other Scenarios are ghosted in white. The network of relations between Scenarios reflects the Designer’s explorations and the changing context of design activity.
between Scenarios. Design work is often executed in an iterative fashion. At each iteration, the Designer studies design possibilities and creates new representations for some portion of the overall design. Informing that work is not only the immediate context of the design project, but the historical context of prior design iterations as well. In our system, Scenarios are used to organize the process of design exploration, and represent those partial designs produced through iteration. Parts of prior design states can be 'included' or referenced into a Scenario as part of its collection of contextual elements. In this way, Scenarios can build on prior design states, information can be shared between Scenarios without the problem of managing change between Scenarios, and changes to elements in one Scenario can be propagated to contextual elements in another.

In certain circumstances, a Designer will want to modify a contextual element in a Scenario, and have that change effect that element only in the current Scenario. In such a circumstance, the system enacts the change by making a copy of the contextual element and inserting the new object into the collection of transactional elements. The system identifies model objects by their names. When the system encounters a transactional object with the same name as a contextual object in the same context, we say that it masks the contextual object from other objects in the context and from the Designer. If the Designer deletes the transactional object from the context, the system then makes the contextual object reappear.

Change presents a significant challenge for Designers. In a parametric model, the Designer models objects, and relations between objects that realize the design. When parts are added, deleted or modified, the work invested in the modeled relations may be lost. Behaviors mediate interactions between parts of the model when there is the expectation that parts will change. Behavior enables the Designer to specify that whenever some condition X occurs, then the system should execute some action Y. For example, a fairly sophisticated Behavior may specify that the path for a fuel line from a gas tank to a fuel pump should always be routed around obstacles. If at some point, the Designer adds new parts to the car model that obstruct the existing fuel line path, the system should then activate the Behavior to update the path. Similarly, when obstructing parts are removed from the car model, the system would likewise activate the Behavior to redefine the path, presumably along a simpler route. Behaviors may be used to mediate interactions in two general cases. First, when two Scenarios interact, and some set of parts from Scenario A are included as contextual elements of Scenario B. As elements in A are added, modified or deleted, the
Figure 2.5: Masking. a) Scenarios may inherit one or more elements of another Scenario as contextual elements. b) Making a change to a contextual object in a Scenario generates a local copy of the object. The copy becomes part of the Scenario’s collection of transactional elements, and hides or masks the original contextual object. This enables the designer to override prior design decisions without compromising other Scenarios that inherit that object.

contextual elements in B would then be modified. Second, when an Assembly A responds or reacts to some Assembly B. When B is present, A would then respond accordingly.

We foresee characteristic patterns of change that will occur between related Scenarios. Three basic patterns of change are development, variation and composition. Development occurs when transformations are enacted on one scenario to produce another new scenario that is closer to the design goal state. The effect of development is to continue a single line of evolution within the design. Variation occurs when transformations are enacted on one Scenario to produce two or more new Scenarios that differ from the original Scenario. The effect of variation is to create a branching point in the evolution of the design. Composition occurs when two or more scenarios are merged to create a new Scenario. The effect of composition is to reduce the number of lines of evolution. Development, variation and composition are not mutually exclusive operations. We expect that in most cases, they will occur in combination together.

We have described the principal objective of the system as that of aiding the Designer in managing the interactions of parts that realize a design. The graphical user interface plays a particularly important role in this regard. The role of the graphical user interface is to provide the Designer with tools for enacting changes on the design model, to make apparent
Figure 2.6: Characteristic patterns of change between Scenarios. a) Development occurs when transformations are enacted on one Scenario to produce another new Scenario that is closer to the design goal state. b) Variation occurs when transformations are enacted on one Scenario to produce two or more new Scenarios that differ from the original Scenario. c) Composition occurs when two or more Scenarios are merged to create a new Scenario.

The nature of those transformations, to see the context of design activity, and to reflect on that design activity and learn over time. Three views are critical for the Designer's use of the system: Scenario interactions, geometric forms realized through the interactions of Assemblies, and the process that realizes those forms.

The technical challenges of implementing the system described are numerous. Two broad challenges stand out: modeling and capturing both the state of the model and the process by which designs are realized, and managing changing relations between model objects. In the following, we report on our research into specific technical approaches for supporting the proposed system functions.

2.3 Modeling State and Process

Traditional CAD applications focus on the state of the model as the principal object of interest. Accordingly, they provide designers with tools for modeling and documenting the geometric entities comprising that state. However, when the context changes, the contingent relations break and the model must be revised to regain its persuasiveness. Propagation based systems enable designers to model the objects and relations comprising a design.
Relations imply and often make explicit the process by which a design is realized. When changes occur in the design context, propagation based systems afford the possibility of modifying prior design steps in order to generate a new design state, thus saving considerable time and effort for the designer.

2.3.1 The Systolic Array

In a traditional CAD application, the system functions like a database, recording the state of individual entities in the model. In a propagation based CAD application, the system records the existence of individual objects, the relations between those objects, and the process that defines the particular state of each. When changes occur to the model, the system recomputes the state of each object automatically. There are many possible system designs that can be employed to support a propagation based model. H.T. Kung and Charles E. Leiserson [22] describe a parallel computing architecture called the Systolic Array. The Systolic Array is a modular computing structure comprising simple Processing Elements, interconnected by Links, and arranged to perform specific computational operations. The modularity and generality of the Systolic Array make it a useful computing architecture for design problems, and appropriate as the basis for a propagation based CAD system.

Kung and Leiserson explain that the term systolic is intended to, “draw an analogy with the human circulatory system, in which the heart sends and receives a large amount of blood as a result of the frequent and rhythmic pumping of small amounts of that fluid through the arteries and veins.” Fortes and Wah explain that, “in many of the first proposed systolic architectures, processing elements alternated between cycles of ‘admission’ and ‘expulsion’ of data — much in the same way that the heart behaves with respect to the pumping of blood.” [15]. The term ‘array’, they explain, “originates in the systolic array’s resemblance to a grid in which each point corresponds to a processor and a line corresponds to a link between processors. As regards this structure, systolic arrays are descendants of array-like architectures such as iterative arrays, cellular automata, and processor arrays. These architectures capitalize on regular and modular structures that match the computational requirements of many algorithms.” [15].

Spreadsheets are an example of a well known application of the Systolic Array, and as

\footnote{That one object takes as its starting point the properties of another implies that the other object must necessarily exist in advance.}
Woodbury [49] points out, parametric systems as well. Individual spreadsheet cells play the role of Processing Elements in the array. Formulas create interdependencies, or Links, between Elements. Elements without any dependencies on other Elements are independent; Elements with dependencies on other Elements are called dependent. If an Element is independent, then it can be processed on its own. If an Element has dependencies, then the Systolic Array looks to the antecedents for input Data Items, then uses that data to process the current Element. Computation across the Systolic Array is sequenced so that execution goes from independent to dependent Elements in sequence. Therefore, changing the values of the independent Elements propagates changes to the dependent Elements. Independent Elements with dependents are parameters of the Systolic Array.

<table>
<thead>
<tr>
<th>Element</th>
<th>Value</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>E2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>E3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>E4</td>
<td>E1+E2</td>
<td>E1,E2</td>
</tr>
<tr>
<td>E5</td>
<td>E3+7</td>
<td>E3</td>
</tr>
<tr>
<td>E6</td>
<td>E4+E5</td>
<td>E4,E5</td>
</tr>
</tbody>
</table>

Figure 2.7: Systolic Array. The Systolic Array is comprised of Processing Elements and Links. Complex computations can be performed by sequencing individually simple Processing Elements in the Array.

Systolic Arrays are a useful computing architecture for design problems because of their ability to be used in both general purpose and special purpose applications, capacity for pipelining and multiprocessing, economy through use of simple Processing Elements, capacity to match granularity to computing tasks, and their natural affinity with parametric systems. Because computations are orchestrated by linking individual Processing Elements, a system that provides means of editing links within the Array enables the user to modify what the Array computes and therefore the purpose of the system. Pipelining and multiprocessing speak to the potential efficiencies of Systolic Arrays. Computational pipelining occurs when, “information flows from one processing element to another in a prespecified order,” so that, “the execution is performed in such a way that the output generated by
one processing element is used as an input by a neighboring processing element.” [15]. Multiprocessing occurs naturally when the Systolic Array contains multiple computational pipelines, thereby enabling computation to occur simultaneously on separate threads or devices. “(Systolic Algorithms) are characterized by repeated computations of a few types of relatively simple operations (read Processing Elements) that are common to many input data items.” [15]. This reliance upon the accumulation of simple Elements to perform complex operations enables Systolic Arrays to be organized into hierarchical units appropriate to the task at hand. Finally, the modularity, mutability, scalability and computation by propagation provided by the Systolic Array makes it an excellent match to the needs of parametric modeling activities.

2.3.2 Propagation System Design

Generative Components provides us with a model that we can use as a starting point for the development of our propagation based system. In Generative Components, there is an apparent relation between the underlying propagation system and the higher level modeling idioms and Designer interactions. Designers develop designs by creating instances of an application primitive called Feature. Features are computing objects that execute an action in the model, such as generating geometric objects, manipulating those objects, importing and exporting data and executing user defined code. Features are identified in the model by a unique name. Each Feature has one or more associated update methods. An update method is an algorithm that specifies how the Feature transforms input data into the intended result. For example, a Feature that generates a line object may be able to do so in a number of different ways: by start and end point coordinates; by start point, direction and distance; by offsetting another line. Each update method produces a line as its result but, each uses a different means of doing so and a different set of corresponding input data. Relations between objects are created by using the properties of one object as input for another. Such relations imply that input objects must already exist in the model prior to their being used. Relations between Features, and by extension, the process by which a design is realized, can be modeled by a graph that describes the interdependencies of elements in the model. The model state is computed by performing a topological sort on the graph to produce a total order, then visiting each Feature in sequence, and calling its update method. The update method updates the state of the Feature, enabling it to be used by subsequent Features in the graph for computation.
CHAPTER 2. EXPLORATORY RESEARCH

We observe a number of parallels between Generative Components and the architecture of the Systolic Array. Features are tantamount to Processing Elements. Update methods are user selectable computing algorithms belonging to the Processing Element. Some update methods require or use optional input data. Input data may include numeric and string literal values, named references to objects and object properties in the model, expressions, collections and function calls.

If the input data is in the form of a reference or expression, its value must be resolved before the Processing Element can update its state. References in the input data create dependencies, that is relations, between Elements in the model. The input data must be stored in the Processing Element so that the Element state can be computed whenever a change takes place, without needing to prompt the Designer to reenter the data. Storing input values is also useful in aiding the Designer to make changes to the data at a later time, and for representation through the graphical user interface.

The model state is updated by first determining the relationships between Processing Elements, that is if Elements have any dependencies. Dependencies between Elements imply a graph. A topological sort is performed on the graph to create an update sequence from independent to dependent Elements. To compute the state of the model, Elements are visited one by one following the update sequence. For each Element, its update method is called. The update method resolves the values of references and expressions in the input data, then invokes the selected update method using those values. If the update succeeds, the system continues by updating the next Element in the update sequence; otherwise, the update event halts. The parts of the propagation system are summarized as follows:

- Systolic Array — A Systolic Array comprises parts in varying degrees of interaction, and represents both the process and result state for some unit of design. The state of the Systolic Array is arrived at by computing the states of the subparts of the Array.

- Processing Element — A Processing Element is a part within the Systolic Array that executes some unit of computation in order to arrive at some desired result state. Processing Elements have one or more update methods. One update method may be set at any given time as the current update method.

- Input Table — Processing Elements may have update methods that require user defined input data. The role of the Input Table is to store input data for computation of the Element state.
• Expression — User defined input data may contain values that need to be translated, dereferenced, or computed to arrive at a value that can be used for execution of the update method. The role of the Expression is to resolve the value of user input data.

• Context — A Context is a model object that can contain other model objects; a Systolic Array, by this definition, is a Context. User input data may contain references to named objects in the model. The role of the Expression is to resolve the value of the reference. A Context is a space where object names are resolved.

These parts serve as the foundation for a propagation based system that can capture both model state and process.

2.4 Managing Changing Relationships

Change presents a significant challenge for parametric modeling systems. Not only does the task of modeling relations in a parametric system increase the complexity of designing but, when changes in design direction take place, the work invested in the modeled relations is lost, further compounding the problem.

We study the process of establishing relationships in a parametric model, conduct research to develop and evaluate alternative approaches to modeling relations, and report our findings. We describe an approach called implicit relational modeling that is enacted by the combination of Behavior and an interface called IContext. Behavior and IContext allow the system to establish relationships automatically when the state of the model meets criteria defined in the Behavior.

2.4.1 Explicit and Implicit Relations

Selection is the first step in the process of establishing relationships between objects in a CAD model. Typically, a Designer will establish a relation by using a mouse or keyboard to select two or more existing objects in the model, then apply a function to generate a relation between those objects. What results is an explicit relationship that binds one particular object or property to another. The system subsequently maintains the relation between the objects. Explicit relational modeling offers precise control and short execution time. However, if one of the related objects is deleted or changed, the work invested in the modeled relationship may be lost. Similarly, if the relationship itself requires changing, the
Designer must visit each object in question, unbind it from other objects, then modify or establish new relations as required. As the size of the model increases and as the complexity of relations grow, changes to the model require increasing investments of time and effort to execute and therefore represent a significant increase in labor over a traditional modeling approach.

Selections may also be defined by other means, such as rules or procedures. In this case, instead of manually selecting the objects to be related, the Designer may define a rule that describes the basis by which relations should be made. When objects in the model satisfy the rule, the computer automatically establishes a relation. Likewise, when those objects no longer satisfy the relational criteria, the computer removes the relation. We characterize this as an *implicit* relational modeling approach, in the sense that the actually formed relations are an implied consequence of the rules expressed and the current configuration of objects.

While both explicit and implicit approaches result in relationships, important differences occur. In an explicit modeling approach, the selection event occurs only once and therefore the objects in the relation must exist in advance of the selection event. In an implicit modeling approach, objects are not required to exist in advance of the user defined selection rule; selection occurs whenever changes in the model trigger a need to update the selection set, and therefore the relations are always up to date. In this way, an implicit approach may reduce the effort of establishing relationships, particularly in situations where a large number of objects are involved. However, this increased expediency may come at the cost of reduced control and significantly increased execution time.

Existing parametric modeling systems provide limited forms of implicit relational modeling. In Generative Components, for example, a Designer could create two objects - a single point (A) at the origin, and a collection of points (B) at some other location - and write a function that generates a line from A to each point in B. When the Designer adds to or removes a point from B, the change prompts the function to update the collection of lines. The generated lines are created on the basis of the rule encoded in the function - ‘from A to each element of B’. The relation between A and B is explicit in the sense that it is hard coded into the function. But, the relation between A and the elements of B is implicit in the sense that as the elements of B change, the corresponding relations will always be updated. Our contention is that the extant facilities for implicit relational modeling are impoverished in comparison to those provided to form explicit relations. By developing facilities for specifying implicit relations, Designer action may be amplified.
2.4.2 Studies in Implicit Relational Modeling

Contingency and change, as we have described, have been motivating concerns in our research. Design is characterized by a process of change and development. Systems that can automatically establish and maintain relationships in a parametric model will do much to enhance Designer action and reduce the loss of work that takes place when changes occur.

Our research began with experiments in object selection, using Bentley Systems’ Generative Components as a working environment. Therein, we developed a tool called the Rule Based Selector to evaluate means of selecting objects using rules rather than manual operations. Encountering limitations in the working environment, we developed our own test bed CAD system and proceeded to extend our prior work. We developed a computing object called Behavior to enact selections dynamically in the environment, and an interface called IContext to enable queries about objects in the parametric model. We observed that Behaviors exhibit a pattern of interaction known as the Dynamic Proxy. We describe our work below.

Rule Based Selector

Selection, as we have described, is the first step in establishing relations between objects in a model. Rule Based Selector enables Designers to dynamically select objects in a model using rules rather than manual operations. It has been developed as a Generative Components Feature in C# with the Generative Components API.

In Generative Components, designs are represented by a symbolic graph that models the process by which geometry and relationships are generated in the application. Nodes in the symbolic graph are operators, and links are relations between the nodes. To employ the Rule Based Selector, the Designer creates an instance of it in the symbolic graph. The Selector takes one or more selection rule statements and an existing object collection as input, then returns references to a subset of the input collection as output. Input rules are entered through the Property Editor. Five selection rules are available in the Selector: Exact Match, Range, Nearest Neighbor, Half Space, and Distance. Rules can match against three basic object properties: spatial coordinates \((x,y,z)\), surface coordinates \((u,v,d)\), and index in a collection \((i,j,k)\). Exact Match takes an explicit location or collection index and returns only those objects that exactly match the specified location or index. Range Match takes an explicit location or collection index and returns all objects that fall...
inside the lower and upper range bounds, inclusively. Nearest Neighbor takes an explicit location for a point in space and finds the nearest elements in the input collection to that point. Half Space takes a plane, or two vectors that designate a half space, and returns all elements in the positive half of the space, inclusive of the boundary. Finally, Distance takes a point or collection as an input set, and a positive distance value and returns those elements of the input set which are equal to or less than the distance away. The result sets from each rule statement can then be further operated upon using logical operators \texttt{AND}, \texttt{OR}, and \texttt{NOT}. If the designer provides improper selection rules or input objects, or if the result is logically null, the Selector returns an empty set. Rules are specified using a functional syntax. For example, \texttt{Px(1.0,1.0,1.0)} uses the Exact Match rule to select an
object that is at coordinates 1.0, 1.0, 1.0. Wildcards may also be entered to create broad matches. For example, \( P_x(\ast, 1.0, 1.0) \) selects objects with any \( x \) coordinate value, and \( y \) and \( z \) coordinates of 1.0.

\[
\begin{align*}
rx([5.0:13.0],[3.0:7.0],[0.0:0.0]); \\
rx([17.0:25.0],[3.0:7.0],[0.0:0.0]);
\end{align*}
\]

Figure 2.9: Selection rules can be used to make complex object selections with relatively little work. Here, the Rule Based Selector is employed to select two subregions of a grid of points. The large dots represent the selected points; the statements at the bottom of the image are the corresponding selection rules.

In developing the Rule Based Selector, three areas for further development became apparent. First, Generative Components provided limited facilities for making queries of the model. Selector could effectively ask questions of its input node in the symbolic graph but, could not query the entire collection of objects that comprise the model, for example. Therefore, providing an interface in the environment to provide much broader querying facilities would aid the process of automating selection. Second, our functional selection syntax, while concise, greatly limited the properties that Designers could match against. Declarative selection languages such as Structured Query Language (SQL), for example,
provide a better model for enacting selections against arbitrary object properties based on logical criteria. Third, selection is only the first step in the process of establishing a relation. Once a selection has been made, some action is still required to instantiate the relation.

2.4.3 Behavior and IContext

Reflecting on our experiences with development of the Rule Based Selector, it became apparent that we needed to have access to the details of the modeling system in order to advance our research further. Consequently, we began work on our own rudimentary, propagation-based parametric modeler. Our immediate research goal was to devise an interface for making queries about objects in the parametric model, improve our selection rule syntax, and couple selection with an action in order to complete the process of establishing relationships. Behaviors are computational entities that assist the Designer in managing contingent relations between objects in a parametric model. Behaviors continue our work in Rule Based Selection and comprise two parts: a selection statement that defines objects and conditions of interest, and a Context that the Behavior observes for those objects and conditions. The selection statement is specified in a SQL-like language, and takes the following form:

```
SELECT [Target] FROM [Context] WHERE [Conditions] [Post-Processing Commands]
```

The statement is comprised of four parts: Target, Context, selection conditions and post-processing commands. Statements always begins with the SELECT keyword, followed by the selection Target. The selection Target is the set of potential candidate objects for matching, and is either *, to designate any or all objects in a particular Context, ObjectName to specify a particular named Target, or ObjectName.* to designate any or all properties of a particular named Target. Object names in Federation must be valid Java identifiers. After the Target specification is the FROM keyword, followed by a named Context. A Context is an object that contains other objects, and is identified by a unique name. Selection conditions begin with the WHERE keyword, and are followed by one or more logical criteria. Currently implemented logical operators include equality (=), greater-than (>), less-than (<), inequality (!=). Operators produce result sets that can be operated on with AND, OR and NOT logical operators. Our current implementation includes only the AND operator. Post-processing commands are used to order, filter, and otherwise modify the result sets following identification of matching objects. For example, a hypothetical command named
LIMIT may be used to limit the number of matching objects in the result set. An example of a complete selection statement is as follows:

```
SELECT * FROM MyModel WHERE x>3.0 AND visible=true OR name='Point01'
LIMIT 1
```

The selection statement is decomposed into a parse tree and examined upon assignment to the Behavior. For the Behavior to be able to operate, the statement must not only be syntactically valid, but the specified Context to which it refers must both exist and be an observable object; that is, an object that extends the Java Observable class. The Behavior in turn implements the Observer interface. If the Context is valid, the Behavior observes it for changes. When a change occurs, the Context signals the Behavior through the update method of its Observer interface. If either condition is false, the Behavior produces an empty result set. Queries of a Context are made through the IContext interface. The interface provides methods to get the list of elements that are members of the Context, and to lookup objects by name. Selections are made, or in other words, target objects are located, by searching through the list of elements in the Context. Object properties are examined through reflection against the conditions held in the parse tree. When a match is encountered, it is added to a target list stored in the Behavior. The target list is updated each time a change occurs in the Context, or when the selection statement is changed. Like Generative Components, relations in Federation can then be instantiated by objects in the symbolic graph that take the target list from the Behavior and use them as input to some other operation.

**Dynamic Proxy Pattern**

Christopher Alexander is credited with having introduced the notion of design pattern. In simple terms, a design pattern is a recurring relationship or interaction. The object-oriented software community has taken up the notion of design pattern because such patterns arise frequently in software development, and because they simplify and bring clarity to the interactions of components in object-oriented systems.

We observe that Behaviors participate in a pattern of interaction known as the *Dynamic Proxy*. The Dynamic Proxy combines the *Observer* and *Proxy* patterns. The Observer
pattern comprises two roles: an *Observable* object and an *Observer* that monitors that object for changes. The Observable object implements a method that enables the Observer to register its interest in that object. When a change occurs in the Observable object, it sends a message to all the registered Observers notifying them of the change. The Observer in turn implements an interface that enables it to receive messages from the Observable object. When the Observer receives a change notification message from the Observable object, it can then take action based on the nature of the change.

![Observer pattern](image)

Figure 2.10: Observer pattern. The Observer registers its interest in the Observable object. The Observable object notifies the Observer when a change occurs to it.

The Proxy pattern comprises three roles: Target, Proxy, and Client. The Target is an object of interest to another object we call the Client. The Proxy is an intermediary object that acts as a surrogate for the target, or otherwise hides the details of how the Target is created or acquired.

![Proxy pattern](image)

Figure 2.11: Proxy pattern. The Proxy acts as a surrogate for the Target object.

A Dynamic Proxy is a Proxy that monitors the Context or Target for changes and, when they occur, updates its relation with the Target and Client automatically. By changes, we suggest that either the object or objects that are the Target themselves change, or the state of the Target changes. Behaviors participate in the Dynamic Proxy pattern in our application.

The Dynamic Proxy pattern comprises four roles: Context, Target, Dynamic Proxy and Client. Figure 2.12 illustrates this interaction pattern. The Context is a collection of objects. The Target is comprised of one or more objects in the Context that match some criteria of interest to the Client. The Client registers its interest in the Target by way of the Dynamic Proxy. The Dynamic Proxy, implemented by Behaviors in our application, observes the
Figure 2.12: Dynamic Proxy pattern. The Client registers its interest in a set of Target objects with the Proxy. The Proxy searches a specified Context for objects matching the Target properties whenever a change occurs in that Context. When a match is found, the Proxy notifies the Client.

Context for changes. When a change occurs in the Context, the Dynamic Proxy is signaled to update. If the Target changes, the Dynamic Proxy informs the Client of the update and relays the new Target set to it. The Client then uses the Target set for some computation.
Chapter 3

Federation Modeler

In the following, we provide a detailed description of the implementation and development of our parametric CAD application, dubbed Federation Modeler. The term *federation* denotes an alliance or union of self-governing states, united by a central (i.e. ‘federal’) government. The name Federation Modeler is intended to suggest the idea of autonomous, ‘self-governing’ objects brought together to realize a larger entity or design.

Federation has been developed in the Java programming language. A number of open source and commercial software components have been employed in the realization of the application, including the HOOPS 3D Application Framework, Netbeans Visual Library and GNU Trove. HOOPS 3D is a graphics toolkit by TechSoft3D\(^1\) that provides a 3D canvas and facilitates the construction, visualization, and manipulation of graphical objects within the application. Netbeans Visual Library is a graph visualization and layout component by Netbeans\(^2\). GNU Trove\(^3\) is a high performance Java collections implementation by Eric D. Friedman and is used throughout. Other third party source code and libraries employed in the development of the application have been credited in the code.

3.1 System Organization

Federation has been developed on the basis of the Model-View-Controller pattern (Figure 3.1). The Model-View-Controller pattern facilitates system development and change by

\(^1\)TechSoft3D - www.techsoft3d.com
\(^2\)Netbeans - www.netbeans.org
\(^3\)GNU Trove - trove4j.sourceforge.net
decoupling the domain model from data presentation and user interaction. We have attempted to implement a strict separation of concerns in the application structure to ensure the viability of our work for future research and development.

ParametricModeller is the main application class. It is responsible for establishing the application environment, building the application frame, and maintaining the Model and associated Viewers. ConfigManager is a singleton that contains identifying codes for various classes of Model, Viewer, and Controller events; in addition to holding static global settings and properties for the application.

Figure 3.1: Model-View-Controller pattern.

ParametricModel is the top level Model object and acts as the parent for all other Model objects. ParametricModel maintains a Hashtable called viewState that stores all Viewer state information, including settings and view objects. This enables the view state to be written to persistent storage so that it can be restored later. All subelements of ParametricModel, that is all Model elements, must implement both the INamed and Serializable interfaces. The SystolicArray class provides a rudimentary implementation of the Systolic Array. The Assembly and Scenario classes extend SystolicArray, adding properties and methods required for the operation of each respective class. The ProcessingElement class provides a rudimentary implementation of the Processing Element. Component objects subclass ProcessingElement. System interactions with these model objects are executed primarily through model object interfaces (Figure A.1).

Federation provides five Viewers to aid the Designer: ModelViewer3D, StackViewer, IContextGraphViewer, AbstractTreeExplorer, and PropertySheet. ModelViewer3D provides a 3D geometric representation of the state of a Scenario, or subpart of a Scenario. StackViewer represents the inheritance relations between Scenarios. IContextGraphViewer

---

4Objects that implement Serializable are also known as Java Beans.
provides a 2D dependency graph representation of the interdependencies between model objects, and by extension, the process that realizes the model state. AbstractTreeExplorer represents the hierarchical relations between model objects. PropertySheet provides a graphical interface for examining and editing the properties of model objects. All Viewers extend the Java Swing JPanel class and implement the Observer interface. Viewers observe their respective model objects, and in most cases, the viewState property of the ParametricModel as well.

3.1.1 Model Interfaces

Interactions between Federation model objects are organized around five interfaces (Figure A.1): INamed, IContext, IGraphable, IUpdateable and IViewable. All model objects in Federation are identified by a unique name, and must implement the INamed interface. We call objects that implement the INamed interface named objects. Other model interfaces subclass INamed to ensure conformity to this implementation requirement.

INamed

All model objects in Federation are identified by a unique name, that is string identifier. Model objects have both a local name and a canonical name. The local name for a model object is its name within the Context it is bound to. Local names must be valid Java identifiers and must be unique within the object’s Context. myObject, Point01, and Name_With_Underscores are examples of valid local names. Canonical names are fully qualified names derived by prefixing, in order, the names of the containing parent Contexts. For example, the canonical name model01.scenario02.object03 tells us that the object named model01 is the root Context, scenario02 is both a subpart and sub-Context of model01, and object03 is a subpart of scenario02. The INamed interface (Listing B.3) provides methods to set the local name for an object, and get both its local and canonical names.

IContext

In Federation, we say that the Context of an object is its parent or containing object. The other children of that parent object are its peers. Objects that implement the IContext interface (Listing B.1) identify themselves as objects that can contain other objects. The
IContext interface provides facilities for adding, removing, locating, confirming the existence of, and enumerating children of a Context. The children of a Context are identified by their names. Named objects must have a unique name within the Context, and must implement the INamed interface. It is the responsibility of objects that implement IContext to ensure the uniqueness of names within the local namespace. IContext specifies a method named lookup that enables the resolution of named objects and property references in a Context. References are of the form objectname.propertyname.

IGraphable

Model objects that require input data from other model objects are dependent upon those other model objects. Dependency relations can be represented by a Dependency Graph. The IGraphable interface (Listing B.2) provides methods to enable model objects and Viewers to query an object through its IGraphable interface in order to determine which objects it is dependent upon.

IUpdateable

Changes to individual model objects or the application environment itself prompt the need to update the state of the entire model, or portions of the model. Model objects whose internal state is dependent upon the state of other model objects, or the state of the application environment itself, have transient states. The IUpdateable interface (Listing B.4) specifies a method named update that is responsible for updating the state of the object. The method returns true if the update process was completed successfully, and false otherwise.

IViewable

We can expect that many model objects in a CAD application will be involved in representing graphical information, such as geometry, images or other symbology. However, not all model objects afford representations that can be displayed in both 2D and 3D Viewers, or even in different types of 2D views. The IViewable interface (Listing B.5) provides a common means for Viewers to query a model object in order to retrieve icons, thumbnail images, visibility state data, and 3D geometry for use in various Viewers. If the object does not support a particular view type, the interfaces makes available alternative representations of the object, such as icons and thumbnail images.
Not included in our listing of interfaces, but important for the discussion of functionalities in our application, are the Observer and Serializable interfaces that are provided as part of the Java utility libraries. The Observer interface provides a method named `update` that takes as its argument a message object from the Observable object. The message can be used to determine the type of a change event. The Serializable interface does not implement any methods but simply marks an object as being intended for serialization, as is required to save the object to persistent storage.

### 3.1.2 Model Classes

Federation models comprise twelve classes: SystolicArray, ProcessingElement, InputTable, Input, ExpressionSolver, Expression, ParametricModel, Assembly, Behavior, SelectionFilter, SelectionCondition, and Scenario. SystolicArray and ProcessingElement provide the foundation for the propagation system. ParametricModel, Assembly, and Scenario subclass SystolicArray and act as the primary modeling objects. ParametricModel is the top level model object, and is the container or parent Context for all other model objects. Assembly models the parts of a design. Scenario models the contingent states of a design. Behavior is a stand-alone class that implements the Observer interface. Supporting the computation of Assembly states are InputTable, Input, ExpressionSolver, and Expression classes. Behaviors are supported by the SelectionFilter and SelectionCondition classes. We describe each of these classes below.

**SystolicArray**

SystolicArray maintains and executes updates on a private collection of ProcessingElements. In principle, ProcessingElements may execute arbitrary computations. However, in the context of Federation, most ProcessingElements will be involved in creating and executing transformations on geometric objects.

The state of the SystolicArray is the aggregate of the states of its ProcessingElements. The state of a ProcessingElement is computed by calling its `update` method. ProcessingElements may have dependencies on other ProcessingElements within the array, or on other named objects in the application environment. ProcessingElement dependencies describe a
partial ordering of the collection of ProcessingElements. To update the collection, the SystolicArray collects the dependencies for each ProcessingElement, then performs a topological sort on the collection to create a total ordering. The SystolicArray visits each element in sequence and, if the element implements the IUpdateable interface, calls its update method. If an element does not implement the IUpdateable interface, it suggests that the element’s state is not transient and therefore does not need updating.

SystolicArray extends the Observable class and implements the INamed, IContext, IUpdateable and Observer interfaces. Viewers and other model objects may wish to be notified when the state of the SystolicArray changes. The SystolicArray notifies Observers through the Observable class notifyObservers method, and sends an update message to the Observer with a change event id. Change event ids are set globally, and are defined in the ConfigManager singleton class. SystolicArray is identified by a name string. The addressable subparts of SystolicArray are the elements of the array. The state of the SystolicArray can be updated by calling its update method. Because the Designer may add or change properties of a SystolicArray’s member ProcessingElements directly, the SystolicArray needs to know when changes take place to its subparts. The SystolicArray observes all its elements for changes and updates the state of the SystolicArray when a change takes place.

**ProcessingElement**

The role of ProcessingElement is to execute user defined or selected units of computation. Examples include generating a geometric Coordinate System, Point or Line object; creating a Plane by offsetting another Plane some distance, or performing a boolean operation on two existing shapes to produce a new shape. ProcessingElement implements INamed and IUpdateable, and comprises one or more update methods, an InputTable, and properties.

Update methods are identified within the ProcessingElement class by means of a Java Annotation, @Update. Java Annotations enable the programmer to add runtime retrievable metadata to various program entities, including methods. The @Update Annotation enables the programmer to mark a particular method within the class as being an update method, and then record a description of what the method does, what its parameters are and what data types are required for each parameter. In other object oriented programming languages such as C#, the method parameter names are available through runtime Reflection services. Java does not yet provide such a facility, so Annotations must be used to record this information manually. The name of the selected update method for each ProcessingElement
instance is stored in a private String called updateMethodName. When the update method is set, a pointer to the selected update method is stored in a field called updateMethod; the pointer is of type Method. The updateMethod field is used by update for invoking the selected update method. However, the value of a field of type Method can not be serialized and must be specified with the Transient attribute. The updateMethod value must be reset whenever the object is deserialized from persistent storage. The IUpdateable interface provides the restore to reset transient variables in the model.

In order for the ProcessingElement to update its state, and perform some useful function, the user must select one of the ProcessingElement’s update methods for execution. When an update method is selected by the user, the update method signature is examined and an InputTable is constructed to hold any required input arguments. For example, if a particular update method adds two numbers together, two user provided numbers or expressions that result in number values must be stored in the ProcessingElement so that the system does not have to prompt the Designer to reenter the values each time a change takes place. The Designer can enter input values through a Viewer such as the PropertySheet (cf. 3.2.4). Figure illustrates the set update method event sequence. When the ProcessingElement’s update method is called, it maps the user defined input values held in the InputTable to the update method arguments and then invokes the selected update method. Once the update call is complete, and the ProcessingElement’s state has been updated, the update method returns a boolean value of true to inform the caller that it has updated successfully, or false if it has not. Figure illustrates the update sequence.

ProcessingElements are always children of SystolicArrays. As such, the Context of a ProcessingElement is the parent SystolicArray and the collection of other ProcessingElements in the SystolicArray. Expressions provided as input to an update method are resolved within that Context.

**InputTable and Input**

InputTable holds arguments for the update method of a ProcessingElement. The table is generated by reading the method signature of the Designer selected update method. For each update method parameter, an Input object is created to hold and resolve user provided arguments. Input resolves the value of input arguments provided by the Designer. When an input value is assigned, it creates an ExpressionSolver to parse the user input. If the ExpressionSolver is able to compute or resolve the expression value, then the Input state is
CHAPTER 3. FEDERATION MODELER

set to VALID. If the expression can not be resolved, the Input state is either set to WARN if there is a syntax error in the expression, or to INVALID if the statement is invalid. Inputs are stored and retrieved in the same order they appear in the update method signature.

Expression and Expression Solver

An Expression is a statement comprising String or Number Literals, Collections, Value References, Function calls, or Arithmetic Statements that can be resolved to a value in a particular Context. ExpressionSolver is a class that aids in building the Expression parse tree and in resolving the value of the Expression. Expression types are described as follows:

- Literals — Literals are sequences of alphanumerical characters that represent string or number values. String literals are sequences of alphanumerical characters delimited by opening and closing quotations marks. Quotation marks that occur within the string must be escaped using a conventional /" escape sequence. For example: "Hello World", "Then I replied, / "Hello World."/
  Number literals represent Integer or Decimal (Double) type number values. Integers are represented as numeric character sequences, preceded by an optional -. character to denote a negative value. Decimal values are represented by numeric character sequences with one decimal mark followed by one or more numeric characters, and preceded by an optional -. character to denote a negative value. Example: 1, -1, 1.0, -3.23

- Value Reference — A value reference is a string designating a named object or one of its publicly accessible subparts or properties. The subparts or properties of a named object may be addressed by using a dot . operator, followed by the subpart or property name. For example, MyObject refers to an object named ‘MyObject’; MyObject.SubpartOrProperty refers to a subpart or property of MyObject. References must be valid Java identifier names. Value references are dereferenced in Context. This means that the system will only resolve the object or property in the ProcessingElement’s parent Context.

- Collection — Collections are sets of values delimited by commas, and encapsulated in opening and closing curly braces. The elements of the Collection must be of the same type, or must be resolvable to the same type in the case of Value References, Functions
or Compound Statements. Example: \{1.0, 2.0, 3.0\}, \{“a”, “b”, “c”\}, \{Point01, Point02, Point03\}, \{substring(“abc”, 0, 1), “b”, Point03.name\}

- Function — Function calls are designated by a string name denoting the name of the function, followed immediately by a set of arguments encapsulated within parenthesis and delimited by commas. Example: MyFunction(arg1, arg2)

- Arithmetic Statement — Arithmetic statements are comprised of unary or binary operators and Expressions. Valid operators include +, -, *, /, %, ^, ( and ). Example: 1+1, (1+1)^2

User arguments are recursively decomposed into a parse tree of sub Expressions. The Expression value is resolved by visiting each node of the parse tree following a pre-order traversal. The means of resolving the value of an Expression varies depending upon its type. In all cases, String and Number Literals are first cast into their respective types, and Value References are resolved by using the lookup method of the ProcessingElement's parent context.\(^5\) Collections, Functions and Compound Statements may then be resolved in turn. Compound Statements, Collections and Functions are partially operational in our current implementation.

Value References create Dependencies between ProcessingElements, or other Named Objects in the application environment. Dependencies are used to determine the update order of the SystolicArray, and for creating visualizations of Dependencies within the SystolicArray.

**ParametricModel**

ParametricModel extends SystolicArray, and implements the IContext, IUpdateable, Observer and Serializable interfaces. It is the top level Model object, and acts as the parent or container for all other Model objects. Subelements of ParametricModel must implement the INamed interface. ParametricModel's add method constrains the type of objects that can be added as subelements to instances of Scenario.

ParametricModel aids in the persistence of state for the Model and associated Viewers. modelParam is a Hashtable field that stores globally accessible modeling parameters.

\(^5\)That is, through the parent Context's IContext interface.
viewState is a Hashtable field that stores globally accessible viewer parameters and individual view instances. Keys for parameters, properties and the like stored in these tables are maintained in the ConfigManager class. Controllers and Viewers observe the ParametricModel for changes to these Hashtable properties.

Assembly

Assembly is an instance of SystolicArray. As such, a number of interface implementations are already provided including INamed, IContext, and IUpdateable. Assembly provides its own implementations of IDisplayable and Observer. The expected default behavior of an Assembly with respect to the IDisplayable interface is to return both rendered thumbnails and geometry of any child elements that also implement the IDisplayable interface. In this way, Assemblies display their states as whole entities, including subparts. The update method provided through the Observer interface listens for deletion, property and update method changes on subelements.

Assembly participates in a recurring interaction pattern called the Composite pattern. Figure 3.2 illustrates the Composite pattern. Assembly plays the role of the Composite element. Component plays the role of the Leaf element. In some cases, Assemblies may also participate in the Dynamic Proxy Pattern in the role of Target or Consumer.

![Figure 3.2: Composite pattern. IContext is a subinterface of INamed, and contains elements that implement INamed.](image)

An Assembly is always bound to a particular Context. The relation between a Model object and its Context is an important facet of the Federation modeling idiom, because the Context informs the design of the object. The parent of an Assembly is always the Context in which it is generated. The Assembly name may be provided optionally. Our implementation generates a unique name for the object automatically if none is provided.
Good practice suggests that one should always name objects appropriately upfront.

**Component**

Component extends the ProcessingElement class, and is in turn extended by the Developer to develop new model objects. Component translates the more general notion of a ProcessingElement into a domain specific entity. For example, classes that extend Component include CoordinateSystem, Point, Line, and Plane. Each of these classes provides its own custom update methods. A Line may include update methods such as LineByPoints and LineByStartEndCoordinates. Like all Model objects, Components are instantiated in and bound to a particular context. In the case of Components, the Context will always be the parent Assembly.

Component participates in the Composite and Dynamic Proxy patterns. As a non-decomposable subpart of an Assembly, Component plays the role of the leaf element in the Composite pattern. In the Dynamic Proxy pattern, Components play the role of a Consumer or Target.

**Behavior**

Behavior implements INamed, IUpdateable, Observer and Serializable interfaces. Behaviors are bound to a particular Assembly or Scenario instance, and observe a target Context for objects and conditions of interest. When a change occurs to the Context, the Context signals the Behavior, and the Behavior in turn searches the Context for target objects or conditions of interest. Targets are specified by a SQL-like selection language (cf. 2.4.3):

```
SELECT [Target] FROM [Context] WHERE [Conditions] [Post-Processing Commands]
```

Behavior may be instantiated without a selection rule however, it can not perform any actions without one. When the selection rule is assigned to the Behavior, it creates an instance of Selection. If the object specified as the Context in the rule statement both exists and extends the Observable class, the Behavior observes it. The Selection instance parses the selection rule and creates a parse tree of matching conditions. Selection filters the collection of elements in the Context to determine which elements match the specified rules. When changes occur in the Context, the Behavior fetches the collection of Context
elements, then checks each element using the Selection filter to see if it matches the target conditions.

When multiple Behaviors are instantiated in a single Context, search overhead becomes highly redundant. An apparent means of increasing performance is to centralize the search process by having Behaviors register their targets of interest with the IContext. The IContext would index elements in the collection, group registered searches into a hierarchy of search conditions, and signal Behaviors when objects pertaining to their registered searches have been added or removed from the Context, or otherwise changed. In this way, redundancy could be diminished considerably and the speed of the system increased.

Behavior participates in a recurring interaction pattern called the Dynamic Proxy (cf. 2.4.3).

**SelectionFilter and SelectionCondition**

SelectionFilter is a utility class that is used to search for objects in a collection that match a set of logical conditions, specified by a selection rule. SelectionFilter parses the selection rule into a binary decision tree of SelectionConditions. SelectionCondition uses the Java Reflection API to inspect an object and determine whether it meets the specified condition. SelectionFilter returns a list of matching objects in the collection.

**Scenario**

Scenario extends SystolicArray, and implements INamed, IContext, IDisplayable, IGraphable, IUpdateable and Serializable. Scenarios comprise collections of contextual and transactional elements (cf. 2.2). All subelements of Scenario must implement the INamed interface. Transactional elements are added using the add method, and removed using the remove method provided through the IContext interface. Contextual elements are added and removed from the Scenario using addContextualElement and removeContextualElement methods of Scenario.

Scenario observes its member elements for changes. When a change takes place, Scenario updates its state by updating the state of its collection of transactional elements. Transactional elements may have dependencies on contextual elements. Therefore, when generating the graph of dependencies, both contextual and transactional elements are included.

Changing a contextual element in a Scenario creates a copy of that element. The copy is
then added to the transactional elements collection (cf. 2.2). We say that the transactional copy *masks* the contextual element.

The implementation of IContext related methods vary from their implementation in other classes due to the need to deal with two collections of elements in Scenario rather than just one. The lookup method, for example, searches both contextual and transactional elements.

In 2D views, Scenario can provide either an icon or a thumbnail representation of the 3D geometric entities that it comprises. In 3D views, Scenario provides a 3D geometric representation of both contextual and transactional elements. Contextual elements are differentiated from transactional elements in 3D views, and visually 'ghosted', either by rendering them in wireframe or with transparency, to differentiate them from transactional elements and to suggest that they are there for purposes of reference.

### 3.2 Graphical User Interface Studies and Implementations

Viewers can be divided into classes of primary and secondary view. Primary Viewers support the central modeling activities of creating and managing Projects, modeling Assemblies, managing interactions between Assemblies within the context of Scenarios, and managing interactions between Scenarios. Secondary viewers provide the Designer with additional controls and alternative views of the state and organization of the Model. Primary viewers in this application are the Dependency Graph and 3D Model Viewers. Secondary viewers are the Stack Viewer, Abstract Tree Explorer, and Property Sheet. For each Viewer, we describe its intended purpose, the controls it provides the Designer, the means by which it generates the view, design studies that led to the current implementation, the state of the current implementation, and areas for further development.

#### 3.2.1 Dependency Graph Viewer

Dependency Graph Viewer (Figures 3.3, 3.4) depicts relations between elements of an IContext in the form of a *directed graph*. Each node in the graph, also called a *visual widget*, represents one element of the IContext. Visual widgets are either 'container' like (Figure A.7) to suggest the object has subparts, or they are 'component' like (Figure A.8) to suggest that the object has no viewable subparts. If the object under consideration implements
Figure 3.3: Dependency Graph, conceptual study of Scenario level view. Thumbnail images depict Scenarios and their corresponding state; links between Scenarios represent inheritance relationships. Text annotations enable the Designer to record her thoughts and label groups of Scenarios. The graph, as a whole, passively captures aspects of the overall design process.
Figure 3.4: Dependency Graph, conceptual study of Assembly level view. Nodes in the graph represent Components, solid links between nodes represent explicit relations, and dashed links represent implicit relations. Behaviors are represented using two nodes. Here, ‘BendLine’ represents the action that is activated when an object matching the conditions specified by ‘TargetObject1’ is found.

the IContext interface, and hence can contain subobjects, then a container-like visual widget is instantiated. The Dependency Graph Viewer checks if the object implements the IDisplayable interface. If it does, it retrieves a thumbnail or icon representation from the object for display in the container-like visual widget\(^6\), otherwise, a default icon is displayed. If the object does not implement IContext, the a component-like visual widget is instantiated. Similarly, the Dependency Graph Viewer checks whether the object implements IDisplayable. If it does, it retrieves an icon representation for the object and displays that in the visual widget, otherwise it uses a default icon. Next, edges representing relationships between elements are added to the view. Elements of the Context that support the IGraphable interface can have dependencies on other objects in that Context. For each dependency, a single directed edge is drawn from the independent node to the dependent node.

\(^6\)The intent here being that the image depicts the contents of the container.
The representation of Behaviors in the dependency graph is a special case. Unlike other component-like visual widgets, Behaviors are represented using two visual widgets rather than one (Figure 3.4). Both widgets are depicted in a reverse colour scheme from other visual widgets in the graph, to make them easier to recognize as part of a Behavior. The first widget represents the Behavior component itself. The second visual widget represents the Target of the Behavior. The Target widget floats off to the side of the Behavior and has the Target name as its label. The Target widget represents that part of the environment that the Behavior, and by extension, the object to which it is bound, responds to; in biological terms, the Target represents the ambient (c.f. 2.2) of the Behavior or host object. The Target of a wall, for example, may be other walls or obstructing objects within a floor. Or the Target of a door may be walls that could interfere with the swing of the door. As such, by representing both the Behavior and the ambient, the graph includes not only a representation of the object itself but that part of the environment (ie. ambient) that is of consequence to the object. Behaviors create a loose coupling between their predecessor nodes and the subgraph of which they are the root; effectively making that subgraph a self-contained module. Distinguishing Behaviors graphically from other components in the graph also makes that subgraph easily identifiable.

The Dependency Graph Viewer provides a number of facilities for interacting with and organizing views of the Model. The first is that the Designer can add *Annotations* to the graph. Annotations in this context are notes that can be placed freely on the dependency graph canvas, and enable the Designer to record her thoughts about a particular Scenario, about the design process, or collect information describing the context of design activity. Figure 3.3 illustrates a dependency graph view of Scenarios comprising a model. In addition to nodes and links representing Scenarios and relations between Scenarios, there are text annotations adjacent individual nodes or groups of nodes – labels such as ‘Design Studies’, or longer descriptive annotations such as the text block immediately right of the node labeled ‘PairedBlockWithAxialVoid’. We envision annotations such as this supporting a simple markup language to enable the Designer to record text or create more elaborately structured documents with links and embedded objects, perhaps similar to a blog or wiki. Uses of Annotations include making notes to oneself about design ideas, critiquing designs, creating lists of work to be done, and collecting references to artifacts of inspiration or sources of data that inform design activity. By Annotating the dependency graph View, the Designer can externalize her subjective state, reflect on what she has produced and
what she was thinking about as she was designing, learn and find new opportunities within the ground that has been covered. As the process unfolds, the dependency graph view will then reflect not only the development of a design artifact but, the development of the Designer's subjective state as well. Second, objects in the view can be interacted with through mouse and button actions. Visual widgets can be placed and moved anywhere on the canvas, thereby allowing the Designer to create adhoc groupings. Containerish objects enable the viewer to 'open' up the object by double clicking it, opening a popup menu and taking further actions by right-clicking it. Finally, the dependency graph Viewer provides zoom and panning controls for navigating the view, which become more important as the complexity of the model increases. Panning and zooming provide the Designer with the ability to focus on particular regions of the graph. We envision additional facilities for filtering the graph view by object properties, states and relations to other objects.

3.2.2 3D Model Viewer

The 3D Model Viewer provides a 3D view of the current state of a Scenario, or of the elements of an IContext (Figure A.13). The view is generated by visiting each element of the current Scenario. If the element implements the IDisplayable interface, the Viewer tries to retrieve a 3D representation of the element and display it. If no 3D representation is available, it continues to the next element in the collection.

Our current implementation supports standard 3D navigation actions such as zooming, panning and rotation about the view center. We envision the 3D Model Viewer playing the role of the primary modeling interface. For the purposes of the current research, no further development was pursued on the Model Viewer.

3.2.3 Stack Viewer

The Stack Viewer (Figure A.20) provides a focused representation of the inheritance relationships for a single Scenario, enabling the Designer to quickly see which other Scenarios effect and contribute to the current Scenario.

The view is created by calling the getParents method of the current Scenario. The method returns a tree of parent Scenarios comprising the inheritance chain. For each Scenario, the Viewer creates and displays a visual widget that provides a thumbnail or icon representation of the Scenario, the Scenario name, state indication, and additional view
controls. The representation is ordered from farthest to nearest in the inheritance sequence, such that root Scenarios are at the top of the stack and the current Scenario is always at the bottom of the stack. Our current implementation is limited to listing the predecessor Scenarios and does not yet provide controls. However, we envision the Stack Viewer providing means of filtering contextual elements from the view, and gaining quick access to predecessor Scenarios; we describe this facility below.

The Stack Viewer was developed through a series of static interface studies. Figure A.18 depicts the inheritance chain as a flat list of Scenarios, ordered parent to child. In both cases, Scenarios are grouped by depth in the inheritance tree, with a hairline dividing them into levels. In addition, the background of the current Scenario is highlighted in order to bring focus to it. The right hand layout employs smaller thumbnails for parent Scenarios in a related attempt to bring focus to the current Scenario. While reducing the thumbnail size of predecessors certainly brings emphasis to the larger depiction of the current Scenario, it also reduces the amount of information in the thumbnail considerably. Background highlighting in this case provides differentiation without loss of information. Figure A.19 illustrates an attempt to develop a tree-like representation in the Stack Viewer, ordered parent to child. To afford space for multiple Scenarios on a level, labels are left out for all but the current Scenario. Background highlighting is employed to emphasize the current Scenario. In the left hand version, inheritance relationships are illustrated by a line connecting the parent and child Scenarios. In the right hand version, inheritance is implied through ordering and the thumbnail size has been doubled to provide more information. While the more explicit depiction of inheritance afforded by the left hand layout is preferable over the implied structure of the right or the prior flat listings, the tree like layout consumes a considerable amount of display space. A more compact flat listing is appropriate given the secondary role of the Stack Viewer.

Interestingly, the flat listing approach prompts one to draw an analogy between Scenario composition and the layer systems common to most CAD applications. Composition effectively superimposes parts of predecessor Scenarios on the current Scenario, creating a layer like relation. Providing a control in the Stack Viewer to toggle Scenarios on and off enables the Designer to control the complexity of the display in the Model View. The flat listing will at times also render a historical view of the Model, with Scenarios ordered naturally from oldest to most recent. However, lacking any particular mechanism or class properties in the model object, there is no guarantee that this is achieved systematically.
3.2.4 Abstract Tree Explorer

The Abstract Tree Explorer (Figure A.21) provides a hierarchical, tree-like representation of the current Model state. 'Abstract' here suggests that the representation provided by the Viewer does not correspond directly with the underlying Model organization.

The tree representation is developed by executing a pre-order traversal of the Model. Beginning with the singleton ParametricModel instance, the Abstract Tree Explorer recurses down into the Model, listing each element by name. Scenarios are displayed using two subnodes, labeled contextual and transactional, to distinguish the object membership.

We envision the Abstract Tree Explorer providing facilities for in place editing of object names, display of groupings, further graphic differentiation of node presentations to distinguish classes of Model objects, cut/paste capabilities to enable the Designer to move Model objects around, and filtering by name or properties. A revised interface may also take a different approach to distinguishing contextual and transactional objects, perhaps by rendering them with contrasting coloured backgrounds. Popup menus bound to each node in the tree would provide object specific editing features. Different menus may be available depending on the specific class or supported interfaces of the object under consideration.

3.2.5 PropertySheet

PropertySheet (Figure A.23) provides a table like property viewing and editing interface. The Designer selects an object for editing through the ModelViewer and AbstractTreeExplorer. The PropertySheet observes the Model to determine if an object is selected. When the Designer selects an object, the PropertySheet gets the selected object name from the Model, then uses the lookup method of ParametricModel to obtain a reference to the object. PropertySheet then uses reflection to determine the class and accessible properties of the object. The object must support the Serializable interface for this operation to succeed.

Accessible properties are listed in a two column format with one property per row. The property name appears in the left column, and the property value appears in the corresponding right hand column. The layout is generated automatically. Our current implementation uses a static layout for each of the primary Model classes; we display the layout based on the class of the selected object. Fields that have white backgrounds are editable; fields that are shaded non-editable. For text or numeric fields, the Designer may enter values directly from the keyboard. Components provided an update method selection pulldown menu. When
an update method is selected in the pulldown, a corresponding input table is automatically
generated. The Designer may enter property values and Expressions in the input fields.
Property values and Expressions are evaluated immediately, and corresponding Viewers are
notified of changes to the Model.
Chapter 4

Evaluation of Current Research

We have arrived at an initial implementation of the parametric modeler described, suitable for purposes of rudimentary demonstration. Given the provisional state of the modeler, we have opted to perform an informal evaluation by way of interviews with experts in the domain. The evaluation will approach the research from three perspectives. First, we understand that any design tool will have to acknowledge its role in addressing problems and challenges particular to the historical period and context that it finds itself in. To this end, we seek to draw out of designers insights into the historical problems they see both design and CAD engaged in, the challenges posed by parametric modeling in current professional design practices, and opportunities they perceive for parametric approaches. Second, given that there is a history of parametric modeling in professional practice, we wish to identify what designers perceive as working best in existing systems and design approaches oriented around them, and if there are impediments to their more effective use. Finally, we wish to solicit direct feedback on the conceptual foundations of the Federation Modeler, the design strategy embodied in its computing approach, and proposed graphical interfaces. As Ackoff suggests, “The performance of a system has two dimensions: the efficiency with which it does whatever it does (doing things right) and the effectiveness of what it does (doing the right thing, its value). These should be taken together because the righter we do the wrong thing, the wronger we are.” [2, p.10]. The intent of our broad evaluation approach is thus to scrutinize our operational assumptions so that we may determine whether we are working on the right set of problems.

Two groups of experts capable of speaking to these concerns have been identified. The first group is comprised of professional designers. In this case we have selected Thom Mayne
and George Yu specifically, both for their work on the IFCCA project and because they are accomplished architects and educators. The second group comprises individuals with specific experience in parametric modeling. For this group, we have approached four Simon Fraser University graduate students who have extensive experience with parametric modeling software and expertise in design. We profile our interviewees, describe the interview process, present relevant excerpts of the interviewee evaluations, and summarize what we learn from each in the following. Partial transcripts of each interview are provided in Appendix A.

4.1 Attacking the Simplicity of a Single System

On the morning of Saturday, February 3, 2007, I visited the office of Morphosis Architects in Santa Monica, California and gave a presentation on my research to Thom Mayne and George Yu. As previously noted, Thom Mayne is the principal of Morphosis Architects, winner of the Pritzker Prize in architecture, a founding member of the Southern California Institute of Architecture, and graduate studio instructor at the UCLA Department of Architecture and Urban Design. George Yu is the principal of George Yu Architects, winner of the Canadian Rome Prize in architecture, and graduate studio instructor at the Southern California Institute of Architecture.

My presentation to Mayne and Yu was in two parts. The first part of the presentation motivated the need for research through problems posed by increasing change and complexity in the design environment. I suggested that the IFCCA project was a model for dealing with such problems and that its clear operational approach created opportunities not only for research but for the development of tools to support that design approach. I provided an overview of the three research domains informing my work and grounded my arguments for the relevance of IFCCA in terms of those domains. The second part of the presentation translated my observations about the IFCCA project, in tandem with other background research, into a proposal for a design tool. Given that the software was still under development, I presented schematic interface mockups and explained the operations of the CAD system from a high level.

Early in my talk, Mayne began to remark about the problem of the 'increment'. By the term increment, Mayne refers to a basic unit from which a design is developed. He suggests that designers are always presented with the dilemma of where and how to start a project. Given the complexity of most architectural design problems, designers often
begin by taking some elementary unit (i.e. an ‘increment’) and multiplying it to make a larger design. The ill-defined nature of design problems makes those starting choices effectively random but, ultimately necessary. Continuing, he argues that the purpose of designing is to transform those initial random decisions into more meaningful decisions that successively begin to locate their rationale in the particular circumstances of a project, “because otherwise, you’re stuck with the aprioriness of the original act, and the whole idea is to get around that aprioriness.” Mayne explains that, at Morphosis, a basic test for their design work lies in the transformation or disappearance of those initial ideas.

“When does the work start to transform so it's no longer literally connected to the first initiating idea? And, its a proof of the power of transformative processes that allow you to produce something that is specific to a huge number of forces, which you can test its aprioriness by the disappearance of those first ideas which were arbitrary, which were just getting started.”

Mayne situates the problem of the increment as part of a larger historical discussion about urban design. He suggests that the problem of the increment is related to the problem of the organizational basis of design approaches. Mayne argues that, “The early Team Ten guys, they already located the problem ... They located the problem and realized that it was a matter of attacking the simplicity of a single system. They both started working with multiple systems and looking for ways of conditionizing the system. ... everyone still has the problem of the increment. The beginning increment becomes so dominant.” The implication of Mayne’s comment is that single systems are challenged in dealing with the complex problems that characterize urban environments, and what is needed are approaches that entail multiple systems or, in other words, multiple increments.

Toward the end of the discussion, Mayne drew a diagram depicting a plot of land belonging to a project currently underway at Morphosis. He explained that the project requires a certain number of parking spaces and, given the size and shape of the lot, two levels of parking spanning the length of the property would be needed. Continuing, he explained that different architectural programs\(^1\) have different performance criteria for parking associated with them. Residential parking needs to be much more compact because there is a limited

\(^1\)In an architectural context, the term ‘program’ refers to the type of occupancy or activity taking place in a space.
distance people are willing to carry their groceries. Commercial parking has fewer restrictions because people are more willing to walk farther in those circumstances. Mayne draws a diagram depicting the shape of prototypical residential and commercial parking structures (Figure 4.1). He then draws a third shape depicting the residential and commercial parking structures hybridized together, and comments that this approach is what they hadn’t done in the IFCCA project. The brief discussion is of interest to the issue of the increment because the hybridization depicted in Mayne’s sketch effectively erases the increments of the design, that is the original residential and commercial parking shapes, and we are left with a new ‘whole’.  

Figure 4.1: Sketch of a hybridized parking organization for a project in Las Vegas. a) Parking diagram for a residential block. b) Parking diagram for a commercial or office block. c) A hybridized parking block that combines the performance criteria of both housing and commercial programs. Adapted from sketch by Mayne. (2007)

During my summary of observations on the IFCCA project, Mayne disagreed with my interpretation of the genesis of Independent Components in the project. My statement describing the Independent Components intended to suggest that the Components were ‘systemic’, and that we could infer this from the fact that they exhibited the system property of identity; no implication as to the origin of those relations was presented. However, it became clear that my statement did not provide sufficient explanation to lead the reader to this interpretation. Mayne explained that the Independent Components were, “a very generalized analysis of typology and functionality ... that was directed toward the diversity required for various functionalities that they have in the city – housing, industry, transportation, special events, a few others that they have; they are not arbitrary.” Yu subsequently characterized their origin in slightly different terms, suggesting that the Independent Components were abstracted from the site itself. For example, Noodles mimicked the shape of the entry ramp to the Lincoln Tunnel; Conquistadors were apparent abstractions of the typical skyscraper. Yu remarks that, “We are trying to get away from typology because its a conventional categorization, to morphology which means that we include typology but
there is a kind of freedom in terms of choice, in terms of the behavior of systems that are driving the morphology.” Yu’s explanation about the connection between the site and the Components was poignant — the city itself provided them with the increments from which to build their work. The remark grew increasingly relevant as the conversation continued.

Modeling designs parametrically creates a number of challenges for the designer, not the least of which is knowing what the particular relations are between parts comprising a design. In the second half of my presentation, I described, in very general terms, the organization of Federation Modeler and its concomitant modeling idiom. Parametric objects in Federation are known as Assemblies. Creating an Assembly in Federation is similar to creating a Feature in Generative Components: designs are realized through incremental and successive transformations of some set of atomic components. Again, the difficulty of this approach to modeling is that it requires the Designer to have knowledge about the relations driving the design in advance. Natalia Traverso Caruana, a staff member at Morphosis Architects, related her experience in working on the design of an interior atrium screen for a project at the Cooper Union in New York. She referred to the screen as a ‘mesh’ in her discussion and explained that the idea for the mesh developed incrementally. At first, the inclination was to panelize it and build it with a perforated material. As the idea about the mesh developed, Traverso Caruana changed her mind about what materials to use and how it would be assembled. She thought about using gypsum wall board and then switched to the idea of using glass fiber reinforced gypsum (GFRG) instead. With that decision behind her, she began to study the particular assembly patterns for the material. She tried different variations on the design. The exercise was complicated by repeated design changes to the surrounding areas. Traverso Caruana stated that after various attempts, “I finally discovered what I wanted, which was a series of very fixed horizontals, which create C-shapes all the way around.” She explained the underlying rules:

“And, now I have the rules. I know the rules. I know that the rules are these horizontal bands. I know that the rule is that it goes from this point, to this point, to this point, which we figured out was ... you know ... we ended up putting fifty points on each of the horizontal rings and then joining point one of ring one, to point two of ring two, to point three of ring three, and that’s how you create the spiral.”

Two points are key in this discussion. The first is that the initial decisions drove much
of what followed, and changes to those decisions entailed significant downstream changes to the design. A wall made of gypsum wall board is constructed to a different standard and in a different way from one made with metal mesh or GFRG. As a consequence, the underlying relations are very different. The second point is that the exercise of designing led to the rules that governed the design. With the rules in hand, she acknowledged, the mesh could be modeled parametrically and tied to the shape of the floor openings. When changes occurred to the floor designs, the mesh could then be regenerated automatically. However, the need to know something in advance about the relations driving the form frustrates the effort to construct the parametric model. Consequently, she suggests that in existing parametric modeling systems it takes longer to model the form parametrically than to model it directly.

On the whole, Mayne and Yu made few comments on the proposal for the parametric modeler itself. Given the schematic nature of the material presented, this is not surprising. Mayne acknowledged late in the conversation that he inferred that I was interested in 'multiple forms', that is, the interaction of multiple organizations or objects: I agreed. He suggested that my working process, as he understood it through my presentation, was very analytical and that I was perhaps working, "on a fixed course." He conveyed to me an important working process at Morphosis as that of attempting to solve a given design problem as quickly as possible but, without need to resolve all the particular details of a solution. I infer the principal intent of such a process as being that of speeding learning, and that by reaching some kind of goal state as quickly as possible, a Designer is afforded the opportunity to see the whole of the problem and to make judgments on that whole. I would simply respond that my presentation did not highlight or attempt to explain my working process, or how I arrived at the design for Federation. Nonetheless, I found the criticism compelling.

4.2 Questions, Criticisms and Opportunities for Further Research

On the afternoon of March 27, 2007, I met with Victor Chen, Maryam Maleki, Cheryl Qian and Roham Sheikholeslami of Simon Fraser University, and gave a short presentation about my work on Federation Modeler. I began my presentation by explaining that the purpose of the meeting was for me to solicit feedback on the practical and technical issues surrounding the use of parametric modeling tools in design practices, and that I hoped we would use the
meeting to compare the conceptual basis of different modeling strategies and interrogate the Federation modeling approach. I described the work done by Morphosis/George Yu on the IFCCA competition, and explained portions of their working process in detail, then suggesting that such an approach could be applied to design projects of different scales with the provision that the basis for such an approach lies in the role of interactions. Continuing, I detailed the role of Assembly, Behavior and Scenario in Federation, and then characterized the modeling idiom that followed from their interactions.

Chen questioned the role of components in the IFCCA design process. He asked, "So, in the real world, what will those Noodles represent? ... I mean, the real scenario, in the real world, what are they going to make them?" I suggested that Independent Components should not necessarily be read as models of actual buildings. Instead, given their schematic description as a shape or form suggestive of an architectural use, it made more sense to conceive of them as being used to capture intent or some idea about the composition of a design.² Given an overall configuration, the Designer could then visit parts in detail to explore means for realizing their particular forms. If the construction of a part could not be resolved for some reason, the Designer could then return to the larger scale organization to reevaluate alternative design possibilities. While we can conceive of Independent Components being used in different kinds of ways, one is led to read a top-down oriented process, whereby an overall configuration of Components is arrived at first and then individual Components are developed in greater detail afterward, based on their specific circumstances. This process is consonant with the breadth-then-depth design strategy employed by many experienced designers. [6]

Sheikholeslami commented that much of the discussion had been focused on the role and operation of Independent Components and Assemblies in the model. He was concerned that there was no equal, or adequate representation of properties and constraints of the site. As he explained,

"What I’m talking about is the site, you know, because the site has some specific rules. Like, the neighborhood of the site, like in this place or this direction you can not put any commercial, residential thing. So its not about the individual, its about the whole thing. So, do they have, first, a master plan, a whole concept? Because, as I see in those alternatives, they are completely

²That is, the mixture of different parts.
different. So that means they have rules for Independent Components, but it
doesn’t seem that they have one certain rule for the plan.”

In a subsequent discussion, Sheikholeslami posed the question of whether it would be
possible to model requirements or rules, as he described them, separately of any particular
design in Federation. By rules, he suggested, for example, descriptions of areas where par-
ticular types of architectural programs could or could not be located, specific requirements
for built forms, or some vision for the whole of a design. In Federation, Scenario is intended
to capture the context of design activity. Modeling the ‘rules’ of a context or site could
be as simple as storing a static list of requirements in a Scenario, or as complex as mod-
eling constraints in the site. Enforcing the rules of a context could entail either an active,
top-down mechanism that enforces compliance with a rule, or a passive system that alerts
the Designer when some condition violates a context requirement. Further study would be
required to determine the most suitable approach.

Sheikholeslami compared Figure 3.3 against Figure A.9\(^3\) and questioned whether the
horizontal orientation of the Scenario Graph in the latter was more appropriate given that
one of its roles was to illustrate the development of a project over time. He suggested that
a timeline was a good model for comparison, and that time lines are customarily oriented
in a horizontal manner. Designers would be more inclined to read the graph as depicting
development in time if it was oriented in a similar fashion. I concurred with his argument and
proceeded to explain that the orientation of graphs in the application had been an ongoing
concern for me. The Stack Viewer, for example, started with the idea that a Designer
accumulates information in the model over time. Therefore, it would be appropriate to
think of development in terms of the addition or superposition of information over prior
information — hence the notion of a stack. However, subsequent discussions redirected the
graphical interpretation of the Stack Viewer into a notion of derivation, and consequently a
top-to-bottom orientation. The issue, however, remains open and needs further study.

Maleki explained that she saw the Scenario Graph representation as not only a means
to aid designers in seeing their whole working process, and understanding its development
over time, but that it, more importantly, provided the opportunity to support team work.
She asked,

“Yes, and if its done by a bunch of people, how can you see that in the

\(^3\)The comparison of Figures arose in a separate discussion on April 4, 2007.
CHAPTER 4. EVALUATION OF CURRENT RESEARCH

Scenario view? Can you find it out, when you look at the Scenario, that this was done by one person, this was done by the other person, and this was done by both of them together? And, also, if any of them can see other people's Scenarios when they are working on them?

... First, as you said yourself yesterday, it would help them know what the other person would require from them. And, I am just thinking that, usually in architectural firms, it is not done by one person. It is team work. But, they are not sitting together mostly, they all sit, working at their computers. So, it would be much easier for them if they could look at the other person's Scenario and see what they are doing.

Stout, Cannon-Bowers and Salas [34] describe the importance of shared mental models to the development of individual and team situational awareness. They argue that, “shared mental models are necessary for effective team coordination because they allow team members to anticipate and predict each other's needs and adapt to task demands in an efficient manner.” Because the Scenario Graph provides a representation of the overall design process and model state, it provides an apparent venue to present information about the authorship of different parts of the model and the status of work in progress on those parts. By seeing not only what has been done, but what is currently being worked on, design team members can better anticipate the needs of their peers and adjust their working processes accordingly to aid them.

Representing authorship and the status of work in progress in the Scenario Graph presents a number of challenges. Authorship and the state information could be represented by color coding graph nodes, providing additional graphic appendages to those nodes to indicate authorship and state, or by overlaying data in an adjustable, semi-transparent layer. If multiple designers were engaged in the development of a project, we could expect that the record of authorship for design work would become quite complex over time. Focusing on current activity in the Scenario Graph rather than all historical information may offer the opportunity to both increase the relevance of information and reduce cognitive load. In either case, the Scenario Graph is an appropriate starting point for asking questions about authorship and current work in the model, and research would be required to determine an appropriate strategy to aid both individual and team work.
CHAPTER 4. EVALUATION OF CURRENT RESEARCH

The ability to write scripts, that is short computer programs, is an increasingly prevalent skill among young designers. Maleki characterized herself as a novice programmer, but described her enthusiasm for the possibility of being able to write small programs that helped in the development of designs. At a recent parametric modeling workshop,\textsuperscript{4} Sheikholeslami observed that workshop participants bypassed modeling functions provided through the Generative Components interface in favor of scripting the same operations directly. He remarked,

\begin{quote}
"But, during the workshop ... your workshop and in the UBC workshop ... people who know programming don't go through the interface and try to find the Feature. If they want to create something that they know how to program or script, they don't even look at the Features. And, Robert keeps say, "Ok, I have this Feature, I have this Feature," but people are trying to ..."
\end{quote}

Given the interest in scripting, one is prompted to question whether it would be more effective to have designers model parametric objects through scripts rather than through a graph-based visual interface. Textual representations are compact, parallel the procedural basis of geometric modeling, and can be used to describe very complex procedures effectively. Graph based representations by contrast become difficult to understand as their complexity grows. The Deutsch Limit \cite{42} is an informal rule of thumb that says once you have more than fifty visual primitives on the screen, the representation becomes too complex to understand. Qian suggests that perhaps twenty to thirty is in fact the practical limit. Beyond that number, a hierarchical graph representation would be required, such that nodes could encapsulate sub-graphs and the Designer could navigate up and down the hierarchy of systems, focusing only on a limited subset or representation of the whole at any one time.

I suggested that Processing could be a model for a scripting based approach that would enable designers to model very complex parametric objects, without the limitations of the graph-based interface. Such an approach would entail the Designer writing source code that extends the Assembly class, then having the system compile the source code, and load the class definition into the model so that the Designer could then instantiate it. Like Processing, a library of utility functions could be made available to decrease the time and

complexity of developing new objects. Maleki offered her concerns that a purely scripting based approach would instead be a detriment to the usability of the modeling system:

“If you tell me, ok I’m an Architect ... look at this Processing ... go make your model. I would never try that because I am afraid. But, in GC you start with the interface. And you start making points and lines, and you see things. And you realize that if I could change this code, write this small function, then I can do other stuff. That’s what grabs your attention, and attracts you ...”

In light of her criticisms, an alternative model can be found in visual interface designers, such as the Matisse GUI builder in the Netbeans IDE, that enables software developers to visually construct application graphical user interfaces while generating source code representations in the background. A Matisse-inspired modeling system would allow the Designer to model geometric objects through the interface while generating source code that described the process for creating the modeled objects in the background. Once the Designer had
sketched out the form through the interface, they would then have the opportunity to refine the object by editing the corresponding source code directly.

4.3 Conclusion

Our research began with the motivating example of the Morphosis/Yu IFCCA competition project. The IFCCA design approach was motivated by the need to address problems of change and complexity in design, and the effect of that approach was to render the design as a system of interacting parts. Drawing on systems models of organizational management, we hypothesized that the capacity of Designers to deal with change and complexity can be increased by focusing on the management of interactions that realize the design, rather than on the specification of the design itself. Recognizing that computer aided design systems are critical to contemporary professional design practices, we undertook exploratory research on the development of a parametric computer aided design system intended to aid Designers by managing the interactions of parts that realize designs.

A top-down framework comprising three computing elements — Assembly, Behavior and Scenario — was proposed. The interactions of these three computing entities enable the Designer to model both state and process, manage changing relationships, explore and realize designs. An initial implementation of Federation Modeler was developed. The current implementation enables the Designer to create rudimentary parametric models composing instances of Assembly, Behavior and Scenario in addition to geometric components such as points, lines and planes. The interface provides the Designer with means to create instances of these principal model elements, build and view the process that realizes Assemblies and Scenarios, and edit a limited number of model object properties. The principal contributions of our system design include the reconceptualization of the parametric model as a system of interacting parts, the modeling of both state and process through a Systolic architecture and associated model object interfaces, management of relations through an implicit relational modeling approach, and the Scenario Graph as a representation of the development and interactions of designs over time.

The system implementation is not yet complete and tested, and additional work is required before empirical testing of the working hypothesis can begin. Behaviors in particular present a number of challenges. First, when changes occur in a Context, Behaviors are notified that the Context has changed, then proceed to filter the Context for objects of interest
and update the objects to which they are bound. Our current implementation uses the Observer pattern to construct a 'poor man's event handling system.' More work should be done to study the event model and develop a more capable system. Second, when multiple Behaviors are instantiated in a single Context, search overhead becomes significant and highly redundant. An apparent means of increasing performance is to centralize the search process by having Behaviors register their targets through the IContext interface. The IContext in turn would index elements in the Context based on registered queries, group registered queries into a hierarchy of search conditions, and signal Behaviors when objects pertaining to their queries have changed. In this way, redundancy could be diminished considerably and search speed increased.

Our experience with the system and feedback from evaluators has brought light to a number of areas for expanded research. First, the Federation modeling idiom is front-loaded in the sense that the Designer needs to know something about the relationships that define an object in advance of being able to model that object. Providing the Designer with means to define or sketch out what they desire an object to be like geometrically and then return later to parametrize it may facilitate modeling and remove the difficulty of having to make significant decisions in advance. Second, we recognize that there is potential to build implicit selection into manual interactions with the system. For example, selecting a source object and then selecting a target object while holding a modifier key could be used to automatically generate an implicit relation based on the target object name. Or, selecting multiple target objects could prompt the system to make inferences about the common properties of the target set, and present the Designer with a palette of conditions to build the relation from. Third, Maleki and Sheikholeslami’s commentary on the design of the Dependency Graph viewer suggests a number of areas for potential investigation, including the organization and layout of the Scenario Graph view to better reflect the development of a design over time, facilities for team work coordination, means for describing or specifying the constraints of a context, and functions to enable the representation of the context of design activity. Finally, feedback from Chen and Qian highlights the need for further research on the representation and development of very complex model objects by either graphical means such as encapsulation in the graph, or through alternative representations such as text (i.e. source code).

Following Mayne’s criticisms of the project’s development process, it is appropriate that more effort should be placed in mocking up interfaces and producing testable use cases in
advance of further programming work. Developing interface mockups in Flash, for example, could provide researchers and developers with means to understand Designer/system interactions in concrete terms sooner rather than later. This would also have the beneficial effect of guiding system development and prioritizing work.

Finally, Mayne's discussion of Team Ten and their identification of single system approaches as a problem for the design of cities is important. Assembly, Behavior and Scenario realize a system that enables the Designer to model designs through the interactions of *multiple* systemic entities. The intent of this approach is to enable the Designer to respond to change and address the complex and particular conditions of different situations. That Mayne identifies multiple system approaches to design as an appropriate response to this historical problem adds support to our work.
Appendix A

Appendix

A.1 Transcript, Telephone Conversation with George Yu

Date: December 14, 2006
Location: George Yu by telephone in Los Angeles, California, USA.
Present: Davis Marques (DM), George Yu (GY).

On December 14, 2006 I had an informal telephone conversation with George Yu about the IFCCA project, the working approach they took, the context surrounding the project and his speculations about future work and research that might follow along the same vein. I did not record the conversation, but transcribed portions of Yu’s remarks as we spoke. My questions and commentary were not included in the original transcription. To guide the reader, Yu’s commentary has been organized into themes.

Approach to the Competition Design

GY: Different characters exist in each scenario. The designs respond to different possibilities for development in Manhattan.

One of the main factors is that the Penn Station yards would be redeveloped into green space. Manhattan needed a public park. The park would be publicly funded, meaning that it would be paid for by government sources, in the form of bonds. The City would eventually make its money back from taxes and from increased land value. The park would be built on top of a plinth. The plinth would have parking facilities underneath that could be public as
well. Other public infrastructure might occupy the plinth as well. Each scheme starts with the plinth, and takes its form as a given.

Each of the subsequent additions to the development are a response to different levels of investment in building costs, floor area ratio, total daytime population, and total floor area. Each scenario has a different weighting of these variables, targets. In reality we can’t predict what the appropriate floor area ratio would be. For example, in scenario 15 there are a series of bar buildings on the west side. These buildings are on a line skewed from the adjacent grid and have office and commercial/institutional programs.

The first set of characters are the bar buildings on the north side. In scenario 15 we have a middle density. The maximum condition is scenario 28, the minimum is 15 in terms of total floor area. The land along the north side is divided into a series of parcels. We assume it wouldn’t necessarily be one master landlord.

The second set of characters are the Noodles on the south side, that grow out of Chelsea. The density responds to the height restriction. Noodles bridge from Chelsea into the Penn Station yard area. This is a way of reinventing zoning laws, which are currently all extrusions from plan.

Our proposal was that, given new tools for visualization, we could zone or create more sophisticated envelopes to not only extrude straight up but to leap across streets, to twist and turn around the grid and really make a much more responsive and organic zoning model that allows for something like a Noodle, like integration of solid and void. So that’s what’s really behind the Noodle form. The point is that there is a kind of complexity and the complexity is allowed because of this zoning model.

It’s also important to know that it doesn’t all happen at one time. Forms are added over time and designed to ensure air and light can still enter.

Scenario 28 has the highest density, followed by 15 and 33. Scenario 33 shows a series of Missile buildings that shoot up from ground through the interstitial void spaces. Scenario 33 also has Noodles going further to the east.

In scenarios 15 and 28, there is a new Madison Square Gardens on the corner. In this scenario, Madison sells their existing land to a developer, then a Conquistador takes over the existing site. Madison Square Gardens is then moved west to a new site.

Next major piece ... Conquistador on the Hudson river. In scenario 33, there aren’t any pier-like buildings. In scenarios 15 and 28 there are recreation buildings, and artificial beaches along the waterfront.
The next major piece is the Snake, which is built over top of the entry to the Holland Tunnel. The Snake takes on the form of the air rights over the ramps that direct traffic below ground to the tunnel.

The Jacob Javitts Conference center plans to expand. There was an assumption that it would expand into the park area. There wasn’t a great deal of consideration made of the expansion. In all three scenarios, you have different types of forms for that expansion to occur.

**Competition Constraints**

GY: The IFCCA presented a completely open program. From the beginning, Thom, Marta Malé-Alemany\(^1\) and myself decided to use it as an opportunity to try out new parametric modeling software. At the time, everyone was using form-Z for their work and Maya had just come on the market. However, trying to learn how to use Maya and execute the project was too difficult. The learning curve was just too high to employ it. Instead we produced a Flash based animation with sliders that could be varied but, everything was pre-scripted. The approach was about rethinking zoning, and city planning as a science that is left open ended. Characters represent different morphologies. Most of the design work was done in the computer except for the first conceptual models which were done with foam. We then switched to form-Z, produced three virtual models and then started rendering all the different views.

**Motivations for Research in the Competition**

GY: For all of us, the initial kind of experiment with visualizing growth using new software that would be useful for this kind of large scale development, but ... you know, what endures and what is still part of our work ... I’m convinced that this is really the only way that you can think about the future because the basic difference is that the Jeffersonian grid of 200 years ago is based on a set of values that the founding fathers of America had that were based on the idea of democratizing land ownership.

For example, prior to the Jeffersonian grid, in Quebec development by farmers was based on their adjacency to water ... but by the time Jefferson democratized the world, farming all across the US became a grid. Everything is on a grid, it doesn’t matter if you get a river,

\(^{1}\)Malé-Alemany was an architect with Morphosis at the time of the competition.
creek or lake on your land, it's not based on adjacency to a resource ... it's based on an abstract grid that has as its priority an equalization of ownership. The other one is about ensuring everyone has access to a resource.

The Jeffersonian grid did served us well, and it did change the value of all these resources but, its also kind of archaic in that is doesn’t necessarily represent what we can do, what we want to do, our human nature or values. It’s not that we don’t have tools to do more sophisticated arrangements. There is a desire to return to a system that is more responsive and relational, and looks at surface of the Earth in less abstract ways. That looks at what we need and how you would actually describe it in terms of geometry and makes it up right there and then. That’s why these morphologies are important and each has their own characteristics, properties and values. Their properties are based on their particular circumstances.

**Future Research**

GY: The only way to really refine it is to put it into some real world context and that is difficult because it takes so much political will to change. Its almost out of the purview of architecture. In order to test something like this you’d have to have your own island.

At the architectural level ... it is more likely. I have taken a step back to shopping centers. With Shop-Lift we were definitely pursuing the same ideas and values at the scale of fifty acres. That’s possible. Mixed use urban form is something that a single developer can take on, finance and govern, kind of self-govern, within existing rules and regulations. That kind of open ended growth ... to be more specific, its about an order that doesn’t necessarily come out of one rule. No single event decided show the city would be formed and its this kind of evolutionary or sequential growth that the order emerges out of, based on this step by step growth. And the steps are not building blocks ... which is what it has been for hundreds of years.

**Teaching About the IFCCA Project & Approach**

GY: I would teach them what is and what could be, and not just in architectural terms. The whole concept of order is so ingrown. That’s why people are so intrigued by biological systems ... plants, etc. Looking at those existing systems in nature as potential influences. The trick being you have to then get the students to recognize what is commercializable.
Buildings will always be buildings for 8000 years – they have been physical facts.

The question is how can you use simple, available tools to make people imagine the world in a different way, even if they are still steel, concrete and glass structures. But it does have to do with order, and less to do with aesthetics. Even a guy like Frank Lloyd Wright was definitely on the right track with Broadacre City ... It would be more interesting to arrive at that form without the aesthetic label.
A.2 Transcript, Evaluation by Thom Mayne and George Yu

Date: February 3, 2007
Location: Morphosis Architects, 2041 Colorado Avenue, Santa Monica, California USA.
Present: Nikki Chen (NC), Davis Marques (DM), Thom Mayne (TM), Aleksander Tamm-Seitz (AT), Natalia Traverso Caruana (NT), George Yu (GY).

On the morning of Saturday, February 3, 2007, I presented a slideshow of my work in progress to Thom Mayne, principal of Morphosis Architects, and George Yu, principal of George Yu Architects. The following is a partial transcription of the presentation and subsequent discussion.

Tape 1
[08:58 - 18:34]

TM: There are some obvious things that took place when you showed the scenarios. It became evident some time later there was a really obvious problematic in that they were basically more or less similar, because in fact a normal audience looked at them and saw them as the same. ... I realized that it was contaminated. In both the academic work and here, we're trying to deal with contamination of the various designs ... formally ... constructs of the original pieces and in that case that some of them were so strong, visually, the worms especially. That, to the normal eye, and to a lot of normal eyes that are architects and planners, they couldn't see the difference. They said that after all these words, the three schemes they are identical. Meaning, it would take a much finer grain to look at but, the way we're showing it, they are more or less the same scheme. Now there is more evidence to me. We could have done that. It would have been a much more powerful project. It was incredibly simple. If all we did was turn off certain ... because we had the morphologies that we started with that were connected to morphologies ... flat things, tall things ... if we made some simple connections of types and their uses. You're stuck with making those first decisions ... because we had 13 of the 14 or something ... and those three, if we had just shut off some ... like one of them should have had ... the worms should have just disappeared. Or, some of them should have had some vast empty spaces because we're looking at that now and realizing it should be produced at a really basic level.
You want to just look at it and just see radically different alternatives ... meaning there is one that has huge intensification in one place and emptiness some place else, some which distribute everything equally, and then you immediately get broad urban differences at an urban level. And, in ours it got contaminated at an architectural level, do you agree?

GY: Yeah, it was too subtle.

TM: Yeah, and I can see that as you make those first pieces, the building blocks become incredibly important. I was looking at a thing I got ... about the Venice Biennale from the Berlage, and they had a project where they used different increments ... in fact, the thing you just did, all of you. ... I was telling John that his was way too unrelenting to contingency, and it killed him. It was aligned to Bakema, Van Eyck ... The early Team Ten guys, they already located the problem ... I'm reading, working on that right now. They located the problem and realized that it was a matter of attacking the simplicity of a single system. They both started working with multiple systems and looking for ways of conditionizing the system. And they were looking at indigenous architecture and all of them went off ... and thats when Gropius, a lot of them, started doing really weird work. ... and then I said, George was already was a step ahead of you because he already realized, one, because there was more than one system, and two although you were operating at different scales there was somewhat ... although in a simple way, in either one of your grids ... there was a play between that order and other orders of nature that had given into ... so finally the grid disappeared into the hill. But, there is a dialog that makes these photographs so interesting because, in fact, that you are reading this natural system versus the grid and it is the interaction that is so powerful, either one of them ...

GY: It is the merging.

TM: Right, it is the merging ... it is actually the dialog that is so powerful ... and the grid itself is only allowing it to happen. The grid itself is not that interesting. Anyway, his was just ... its actually a CIAM discussion, which was completely dumped by Team Ten. Now we're going back to mid 1940s.

The point, for all of you ... everyone still has the problem of the increment. The beginning increment becomes so dominant. It seems as though somehow, as this process

---

2Referring to a recent design competition on the Discovery Channel that Yu participated in and Mayne judged.

3Congrès International d'Architecture Moderne (International Congress of Modern Architecture)
becomes more sophisticated, somehow its going to attack the increment at some point and change the increment itself. Do you think?

GY: I think so.

TM: Because otherwise, you're still stuck with the aprioriness of the original act, and the whole idea is to get around that aprioriness. And, in a lot of the schemes ... the one I was looking at which was close to your's and John's ... they started planning operations. Now the words were right but finally its so simple because you just see this repeating increment and the increment was the most arbitrary part of the whole project. Its the part you just have to make up and say its the shape. And, he wasn't able to somehow ... somehow the thinking wasn't able to develop to a point which somehow challenged and transformed the original thing, which I've always used as a model and a basic test for our design work. When does the work start to transform so its no longer literally connected to the first initiating idea and its a proof ...

GY: it transcends ...

TM: ... of the power of transformative processes that allow you to produce something that is specific to huge number of forces, which you can test its aprioriness by the disappearance of those first ideas which were arbitrary, which were just getting started. And, the less it does that ... either we're lazy or we were just incredibly insightful in our first instinct, which is very seldom. It defines it as a simple problem if you can operate that way.

GY: But it means going back to those two slides ... the Jeffersonian grid versus the Quebecois grid ... which is that there are a huge set of values that are built into the existing order of the resource based versus the democratic distribution of land. The Jeffersonian grid is a 200 year old ideal and it may be bankrupt at this point. And, who knows, maybe the kind of European river side distribution of land which is resource based is on its way back because we have the resources now, computers that allow us to visualize in a client by client way, how to actually use those resources. But, you're right, if you don't go back to first principles you're kind of being lazy. You're not really questioning the beginning point.

[53:37 - 59:14]

TM: The Independent Components were invented through a very generalized analysis of typology and functionality and so we came up with — I forget how many there were ... 13, 14 pieces? — and they were really located in typologies still, but really generalized
APPENDIX A. APPENDIX

typologies, that was directed toward the diversity required for various functionalities that they have in the city ... housing, industry, transportation, special events, a few others that were idiosyncratic. ... they are not arbitrary. They are very simple summations that get a process started, that allow us to deal with ... kind of a convention, kind of understanding this typology versus morphology ...

GY: Its a step above. We are trying to get away from typology because its a conventional categorization, to morphology which means that we include typology but there is a kind of freedom in terms of choice, in terms of the behavior of systems that are driving the morphology.

TM: ... and some of these very quickly adapt to known constructs so that right now we are dealing with various types of housing — bands, matrices, towers — and they already have broader dimensions but we already know they have to operate within those dimensions. So, if you see an office tower, it in fact can have much larger dimensions because of the nature of the functionality, although that is another discussion. Where housing, because it is made up of componentized rooms, and those rooms have similar ... criteria — air and light — that you have maximum distances from the center of a room ... and then all of a sudden you can come up with certain types of these generalized beginning components, but they have characteristics that link them to resolution because you need that ... In the beginning of the process, everything is arbitrary, except what is found on the site, which is a lot, except what exists. And then, as you look at the various components, each of them have attributes connected to their potential connection with some reality. So, freeway, pedestrian movement, greenspace, art scape, landscape, office tower, stadium ... all of those were able to, somehow, start with some notion. So, a freeway is based on the radius of a turn at sixty miles per hour, and a certain width characteristic, and it’s fluid and it has a certain dimension, and it comes in twelve foot increments sideways, on and on. And a pedestrian ... gets easier ... but then its lineal, and its ambulatory.

We’re looking at palm groves right now. Well, its just a dot grid, and its the most literal. So we’re looking at a certain kind of landscape in the desert, and we’re going ... let’s just start with something we know, and we start with a date grove. Well, we can make a grid and now that’s the most maybe literal component of what looks like a very diagrammatic scheme. And ours did the same thing, and some of them became quite architectural, like the worm. Because we put the worm down, and it was still the worm, and it was now in relation, it was becoming a more complicated thing. But the component made up something
that was quite more or less predictable, well atleast to an architect it was. And anyway, the
point is really an important one.

... Its that you start with those beginning components ... I should pull out a journal
on my desk. There’s this project at the Berlage, it does the same thing. ... Your project4,
because finally your project ... they are all these same pieces and finally you never get away
from ... in fact, your discussion of the car is perfect.5 Its not the addition of the pieces, its
finally this other thing. It isn’t the summation of the parts. Those parts could make several
things and they wouldn’t make that car. Especially if you start transforming some of the
parts. If the body was transformed you would get a dragster. You could get very different
kinds of car, with very different kinds of looks. Just by shifting the body and making it
very small, you could come up with a funny car dragster, and you could go on and on.

... Because this seemed to be the problem with you and John, and some of the people6
... it seemed to be where you are stuck with. Am I right?

GY: Yeah ...

TM: ... and you can’t ... you still have the increment. And here it seemed to be so
important to your whole ... your beginning thesis.

Tape 2

[23:30 - 32:53]

TM: Because again, the bar on the side.7 Its just a ... that’s just a kind of a program
... it depends on what you are interrogating, the kind of idea you are trying to explore.
Because we are going to do the same thing. The bar will be what ever we think we need to
be, just to move information ...

GY: The bar of the scenarios you mean?

TM: The .. yeah, well. It won’t be scenarios. Because we’re now done with scenarios.

GY: The history bar.

TM: Now, they’re infinite. Now you construct the scenario and give it its own number.
Because now I don’t even want to do the three ... infinite ... The scenarios are developed

---

4Referring to a recent project by Yu.
5Referring to Marques’ slide of an old Volvo car decomposed into parts.
6Referring to participants in the Discovery Channel competition.
7Referring to the Stack Viewer.
by us at the presentation, or they’re staged like ...

GY: But its more like ... Davis’ suggestion is that its a series of historical steps that ... or layers. Its not really scenarios. Scenarios are made up of combinations.

TM: No, but that’s interesting too. We’re having that discussion. Some things ... what do you put down first? It seems like you put down the one that’s the most stable and that you have the most knowledge of, because you’re forced to do that. Any explorer ... that’s how science works I think. They start with what they know the most about, or what they think they know the most about. Its not that it can’t be destabilized, or that you can’t, in the process recant your initial theory, right? But, in this case, its contingent matter. Its the found site that’s outside your territory, atleast immediately. So, its, in our case, its a freeway and its fixed, and there’s a couple of roadways, and the shape of the site, which are given. ...

But, it actually makes sense. Without that, then you’d really be stuck. So we have a very funny shape of a site, which is completely contingent. ... We have a major freeway, and we have secondary streets. And, one of those is an access road, and that’s it. But those are pretty stable, so those show up first, right? And then it also has to do with different scale relationships which we didn’t get into, but in ours was really important. If you go through the building up of our system. Because one of them, they did the same thing. One of them was the complete map, which was global ... and then inside that was an infrastructure of the rail, right, etc. And there were certain parts that were somewhat radically differentiated by scale. And now it becomes even more important, I’m realizing, in our work that part of the mechanism of the relationships or the process prior to the form has to discuss essences of that process which includes scale. Because that’s how a developer works. If you build up a series of little tiny sites, you’re going to get a very different city, you’re going to get Santa Monica. And each person has the control of that, right? And you’re going to get a certain kind of city. And if you take a whole block, you get a different one. Well, the form ... the first development of the idea is going to be based on the increment of development, it will be the most powerful shift that the Architect will be involved with ... that will be the huge shift, and if you get four blocks it keeps going. And so now you can work back, and then that becomes, when you are interested more and more in the processes than the result. That becomes really essential, the increments.

---

8Referring to the IFCCA project.
9Gestures to suggest a parcel of land.
... Your building ... I don't know what the actual problem is.

But, its ... because then you’re looking at what are the fundamental components of those relationships that do lead you to form making, that are ... you are locating specifically what those processes are and what those interactions are. Because that’s really hard in architecture, because we’re still starting with the ... we can’t start with the interactions, we don’t have that luxury. Like everything, we just model the total amount of square footage and it just shows in blocks and that stuff has to fit in here. And some of it is called residential, and some commercial, and some entertainment, and some parking, and ... and now, that stuff, we have to somehow form that within this site. Its a mile long. Its kind of ridiculous. Its a city, right?

And, so it starts with ... because that’s how we started way in the back ... Because it starts with the objects and then ... and then you start ... the relationships. Because in our scheme its the reverse.\(^\text{10}\) It started still with the pieces and the relationships were a product of our placement of the pieces, but were the study – we were studying the relationships. But they came from the association of the objects, and we did not model the relationships first.

GY: But the pieces, for the most, part did come from the context. For example, the Noodles came out of Chelsey. I mean, the scale of the blocks, the height of the buildings, and the extension northward of that fabric. The long bar came out of the kind of East-West corridor to connect to Broadway. It was also the North, South, East, West kind of cardinal points. The park came from the yards, and so on. So, they were not independent totally, completely autonomous from the context. They were absolutely at one to one relationships in terms of dimension and scale, and function which then took it from typology up to this next level of morphology; because they didn’t necessarily have to be the same functions as in Chelsey.

TM: Right, but we never ... we could only move around numbers of those pieces. And never did the large public piece change radically. It changed in terms of its positioning with other pieces, but in itself never morphed. It could but it just ... a more complicated thing. And then there were pieces, that were just connecting pieces ... the Linkers. That were just meant to mimic existing buildings, that were literally the most traditional pieces, right? That glue together New Urbanism and contemporary thinking, that proves we’re not

\(^{10}\text{Referring to the IFCCA project.}\)
in either camp. That's both about connective tissue and about exploring new modes of urbanism.

GY: Yeah. Some pieces literally, just kind of extruded up from the tunnel ... to the, what do you call it? The Holland Tunnel. And then, that tunnel that's under the East River. What do you call it?

TM: ... Lincoln Tunnel ...

GY: Anyway, so there were very substantial linkages to all of those existing geometries and fabric.

TM: But, see, we intuited a lot of that.

GY: ... yeah ...

TM: They were made by rapid interpreted decisions by people who already had established form languages. I mean there was a lot of implicit stuff going into that. ...

GY: I think in your demonstration, using the various parts. You've got the existing kind of given context, right? The wireframe. But, then there is a kind of distinction that is a little bit naive in that your representation of the Noodle type, or whatever, Independent Component, that are not really discussed as connected to anything. So there is this kind of dichotomy. You know what I mean? There's got to be a connection in all cases between the existing reality on the site, and the form.

[36:09 - 43:10]

TM: If you could deal with relationships, and do what you said in the beginning. I'm not sure quite what that means. It would be modeling the relationships prior to having even things; I don't know how we would do that. Because if you didn't have the stuff – the seven, eight, nine million square feet and we were just modeling the relationships, I don't know how we would quite do that.\footnote{There was some confusion about the meaning of \textit{reduction to dynamics} and the role of relations vis-à-vis modeling and designing in my presentation. Thom's remark here seems to suggest that he has interpreted my proposal as being that designs should or could be arrived at by modeling relations in advance of forms. I should note that reduction to dynamics is presented here as a means to acquire knowledge about existing systems, rather than as a proposal for an approach to realizing systems or designs. The means or method of moving between relations and form remain an open question for me. Despite the confusion, the conversation is productive.}

GY: Its the difference between the different generations of software. Its the difference between using AutoCAD versus Maya, versus CATIA, versus using Generative Systems. Or,
is it Generative Components? And, we've been talking about this. If you use Generative Components, its impossible right now. Because its so much about knowing the relationships before hand. That you are writing a script ...

TM: But the thing you're doing is really interesting. But the thing is, which relationships? Economic relationships? Social relationships?

NT: We did the atrium in this kind of program ... Do you know the Cooper Union mesh? Well, when we were developing this we started thinking that we'd panelize ... you know, maybe a perf panel, maybe ... different ideas. We went from gyp board to GFRG in the end, and we discovered that this is the solution – we wanted permeability, we wanted to see through the space and also create a volume – so that answer was that. So, I started studying, not through Generative Components, just in Microstation, what do I want this pattern to do, what do I want to pattern, how do I want it to look? And we started working on it, and when I finally discovered what I wanted, which was a series of very fixed horizontals, which create C-shapes all the way around ... We wanted that to be fixed so that one was guard rail height, so that one is door height and one is floor height, so that these were quite fixed. And then, instead of having the diamond or typicals that you see over the verticals, I wanted it to stream around, so that it kind of spirals around, that was the idea for the design. And, I modeled this but, stairs changed, the floor plan changes, and it was impossible to be able to remodel, and remodel this mesh again. And, now I have the rules. I know the rules. I know that the rules are these horizontal bands. I know that the rule is that it goes from this point, to this point, to this point, which we figured out was ... you know ... we ended up putting fifty points on each of the horizontal rings and then joining point one of ring one, to point two of ring two, to point three of ring three, and that's how you create the spiral. Now that is an example where having this parametric modeling or scripting is beautiful because we would change the floor plan, so suddenly you change the ring. You press the button, and then it goes ...

... but you definitely have to know what you want before you start using Generative Components because its impossible. It takes you longer to do what you want than to actually ... something changes in the process and you have to do it over again.

[49:00 - 55:05]

12Glass fiber reinforced gypsum board.
GY: ... and then we did a Jusco after that. Which was this big mall, and is actually one of the drawings that is used in the demo. Which kind of took IFCCA and compressed it into a very small, well relatively small – a million square feet as opposed to twenty two blocks in New York. But, you know, it was the same problem that we talked about before ... how do you figure out how to make something that is complex at an urban scale, with the ability to shift and vary over different sites.\textsuperscript{13} Because it wasn’t just that site, Jusco has become the biggest Japanese retail developer recently. It was very fast growing. It started twenty years ago using an American shopping center model. They were looking to us to basically give them a denser model for inner cities. So, it had to work not only in Nagoya and elsewhere, and so we had to come up with not just the kind of singular plan, but a strategy that could work on different sites ...

TM: We were just talking about this\textsuperscript{14} ... We were looking at this site, and if you put parking on it, its two layers for the whole site. But, with parking, it has different performance criteria – if its commercial, or if its office, or if its housing, right? So, housing looks like this – the parking pretty much has to look like this because there’s a certain distance over which you can carry groceries.\textsuperscript{15} If its a commercial center it can look like this because you’re willing to walk, and if its an office block, and so on ... right? So the thing, it seems like what this can allude to about relationships that finally have some implication to form, because we were discussing this, that finally this plus this equals kind of that ... and that’s the trick.\textsuperscript{16} Where does the transformation take place? Because that’s what we didn’t do in ours. Where things glue together. And, I can do it through differencing and adding and subtraction. And, I can do it through deformation of form. ... 

And, its a huge problem because intuitively you’ve got a bunch of people who can do this. Through the human, using your total intricate capability. And, it seems like, even in your definition of simple problems to complex problems, that the simple problems could be

\textsuperscript{13}Yu discusses a previous design commission for the Jusco Diado Park Way shopping mall project in Nagoya, Japan. The project itself was for a single shopping mall however, the intent was that this design would be replicated and adapted to different sites throughout Japan. The challenge of the project was to develop a system that could respond to the diversity of conditions that would be encountered in different sites while maintaining the identity of the design.

\textsuperscript{14}Referring to a project in Las Vegas. See Figure 4.1)

\textsuperscript{15}Mayne begins to draw diagram of parking. See Figure 4.1

\textsuperscript{16}Gestures to diagram of hybridized form. See Figure 4.1
solved with some computers because a whole bunch of them could do a form now. And it seems like your interest is not a single form but, multiple forms and now the interactions between those things.\textsuperscript{17} But ... in fact it could look like that, these blocks.\textsuperscript{18} And then, these two other guys, if something else happens, immediately you could move off intuitively ... You would have some idea of where you want to go, and you have some focus you could get to very quickly that’s going, “this is what I’m trying to do in terms of these relationships.”

... GY: That’s what differentiated IFCCA. It really did kind of drive that kind of initial intuition of the components, having to be integrated. Because if you do start with a desert site, its a totally different spirit.

\textsuperscript{17}Referring to Marques’ thesis research.
\textsuperscript{18}Referring to parking diagram.
A.3 Transcript, Evaluation by Experts in Parametric Modeling

Dates: March 27 - 29, 2007
Location: Simon Fraser University, Surrey, British Columbia, Canada
Present: Victor Chen (VC), Maryam Maleki (MM), Davis Marques (DM), Cheryl Qian (CQ), Roham Sheikholeslami (RS).

On Tuesday, March 27, 2007, I gave an informal presentation about my work on Federation Modeler to Victor Chen, Maryam Maleki, Cheryl Qian, and Roham Sheikholeslami. Maleki and Sheikholeslami are graduate students at Simon Fraser University, have graduate degrees in architecture, professional work experience in architectural design firms, and extensive experience with Generative Components and other 3D modeling applications. Chen and Qian are both Ph.D students at Simon Fraser University. Chen has an undergraduate degree in engineering, a graduate degree in Science, and extensive experience with both 3D modeling and software development. Qian has an undergraduate degree in architecture, graduate degree in Science, extensive experience with Generative Components, and is currently conducting qualitative research on user experience with parametric modeling applications.

Due to technical problems with the recording equipment, the March 27 discussion was only partially recorded. On March 28, I met with Maleki and Sheikholeslami to continue the discussion started on the previous day. On March 29, I met with Chen and Qian to reflect on the prior group discussion. The following is a partial transcription of these conversations.

Group Discussion on March 27, 2007
[01:36 - 08:34]

VC: So, in the real world, what will those Noodles represent? So, some Noodles are above the other ones, some will be sloped in the middle, so ... I mean, the real scenario, in the real world, what are they going to make them?

DM: Well, I think the answer is that these shapes that you see here are not intended as the actual building. You shouldn’t read that as being a building. You should read it as being suggestive of the way they might mass things together.
VC: Right, but still ... eventually you need to convert them into something.
DM: Yeah, absolutely.
VC: Well. Did they ever consider about that?
DM: Yeah, and I think that kind of accumulates. So, first, they mass these things together to get a sense of what it is they want to accomplish. And then, given these specific configurations, they can start to work on them as actual buildings, and figure out 'How do you actually hold that up?', 'How do you ...' whatever, right? But, the first step is to capture the intent of what they want to get to.

So they would do stuff like this ... they had this collection of 14 different objects, and ... so they would describe what the shape is like. So the shape is like a ... here's a Noodle. So the shape is multi-directional, linear volume, small, asymmetrical ... so it wouldn't bend in the center. And then, there are certain types of programs that you can ... that can occupy that kind of shape. So they said ... incubators, offices, residential, live-work lofts. Which is different than say, this object here ... they called it a Conquistador, which is a kind of skyscraper. So, its shape was a vertical volume, and it was after large profit, ambition, infinite form. It was ... I guess it was ... they are trying to characterize what that shape is about in one thing, and then they are trying to describe how you might occupy it. But they are two separate kinds of things, right? Normally, I think Architects usually put them together. You make a shape that is very specific for some kind of thing. But here they pulled it apart. And that gives them a choice later on, how to use the shape.

RS: Can I ask a question? You are talking about the components and not the site. And you are saying that each component has a rule, and they put the component inside that site, and then they interact with each other.
DM: Right.
RS: But the site needs to have some rules as well, right? Because they can not put just the component ... and you said that by inserting each design component the context will change, in the whole master plan, or the whole plan. So, there are two things. ... I don't know. I'm just ... I don't know the procedure of this design process, so I'm just asking ... Is there any rule for the plan itself, or for the site itself, that they can not put each component in any place on every place in the site? Do you know what I mean?
DM: So, yes. The site was like this ...
VC: Basically ... I mean, this is the same question as Cheryl asked. ...
RS: ... What Cheryl was talking about was the component itself. What I'm talking
about is the site, you know, because the site has some specific rules. Like, the neighborhood of the site, like in this place or this direction you can not put any commercial, residential thing. So its not about the individual, its about the whole thing. So, do they have, first, a master plan, a whole concept? Because, as I see in those alternatives, they are completely different. So that means they have rules for Independent Components, but it doesn’t seem that they have one certain rule for the plan.

DM: Ok, well, that’s where the idea of the Scenario comes in. So, on one side you have this kind of bottom-up strategy where you make parts, you put parts together and they kind of interact with each other, and then you aggregate all those interactions together to make some kind of configuration, right? So that’s one way to get to it, by adding stuff together until it gets to something. The other way is top-down, where you are going ... “We envision some situation where all the high-rises are on over here and all the housing is over here, and there is a park that meshes them together.” And, then you proceed to add things to meet that ... to match that ... kind of thing that you envision.19

RS: Right. So, can you please explain the meaning of interaction here?

DM: Right, ok ... well, I conceive of interaction as being similar to the idea of relationship. So, you know, we put two objects together, because they have certain properties, they have to relate in a certain way. Now, I don’t call it relationship, I call it interaction, and the reason is because I feel like the word relationship suggests that it is a kind of static thing, and once you put them together its fixed, right? But, we know that the design is going to keep changing. The context is changing, and as we are designing our ideas of what we making are changing too. So, my assumption is that these relationships will keep getting adjusted over time, so that ... at the beginning of the project they may be configured in a certain way, but by the end of the project you will have completely flipped it all around. So, what I am suggesting is that the word interaction just means that the relationships are transient while you designing. So, they interact and then they adjust themselves, and there is an implied relationship between the objects, right? But, its not fixed.

19The remark was intended to convey the gamut of design approaches vis-à-vis Scenarios. I should add that I infer the intent of the IFCCA approach as being one of attempting to overcome the problems of master-planning by focusing on localized interactions, in order to maintain the ability to keep adapting to change, rather than on fixed total configurations. Roham’s question, I believe, is motivated by a concern for the identity of the whole rather than a question of whether they employed a particular strategy (ie. master planning) to realize designs.
Discussion with Maryam Maleki and Roham Sheikholeslami on March 28, 2007

[03:14 - 07:27]

DM: One thing that you brought up, that I think was really valuable, was the idea of team work. Is that ... maybe other people could see Scenarios that they are working on.

MM: Yes, and if its done by a bunch of people, how can you see that in the Scenario view? Can you find it out, when you look at the Scenario, that this was done by one person, this was done by the other person, and this was done by both of them together? And, also, if any of them can see other people’s Scenarios when they are working on them?

DM: Right. So, how would somebody use that? What would be the ...

MM: First, as you said yourself yesterday, it would help them know what the other person would require from them.

DM: So you would see what someone is working on right now ...

MM: And, I am just thinking that, usually in architectural firms, it is not done by one person.

DM: ... yes, exactly ...

MM: It is team work. But, they are not sitting together mostly, they all sit, working at their computers. So, it would be much easier for them if they could look at the other person’s Scenario and see what they are doing.

RS: Or even, when explaining to the Project Manager ... “if you look at my last alternative, ...” So, the Project Manager should be able to distinguish, this is a Scenario done by Maryam, and this is one done by me. And, so you can distinguish the Scenarios, and not by separating the files.

... DM: Ok, so its got a big social part of it. And, it sounds like there is a version control part of it.

... 

[12:26 - 17:00]

RS: What is the definition of Scenario in your program?

DM: Its a collection of elements in interaction, responding to some situation. Or ... just
a second ... let's see what I said.

RS: Is it a design?

DM: ... no, it is not necessarily a design. ... You can say that it is a design, but ...

MM: Is it an alternative? I asked yesterday, but I don't understand.

DM: It is a way of saying that this situation is distinct from another situation. And, the things that you make follow from that distinction, whatever it is. So, I am trying to distinguish between your having organized something, and made it distinct from something else, and then the effect of making that distinction. Like, if you say its a variation — that's an effect of making the distinction, its not the distinction itself. Does that make sense?

... 

DM: I guess I am saying that there are two things happening. The first is that you say that something is different than something else. And, the second step, is that you say what is the nature of that difference.

MM: So, these Scenarios are differences in general?

DM: Yes, somehow there is some difference that leads you to having made them.

MM: But, they can be just small changes between them. And it can be completely different alternatives.

DM: Yes. Ok, here's ... let's say we make one scenario ... let's say we made a whole bunch of scenarios. And we're going to make a new scenario. Its empty. And when we make it, it doesn't contain anything but then we say we're going to borrow part of this one, and then we're going to work on it. So, maybe you call it a variation ... you've made something different than this one. But, then later on you go back and say, ok I'm going to borrow part of this one too, and you put them together and now its not a variation any more, its a hybrid of two things. So, the fact that its a variation or a hybrid has to do with its relation to these other things. But, then those relations are changing all the time. so, that leads me to think that I can't think about that object structurally, like, I can't say that you can make a variation type of Scenario or that you can't make a hybrid type of Scenario. You just make a Scenario. and a Scenario is just some kind of difference, whatever it is, or some way to organize differences. And, its relationship to other Scenarios then tells you that its a variation, or that its a hybridization or something else.
MM: I wanted to know, what your program does? Is it going to be like a 3D modeling program, plus a 2D modeling program, plus Scenarios?

RS: ... Project management? ...

MM: Or, its just that the Scenario part is important and that all the 2D, 3D parts are important? I don't know how it works.

DM: I think its intended primarily as a 3D modeling application. It is not intended to make symbolic representations of things. So, you probably wouldn't make construction drawings with this kind of thing. I suppose you could, but I don't think it would make any sense. You're trying to make the actual 3D objects and have them interact with each other. The goal of the application is to help you manage the interactions of all these objects that lead to some kind of design at the end.

MM: And then it goes to the 2D level ... drawings, plans. The design is from the plans, details. So, you have to go to another program?

DM: Yeah, I think you would export it to AutoCAD or something like that.

MM: So, these Scenario things don't work at that level. ...

My thesis was a hospital. And these interactions were all very important from the inside ... the rooms, the hallways ... they were all interacting from the inside, all the rooms and behaviors among them. It wasn't just 3D modeling. It was very important, the 2D part. I wouldn't be able to have the scenarios and all these things in the planning.

DM: Ok, you're just meaning the plan organization was really important?

MM: Yes. But, I'm not just thinking about 3D. I thinking about the inside, which you usually design in 2D. If I want to take it to another program to draw the design, the 2D part, I wouldn't be able to use the Scenario. Because the ... For example, you had the Noodle. And you said it can be commercial, or it can be some offices. But then you go inside, and you want to decide what it is, you start planning ...

DM: But, are you thinking you need a more diagrammatic view, and that helps you organize things?

MM: You are saying that it is a 3D modeling program. So, when I go to a 2D modeling part, I am not here any more. So, all these Scenarios and charts and whatever are just up to the 2D environment, right? Are you saying that we don't need 2D design, and at that point we can just import images, or?

DM: Well, I guess I am thinking ... I guess there are a bunch of things in my mind. I am sort of the feeling that 2D drawings are sort of, have reached their limit in terms of
their descriptive power and, that the kinds of objects we make these days mean that we can’t really use them for certain types of projects. For example, in San Francisco, Morphosis was building the new federal office tower, and so they designed the curtain wall ... its this big steel system, this big steel shading system ... and, all the fabrication of all the steel in the building was made from 3D models. They didn’t use any two dimensional drawings to do that. And, the reason was that because it was far more efficient for them to do all the fabrication from the model and to make sure that it was accurate. Also, the complexity of the project meant that the 2D drawings that would describe it would not capture all the facets of the model. It is very difficult to understand all the shapes in a two dimensional projection. But, they ended up making 2D models of the systems anyways, not for fabrication but for the city, because the city just won’t accept 3D models. And, that was also the story for a bunch of other parts of the building. And, these days, the way Morphosis is working is that they model everything in the building, and from that they cut sections through the model, and use the sections to make 2D drawings. ... Ok, so that’s one thing ...

A parallel story is that a friend of mine works for a concrete company. And, he does all the form work drawings for the company. So, basically what he does ... the Architects produce drawings of the building, and then he produces his own shop drawings, to produce form work for the building. So, basically everything that the Architect does goes in the garbage, except for the major outlines of the concrete. He then he takes the drawings, and he redoes the whole drawing package just for concrete, and then he makes formwork drawings based on those drawings. So, what I infer from all this is that ... Architects are generalists and don’t know all the intricate details of producing all these systems ... the steel, the concrete, the glass, everything that makes these buildings. They just describe intent, and then its up to the manufacturers to translate the intent into specific drawings or representations that they can then fabricate from.

So, in my mind, 2D is important of course because its how we communicate for the most part. But, its important after the fact of designing. That, most of the value that we will produce ... I mean, I’m just speculating ... in the future, that most of the value will be in 3D models as opposed to 2D drawings. So, then I guess the way I see it in the application, is that the purpose of the application is to model all these things, and to actually model every object. And, then from that, we can generate 2D representations.

But, I understand that it helps a lot to have a simplified view. Sometimes its easier to
see a plan as a diagram as opposed to ... but, that doesn’t really answer your question.

[58:00 - 1:01:40]

RS: The scripting part is a completely new thing for Designers.

MM: And, I think its not that difficult to learn. You know, I don’t know programming. I started working in GC\textsuperscript{20}, and now I can write functions, and I can work with GCScript.

... 

DM: Do you think that if you kept using it, you would want to do more scripting?

MM: Using GC?

DM: ... Well, does the idea of scripting seem useful?

MM: Yes, very helpful.

RS: Well, she does right now. Sometimes she doesn’t even use the Feature. I say, “the Feature is here but still you write your own script.”

MM: Yes, I think its very ... especially for me, its very interesting. I used to like programming ... Pascal, and GWBasic, ... in high school. And, I find it very interesting, to challenge myself to write a script, and see if it works.

RS: But, during the workshop ... your workshop and in the UBC workshop ... people who know programming don’t go through the interface and try to find the Feature. If they want to create something that they know how to program or script, they don’t even look at the Features. And, Robert keeps say, “Ok, I have this Feature, I have this Feature,” but people are trying to ...

DM: See, this is what I have been thinking is that ... maybe it would be more useful to have something like Processing\textsuperscript{21} where you don’t have the fancy graph interface, and all that other kind of stuff. You just write code that says what you want, and it can give you a graph-based representation if you want.

Just like we were saying yesterday, there is this thing called the Deutsch Limit,\textsuperscript{22} which is that once you get 50 nodes on the screen, you can’t really understand what is going on any more. Its ... the text is a much more compact, and easier to understand representation.

\textsuperscript{20}Generative Components

\textsuperscript{21}Processing is an interactive programming language for teaching computer graphics to non-programmers.

\textsuperscript{22}The Deutsch Limit [42] is an informal ‘rule of thumb’ that says once you have more than 50 visual primitives on the screen, the representation becomes too difficult to comprehend.
And, its also that procedural way, like you understand that you went through these steps to get to something.

MM: But, that might be frightening for Architects.

DM: Yeah, it might.

MM: If you tell me, ok I'm an Architect ... look at this Processing ... go make your model. I would never try that because I am afraid. But, in GC you start with the interface. And you start making points and lines, and you see things. And you realize that if I could change this code, write this small function, then I can do other stuff. That's what grabs your attention, and attracts you ...

DM: But, at some point, it tops out and you can't describe the complex things you want to describe.

MM: Yes.

Discussion with Victor Chen and Cheryl Qian on March 29, 2007
[9:05 - 11:46]

DM: I was going to say ... I was thinking, maybe instead of a graph based approach, it should be more like Processing.

CQ: What do you mean by Processing?

DM: Oh, like Processing the 2D graphics application. Which is a sort of very simple application for making 2D computer graphics. Which ... what I mean ... rather than focusing on an interface where you build this graph, and you connect them, and then you add functions as inputs to the various objects. Instead of doing that whole thing, you just go straight to code and you just script it directly in code to get what you want. The reason being that you can ... the text is a more compact description. It already has a natural procedural basis.

VC: Probably that is easier ... I think the user base for your application should be some high level, abstract thinking people. ... I was thinking that way.

CQ: Because, in Atlas Ti ... that is a quantitative research tool ... and, in the tool, you basically open every document to attach code, and analyze the code. There is one function that opens a small visualization window where you can import the notes, and analyze their relation. Maybe just from that approach, that would be enough to analyze the relationship between Scenarios. But, you basically want your application to be functional in design
instead of only analyzing the functional relations. So, what is the priority of use? What is ...
what is the primary use? What is the secondary use?

VC: Yes, basically your interface could be just a visualization part, to visualize just your code.

DM: Yeah ... just sort of like, you know, when you are writing a software application and you’re going to make the interface, they often give you tools to do that graphically.
You drag a button, you click it here, and then you stretch it, and then it figures it all out ...
and then you see the thing. But, in the background it generated the code for you, and I think that’s actually what needs to happen.

VC: Yes, probably ... yeah.

DM: ... Because you can then take that as a starting point, so that you can understand what you are getting, and then you go in and tweak it to make it do exactly what you want it to ...

CQ: Yes

DM: That seems to be better.

[15:10 - 18:56]

CQ: The other thing I mentioned last time is about the comparison between solids and voids.

DM: Yes, that is a really good remark.

CQ: Yes, because you have a bunch of Independent Components ... Snakes, you know ...
who knows ... different parts ... they organize together. But, in our very first year of undergraduate architecture education, we have the idea of solids and voids. So, also base on Gestalt of human visualization, there is also a pattern called the figure and ground. Also, for people looking at images, see the solid and void part. I think for architectural design, it is also very true. So ... I’m just curious, is that opposite part of the design, which is also worth analyzing, and maybe informative?

DM: Yeah, I think it would be really good to have some kind of void object that you would put in the design. And, the purpose of that would be to say, “Yeah ... all you components, stay out of my space,” right?

CQ: Yes. And one part is that the solids are buildings. We want buildings to be connected and communicate. But for the voids part, ... maybe the gardens, the green areas
... we also want them to be linked, and communicate. And that flows...

DM: Yes, voids are very important in architecture.

...

CQ: In Asian and Chinese architecture, especially in the Forbidden City design, the walls are used, especially the windows and doors, as frames of the view. For example, in the Forbidden City, when you stand in front of a gate, definitely, it is a nice frame of the front view ... and, in the South part of China, garden designs ... they are usually some garden walls, like snakes, again inside the garden. And, there are small windows — actually, just openings without glass, just openings in the wall. When you look through each window, you are looking at a picture, there, framed with some trees and some things. You are not just looking at some water or a piece of a wall.

I think we don't have such a tool in which you can generate a model, and it can generate the void part to show you whether or not it's correct, or it flows. So, if there is such a tool, some Designers would really appreciate it.
A.4 Federation Modeler, Application UML Diagrams

Figure A.1: Model interface hierarchy. All Federation model objects implement the INamed interface. This enables them to be identified and addressed by their name. Objects implementing IContext contain other addressable subobjects. Objects implementing IDisplayable provide 2D and 3D graphic representations. Objects implementing IGraphable may have dependencies on other model objects. Objects implementing IUpdateable have states that are transient or may change over time.
Figure A.2: Model class hierarchy.
Figure A.3: ProcessingElement and Behavior class hierarchy. ProcessingElements may be extended to create more specialized objects in the model. ProcessingElements and Behaviors are subparts of container-like objects that implement the IContext interface.
Figure A.4: Systolic Array update sequence. The Systolic Array gathers the dependancies for each ProcessingElement, then performs a topological sort on the graph to order the update sequence from independant elements to dependant elements. The update method on each element that implements IUpdateable is then called in order to update the state of the Systolic Array.
Figure A.5: New Behavior construction and initialization sequence. Behaviors are ProcessingElements instantiated as subparts of an IContext. When given a selection rule, the Behavior observes a target context for changes.
Figure A.6: Behavior update sequence. When the Behavior receives an update notification from the target context, it searches the context for objects of interest and then notifies the model object to which it is bound to update.
A.5 Federation Modeler, Graphical User Interface Studies

Figure A.7: Dependency Graph Viewer, ‘container’ icon study. a) The icon depicts the current state of the object and its subparts. b) If a Behavior is associated with the object, a gray bar appendage appears at the top of the icon. c) In this study, we envision that double-clicking the Behavior bar opens it to reveal the list of Behaviors bound to that element.

Figure A.8: Dependency Graph Viewer, ‘component’ icon study. The component has no accessible or representable subparts. Consequently, the icon provides only a symbolic representation of the Component.
Figure A.9: Scenario Graph, conceptual study, June 2005. The Scenario Graph represents the iterative and incremental development of designs over time.
Figure A.10: Modeler, conceptual study, June 2005. The Modeler provides an interface for creating and editing geometric objects. Prior model states that inform the current state are depicted schematically at the bottom of the modeling view.
Figure A.11: Scenario annotation, discussion and markup, conceptual study, June 2005. The interface allows Designers to collect references to external data that describes the context, record thoughts about that context, and annotate the model.
Figure A.12: Scenario Graph Viewer, interface study.
Figure A.13: Model Viewer, interface study.
Figure A.14: Model Viewer with Scenario Graph overlay, interface study.
Figure A.15: Model Viewer with single model object, interface study.
Figure A.16: Graph Viewer showing graph for a single model object, interface study.
Figure A.17: Code Editor, interface study.
Figure A.18: Stack Viewer, conceptual study depicting a flat listing layout. The Stack Viewer depicts the inheritance chain for the current Scenario. The listing is ordered from farthest to nearest predecessor. a) The current Scenario is highlighted at the bottom of the stack. Predecessors at the same depth in the inheritance tree (ex. Scenario3, Scenario4) are grouped in the listing. b) Alternative means of differentiating the current Scenario from the predecessors were explored.

Figure A.19: Stack Viewer, conceptual study depicting a pseudo-tree layout. The pseudo-tree layout is intended to aid the Designer in understanding the structure of inheritance relations. However, the representation is uneconomical in terms of the screen space it requires and can not scale to represent large, complex inheritance chains.
Figure A.20: Stack Viewer, current implementation.

Figure A.21: Abstract Tree Explorer, current implementation. The Abstract Tree Explorer provides a hierarchical view of the model.
Figure A.22: Model Viewer, current implementation. The Model Viewer is the principal application interface. It enables the Designer to see and manipulate geometric objects in the model.
Figure A.23: PropertySheet viewer, as implemented. The Property Sheet enables the Designer to inspect and edit model object properties.
Appendix B

Illustrating Lists

B.1 List of Programs

Program B.1: IContext.java

```java
package ca.sfu.dmarques.model;

import java.util.List;
import java.util.Map;

public interface IContext extends Named {

    /**
     * Add NamedObject to the Context.
     * @param Named NamedObject to add to the Context.
     * @throws IllegalArgument Exception An object identified by the same name already exists in the Context.
     */
    public void add(Named Named) throws IllegalArgument Exception;

    /**
     * Get Element collection.
     * @return Collection of NamedObjects in this context.
     */
    public Map getElements();

    /**
     * Get List of Parent Contexts, inclusive of the current element. The list
```

157
APPENDIX B. ILLUSTRATING LISTS

* is ordered from root context to the current element. An instance of
  ParametricModel will always be the first element.
* @return List of Parent contexts.
  */
public List getParents();
/**
  * Determine if a NamedObject exists in the local collection.
  * @return True if NamedObject is in the collection, false otherwise.
  */
public boolean hasObject(String Name);
/**
  * Looks up an object or object property value of the form object.property
  * @param Query A NamedObject reference of the form objectname.propertyname
  * @throws IllegalArgument Exception The object reference could not be resolved.
  */
public Object lookup(String Query) throws IllegalArgumentException;
/**
  * Remove a NamedObject from the Context.
  * @param Named NamedObject to be removed from the Context.
  */
public void remove(INamed Named);
} // end interface

Program B.2: IGraphable.java

1 /*
   * IGraphable.java
   * Copyright (c) 2007, Davis M. Marques <dmarques@sfu.ca>
   */
6 package ca.sfu.dmarques.federation.model;

  import java.util.Map;

  /**
   * A model object that may have dependencies on other models objects, that can
   * be represented in a graph.
   *
   * @author Davis Marques
   * @version 0.0.1
   */
16 public interface IGraphable extends INamed {

  /**
   * Get object dependencies.
   * @return Objects upon which this object is dependant.
   */
20  public Map getDependancies();
}
APPENDIX B. ILLUSTRATING LISTS

Program B.3: INamed.java

```java
package ca.sfu.dmarques.federation.model;

import java.awt.Image;

/**
 * INamed.java
 * Copyright (C) 2007, Davis M. Marques <dmarques@sfu.ca>
 */

public interface INamed {

    /**
     * Delete the object from the model.
     */
    public void delete();

    /**
     * Get Context.
     * @return Context of this object.
     */
    public IContext getContext();

    /**
     * Get fully qualified name.
     * @return Canonical name.
     */
    public String getCanonicalName();

    /**
     * Get fully qualified display list name.
     * @return Display list name.
     */
    public String getDisplayListName();

    /**
     * Get an icon representation of the object.
     * @return Image.
     */
    public abstract Image getIcon();

    /**
     * Retrieves name value.
     * @return Name of this object.
     */
    public String getName();

    /**
     * Set Context.
     * @param MyContext Context of this object.
     */
    public void setContext(IContext MyContext);
```
/**
 * Sets name value.
 * @param name Name of this object.
 */
public void setName(String name);

} // end interface

Program B.4: IUpdateable.java

/*
 * IUpdateable.java
 * Copyright (C) 2007, Davis M. Marques <dmarques@sfu.ca>
 */
package ca.sfu.dmarques.federation.model;
import ca.sfu.dmarques.federation.model.*;
import java.lang.reflect.Method;
import java.util.List;

/**
 * A model object whose state may be dependent upon the state of other model
 * objects, and may consequently require updating when changes occur in the
 * model or the application environment itself.
 */

/**
 * Objects that implement IUpdateable may retain properties that require
 * Properties of type Method can not be serialized, and must be restored when
 * the object is retrieved from persistent storage.
 */

@Author Davis Marques
@Version 0.1.0

/**
 * Get the current update method.
 * @return Update method name.
 */
public String getUpdateMethodName();

/**
 * Get the list of available update methods.
 * @return List of update methods.
 */
public List getUpdateMethods();

/**
 * Restore transient and non-serializable values after deserialization.
 */
public void restore();

/**
 * Update the state of this object.
 * @return True if the object has updated successfully, false otherwise.
 */
public boolean update();
Program B.5: IViewable.java

```java
package ca.sfu.dmarques.federation.model;

import java.awt.Image;
import java.media.j3d.Node;

/**
 * A model object that has 2D, 3D graphic or symbolic representations.
 *
 * @author Davis Marques
 * @version 0.1
 */
public interface IViewable extends INamed {

    /**
     * Get an icon representation of the object.
     * @return Image.
     */
    public abstract Image getIcon();

    /**
     * Get display geometry key.
     * @return Key to HOOPS geometric representation.
     */
    public abstract int getDisplayKey();

    /**
     * Get a thumbnail representation of the object.
     * TODO: consider returning a thumbnail set at different resolutions rather than a single thumbnail
     * @return Thumbnail image representation of the object.
     */
    public abstract Image getThumbnail();

    /**
     * Get visibility state for the object.
     * @return True if visible, false otherwise.
     */
    public abstract boolean getVisible();

    /**
     * Set the visibility.
     * @param state True to set visible, false to turn off visibility.
     */
    public abstract void setVisible(boolean state);
}
```
Bibliography


[40] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. 1937.


Index

3D Model Viewer, 92
3DStudio, 40

Abstract Tree Explorer, 93
Ackoff, Russell L., 22
Akin, Ömer, 31
Alexander, Christopher, 73
Ambient, 54, 90
Annotation, Dependency Graph, 91
Appendix, 110
Application, see Federation Modeler
Architectural Desktop, 40
Artificial Intelligence, 29
  General Problem Solver, 29
  Logic Theorist, 29
Assembly, 57, 85
AutoCAD, 40

Backup, 46
Behavior, 54, 57, 86, 90
  Host object, 54
Behaviorism, 26
Binary Decision Tree, 87
Biology
  Mechanistic Conception, 15
  Organismic Paradigm, 15
  Vitalistic Conception, 15
Blog, 91
Burrow, Andrew L., 43
Bush, Vannevar, 28

CAD, see Computer Aided Design
Canonical Name, 78
Chess, 27, 28, 31, 32
Chu, Karl, xx
Chunk, 27, 37, 38

CIAM, 116
Classification, 51
Codification, 45
Cognitivism, 25
Competition for the Design of Cities, 3
Component, 57
Composition, 61
Computer Aided Design, 25, 40
  Research, 31
  Sketchpad, 29
Conceptual Inference Structure, 38
Conceptual Inference Structures, 37
ConfigManager, 81, 84
Context, 55, 57, 66, 77, 78, 82, 85
Cybernetics, 16

dе Geus, Arie, 22
dе Groot, Adriaan, 27
Decision Theory, 16
Dependency, 84
Dependency Graph, 79, see Dependency Graph Viewer
Dependency Graph Viewer, 88
Description Logics, 52
Design
  Creative Design, 31
  Routine Design, 31
Design Office, 3
Design Pattern, 73
  Composite Pattern, 85, 86
  Dynamic Proxy, 74, 87
  Model View Controller, 76
Observer, 73, 77, 80
Proxy, 74
Singleton, 77
Design Space, xx, 43
   Exploration, 43
   Path, 43
Designer, 56
Developer, 56
Development, 61
Directed Graph, 88
Drucker, Peter, 1

Event Id, 81
Expression, 66, 83
   Arithmetic Statement, 84
   Collection, 83
   Function, 84
   Literals, 83
   Parse Tree, 84
   Value Reference, 83
Expression Solver, 83

Factor Analysis, 16
Federation, 76
Federation Modeler, 76
   Class
      AbstractTreeExplorer, 77
      ConfigManager, 77
      IContextGraphViewer, 77
      ModelViewer3D, 77
      ParametricModel, 77
      PropertySheet, 77
      StackViewer, 77
   Classes
      ConfigManager, 77
      ParametricModeler, 77
Event
   Event codes, 77
Interface
   IContext, 77
   INamed, 77
   Observer, 77
Model Classes
   Assembly, 85
   Behavior, 86
   Component, 86

Expression, 83
   Expression Solver, 83
   Input, 82
   Input Table, 82
   Parametric Model, 84
   Processing Element, 81
   Scenario, 87
   Systolic Array, 80

Roles, see User

Federation Modeler
   Interface, 78

Game Theory, 16
General System Theory, 14
   Analogy, 20
   Goal Directedness, 18, 21
      Analytic-Causal Explanation, 21
      Efficient Cause, 21
      Final Cause, 21
      Telelogical Explanation, 21
   Historical Origins, 15
   Identity, 20
   Invariance, 20
   Organization, 20
   Organized Complexity, 17
   Reduction to Dynamics, 18
   Unorganized Complexity, 17

Generative Components, 42, 65, 104, 122

Graphical User Interface, 61

Human Problem Solving, 29
IContext, 78, 88
Identity, 13
IDisplayable, 88, 92
IGraphable, 79, 88
Ill-defined Problems, 32
Implicature, 46
INamed, 78
Independent Component, 5, 7, 12, 13, 102, 117, 122

Information Processing System, 30-32, 34
Information Theory, 16
INDEX

Input, 82
Input Table, 66
InputTable, 82
International Foundation for the Canadian Centre for Architecture, 3
IUUpdateable, 79
IViewable, 79
Java Bean, see Serializable
Köhler, Wolfgang, 15
List of Programs, 157
Local Name, 78
Lotka, Alfred, 15
Map, 79
Markup, 91
Maya, 7, 40
Mayne, Thom, 3, 96
McCarthy, John, 29
Memory
Chunk, 27
Template, 28
Metadata, 52
Method, type, 81
Microstation, 40
Miller, George, 26
Morphosis Architects, 3
Named Object, 78
Namespace, see Context
Newell, Allan, 29
Ontology, 52
Organism
Ambient, 54
Organizational Management
Social-Systemic Approach, 14, 22
Parameter, 63
Parametric CAD
Constraint Based, 40
Propagation Based, 40
ParametricModel, 57, 84
Parse Tree, 84, 86
Part, 57
Peer, 78
Probability Theory, 17, 18
Problem Decomposition, 32
Problem Description Graph, 34
Problem Space, 32, 34
Problems of Simplicity, 17
Processing, 105
Processing Element, 57, 66
ProcessingElement, 80
Update Annotation, 81
Update Method, 81
Production, 37
Project, 56, 57
PropertySheet, 94
Psychology, 26
Rapaport, Anatol, 15, 19
Recall, 46
Reflection, 87, 94
Relation
Loose coupling, 54
Replay, 46
Representation, 37
Intentionality, 43
Partiality, 43
Strong, 45
Weak, 45
Revit, 40
Rewriting Rule, 39
Satisficing, 32
Scenario, 5–7, 12, 56, 57, 87, 92
Contextual Element, 57, 60
Elements
Contextual, 55
Transactional, 55
Environment, 55
Masking, 60
Patterns of change, 61
Transactional Element, 57, 60
Scientific Method
INDEX

Reduction to Components, 19
Scripting, 104
Search
   Breadth-First, 35
   Depth-First, 35
   Global, 35
   Local, 35
      Generate-and-Test, 35
      Heuristic Search, 35
      Hill Climbing, 35
      Induction, 35
      Means-Ends Analysis, 35
Selection Rule, 87
Selection Condition, 87
Selection Filter, 87
Serializable, 77, 80, 94
Shannon, Claude, 27
Shared Mental Models, 104
Simon, Herbert A., 1, 28, 29
Situational Awareness, 104
Sketchpad, 29, 40
SolidWorks, 43
Solution Decomposition, 32
Span of Absolute Judgment, 27
SQL, see Structured Query Language
Stack Viewer, 92
State, 32
State Transformations, 32
Statistical Mechanics, 17, 18
Structured Query Language, 71
Sutherland, Ivan, 29, 40
System
   Behavior, 53
   Closed, 15, 23
   Environment, 23
   Open, 15, 23
Properties
   Contextual Elements, 23, 87
   Defining Properties & Functions, 22
   Essential Parts, 23
   Interdependencies, 23
   Nonessential Parts, 23
Transaction Elements, 23, 87
Systolic Array, 66
   Architecture, 63
   Data Item, 63
   Link, 63
   Processing Element, 63
   Dependent, 63
   Independent, 63
Team Ten, 116
Theory of Information Processing, 29, 30
   Information Processing System, 30
Topology, 16
Turing Machine, 28
Turing, Alan, 28
Type, 51
Update Method, 57, 65, 79–81
User
   Designer, 56
   Developer, 56
Variation, 61
Visual Widget, 88
von Bertalanffy, Ludwig, 15
von Uexküll, Jakob, 54
Watson, John B., 26
Weaver, Warren, 17
Wiki, 91
Woodbury, Robert F., 43
Yu, George, xxii, 3, 96