Data Structures for Context Responsive Modeling in Architecture

By

Si Li

B.S. Architecture (2005)
State University of New York at Buffalo

SUBMITTED TO THE DEPARTMENT OF ARCHITECTURE IN PARTIAL FULFILLMENT OF THE REQUIRED DEGREE OF
MASTER OF SCIENCE IN ARCHITECTURE STUDIES
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

© 2008 Si Li. All rights reserved

The author hereby grants to MIT permission to reproduce and to distribute publicly
paper and electronic copies of this thesis document in whole or in part in any medium
now known or hereafter created.

Signature of Author

Certified by

Accepted by

William J. Mitchell,
Thesis Supervisor, Professor of
Architecture and Media Arts and Sciences

Julian Beinart, Professor
of Architecture, Chair of the Department
Committee on Graduate Students.
Data Structures for Context Responsive Modeling in Architecture

Reader:

Terry Knight
Professor of Architecture
Data Structures for Context Responsive Modeling in Architecture

by

Si Li

Submitted to the department of architecture
on May 22 2008 in partial fulfillment of the required degree of
Master of Science in Architecture Studies At the
Massachusetts Institute of Technology

Abstract

Context plays important roles in architecture design, any feelings we try to provoke, any functions we try to implement, or any existing conditions that we try to address in a design process becomes the context in which our design intents emerge. Understanding the relationships between elements in a given design context and how they affect the decision making for new ideas is crucial in understand how a design comes into form. By capturing this context responsiveness, we can capture the design knowledge into computers for different kinds of automated processes.

Parametric modeling tools and scripting interfaces in modern modeling software have given designers powerful ways to model data. The objects we create from these tools could potentially carry many useful information that can be used in various computation processes and allow designers to interact with geometries with unlimited possibilities. With slight reconfigurations, they could help designers to capture design knowledge into computers.

My thesis will try to use computation concepts such as Object Oriented Modeling and Polymorphism as the basis of technology to allow modeling software to model parametric objects in a way that provides flexible interfaces amongst parametric object, script, and designers. Then I will propose a context responsive data model with its modeling techniques to address the questions of how to capture design in a more effective way.

Thesis Supervisor: William J. Mitchell
Title: Professor of Architecture and Media Arts and Sciences
# Table of Contents

1 INTRODUCTION ............................................................................. 5  
   1.1 Overview of Architecture Design ........................................... 5  
   1.2 Overview Of Current Technologies ........................................ 8  
   1.3 Problem Statement ............................................................... 10  

2 BACKGROUND ........................................................................... 11  
   2.1 Context Awareness and Responsiveness ............................... 11  
   2.2 Object Oriented Modeling and Polymorphism ....................... 12  
   2.3 Related Works ...................................................................... 13  

3 MULTIPLE REPRESENTATIONS OF DATA ............................. 17  
   3.1 Purpose of Multiple Representation of Data In Architecture  
       Modeling .............................................................................. 17  
   3.2 Implementation of Multiple Representations In Data ........... 19  

4 A CONTEXT Responsive DATA MODEL ................................. 21  
   4.1 Proposing the Context Responsive Data Model (CRD) ....... 21  
   4.2 Context Modeling With CRD ................................................... 23  

5 MODELING WITH DYNAMIC BEHAVIOR LISTS ..................... 28  

6 A DESIGN PROJECT USING CRD ........................................... 31  

7 CONCLUSION ............................................................................ 37  

BIBLIOGRAPHY ........................................................................... 39
1 INTRODUCTION

1.1 Overview of Architecture design

Expressions of Design Intentions in Design Contexts

Context plays important roles in architecture design, any feelings we try to provoke, any functions we try to implement, or any existing conditions that we try to address in a design process becomes the context in which our design intents emerge.

When a building is designed in context with its surrounding, there is a sense of harmony between the building and the site. The designer weaves the design solution according to the existing fabric of site conditions, problems and opportunities. This weaving activity involves three actors in the 'consequence triangle'. These are the users, context and the building (White, 1983)

Understanding the relationships between elements in a given design context and how they affect the decision making of new ideas is crucial in understanding how a design comes into form. The importance of relationships between contextual elements and design ideas is prominently reflected in oriental garden designs. For example, in Chinese garden design, the sensational effects of spaces have been the most crucial considerations for designers. From a bright open courtyard to a gloomy twisting hallways, from a rocky hillock accompanied by bamboo bushes to a serene pond along cold stone walls, every space is carefully arranged with regard to its surroundings, in attempts to evoke contrasting feelings and complex visual experiences while one moves through the garden. In “Chinese classical Gardens of Su Zhou” (1993), Liu listed many rules that are used by traditional garden designs. In the chapter “Observation Spots and Routs” he described that every designed space will become a scenic focus from which the surrounding hallways are derived bases on the best viewing angels. Thus the spatial relationships amongst human, spaces, and activities become the primary focuses in the context of spatial planning.

No different from garden design, most decisions exist in context. In any given design scenario, there is no limitation to the types of context we will encounter, and there is no limitation to the types of contextual elements we have to be aware of in order to formulate a design move. For example, the contextual elements we have to consider in a design may be a piece of brick, a human being, or an event. However, regardless of the context we are designing in, the design description “always involves the comprising components of what we are describing and their relationships to other things we know” (White, 1972, p30)

Therefore, by understanding the relationships amongst
contextual elements that influence our decision making and the new decision emerge from the analysis of the relationships, we can understand the design intention for the given context. On the other hand, we can express our design idea in a given context by formulating relationships amongst the conceptual elements and our response to the existing condition.

Expressing Design Intentions Through Relationships Amongst Contextual Elements

Documenting design intentions, or the processes of how an architecture comes into form has played an important role in architecture histories of all cultures. From the text description of the relationship between proportions of classical column and female body perfection, to the ideal formation of Palladian architecture, most of the design intentions are heavily relied on textual documentation. Further more, most of these documentations are served as an educational tool to pass down the knowledge of design. David Slawson described in his book *Secret Teaching In The Art of Japanese Gardens* (1985) that in order to pass down the trade of garden design, many textual codes which specified spatial proportions, geometric configurations for spaces and scenic elements are employed to capture design intentions in Japanese literatures. The following is an excerpt of an explanation for the design intention behind a two-rocks arrangement:

*The stones should be placed firmly, which means the bottom of the stone should be buried deep into the ground. However, even when the bottom is deeply buried the stone might appear weak unless it is associated with the front stone. On the other hand, a shallowly placed stone might give a firm appearance because of the front stone attached to it.* (Slawson)

Constraints in predefined or free form formulations

Capturing design intentions into documents is important not only to pass down the trades, but also crucial in the communication of ideas amongst designers and builders. As digital media played more important role in design nowadays, the question of how much can digital media capture becomes a more prominent question; How to capture design intentions had been a popular research topic in the nineties in mechanical engineering. Perry and Grisham (2006) expressed the importance in capturing design intent along with geometric documents in their paper *Architecture and Design Intent in Component & COTS Based Systems*:

*When we create a product or a component, we also have an idea of how we intend it to be used and express that intent in the documentation as well as in the interface descriptions. Our intent may be specific or general relative to the product or component.*

..., in evolving our systems and components we are dependent on understanding the original intentions that led to the current state that we need to change. Indeed, we spend as much as 80% of our time in discovery or rediscovery in legacy systems. Much of this time is spent trying to determine the original intent of the architecture, design and code. (Perry)

Since the relationships between elements is one of the most important reflections of design intentions. In mechanical engineering, many people such as Chia-Hui Shih who wrote the paper “A Design/Constraint Model to Capture Design Intent”, see constraints amongst geometries as a way to capture design intent:
capture the logical inter-relationship as well as the inter-dependence of design/shape data. The design intent is an accumulation of these interactions of design/shape and constraints. (Shih)

This perspective of seeing parametric constraints as expression of design intents is a widely believed and applied concept. However, the intentions expressed by parametric constraints have many limitations and may be too narrow for architecture. Mostly, design intentions expressed by geometric constraints are limited by the explicit formality suggested by the specific kinds of constraint; in reality many concepts or design intents require non-predetermined forms in order to reach desired outputs. For example, the design intention to find the shape that shelters a given number of people located randomly with minimum surface area. This intention can not be described by geometric constraints alone, it has to consider expanding or reducing the number of components of that makes up the shape which becomes the shelter.

In the past decades, Shape grammar invented by George Stiny has been one of a few successful ways to document design intentions and the forming processes of spaces and geometries in architecture. Basically it generalizes relationships between spaces and design elements into logical rules from which a type of spaces can be generated. Using technology today, rules of shape grammars can easily be captured into digital documentation for automation, education or other purpose. The Palladian Grammar by George Sting and Bill Mitchell, which uses explicit geometric formation rules to generate Palladian Style architecture, is a famous example of Shape Grammar capturing the process of how an architecture comes into form.

The concept of Design Pattern demonstrated in A Pattern Language by Christopher Alexander is another successful concept that greatly influenced design across many different disciplines including architecture, computer science and mechanical engineering. Alexander's pattern articulates the processing of forming design solutions into logical patterns. Each pattern is composed of a description of context, a description of the problem found in the context, and a solution that addresses the problem. Alexander's patterns a reflections of the form follows function ideal. It is not only an effective way for documentation, but also serves more as a problem solving technique which allows designers to break down complex problem into small modules, and solve them with logical solutions.

There are many ways to document design intentions, with technologies today, how can we capture design intentions more effectively and how to make them more useful and more accessible is the research question that drove me to write this thesis. Computer automation has integrated into large parts of our life, and Computer Aided Design(CAD) is no longer just about drafting. Can we use current technologies to better capture design intentions so that computers can use the captured knowledge to generate design or facilitate design processes? Can designers model a style instead of a product by modeling the design intents that will be employed in designs? My thesis will propose a method that answers these question through modeling relationships amongst contextual elements in specific design context.
1.2 Overview Of Current Technologies

When we say the word “Modeling”, architects will usually associate it with the process of visualizing architectural forms in three dimensional spaces virtually or physically. Geometric modeling tools, such as AutoCAD which captures only the geometries of architectural forms, have been widely used in the past decades to illustrate formal characters of a design in virtual environments. The most prominent aspect of this kind of tools is that their data structures are incapable of storing anything other than geometries. All information such as dimensions textual descriptions, or geometric relationships are either not stored or stored separately from the geometries, and there is no logical linkage amongst these data entries. Therefore laborious modifications and data managements become a necessity in adapting this technology. As a result, computer automation is desperately needed to relieve the human labor in modifying and managing this kind data produced from these kind of tools.

Recent years, another type of modeling tools, known as parametric modeling tools, has been adapted in the architectural industry. It offered much more than geometric modeling tools. Constrain Logic Programming, which serves as the basis of parametric modeling tools dated back to the late eighties proposed by Jaffar and Lassez (Colmerauer, 1987). This computation approach models data into constraint networks which are capable of storing logical relationships, such as mathematical relationship between different geometries, between data entities. Idealistically it is much easier to manage than geometric models because once the constraints are built, designers can manage only a small amount of selected human-readable attributes from the data, and let the constraints take care of the rest. For example, modifying the size of a window is no longer required to go into each vertex of the geometry but only change the numeric representation of width and height. However, it takes great engineering efforts to build all the relationships that are required to form a useful constraint network. The effort to build such a network becomes more precious due to the fact that the network is extremely fragile; removing or replacing a node may cause the network to collapse. Further more, once the data is built, modifications can only be done on parameters used by the Constraining Handler Rules, which are used to solve the constraint network. In another words, only the parameters of constrains can be changes but not the constraints them selves. Therefore there is a great demand in designing parametric models as flexible as possible to extend the application and life span of the model. Up to today, parametric modeling tools are generally believed as effective production tools but not design tools.

Figure 1.2.1 showing a parametric model where geometries are defined by parametric constraints

Scripting interface is another popular technology that is adapted in architecture design recently. The purpose of scripts is to makeup functionalities missing in particular software systems or allow users to perform their own automations in particular
software systems. Almost all modeling software today has scripting interfaces. These scripting engines usually allow full functionality of procedural programming. However, they are not equally powerful or flexibly due to the data structure design in different modeling software; what users allowed to do in scripts are largely limited by the Application Programming Interface (API) provided by the software providers. Furthermore, different data structure and API in different software usually have their own design logic, which result in completely different ways of operating objects from script. It usually takes lengthy time to adapt a new data structure and API, and depends on the flexibility of data structures, it usually takes great engineering efforts to write a script that does very little. This aspect of scripting has made many designers reluctant to learn this technology and thus discouraged the discovery of potentials of scripting in designs. Therefore there is also a great demand in making scripts more reusable.
1.3 Problem Statement

The technology is ready for designers to express designs in boundless ways, with minor reconfigurations, the technology we use in daily design practice will give totally different opportunities for designers to think about design. My thesis propose a frame work which utilizes current technologies to let designers to capture design intentions into scripting modules, and using these modules to model behavioral patterns that manipulates exist parametric architecture models for design automation or facilitation. As a result, designers would turn their design focuses to formulate modulated design ideas and the composition of small ideas into large ones.
2 BACKGROUND

2.1 Context Awareness and Responsiveness

The first question in Context Modeling is the definition of Context. Tarek, Narjes and others spent a large chapter in their paper *Towards a Methodology for Context Sensitive Systems Development* (2007) explained that context cannot be defined without a larger context. However, they also stated that:

*Nevertheless, there are recurring key questions i.e. “invariants” characteristics in the definitions such as the context relates always to an entity, is used to solve a problem, depends on the scope of use (domain) and time, and also that context is evolutionary by regarding it as a dynamic process in a dynamic environment and not as a set of states.*

In general terms, context can be seen as a domain that contains all the influential factor that we have to consider in a decision making process. In 1994, Albrecht Schmidt proposed the idea of context awareness for mobile computing devices in the paper *Context-aware computing applications*. It’s an idea that the software can reconfigure its user interfaces or auto-start-proposals depending on the physical environment such as location or noise level.

In computer science, context modeling has been focusing on how to let computers to identify different contexts using retrievable information. As a designer, my thesis will focus on how to model contextual awareness and responsiveness into the data we use in design.
2.2 Object Oriented Modeling and Polymorphism

Object Oriented Modeling (OOM) and the concept of Polymorphism which enabled by OOM are the technological foundation of the framework I propose in this thesis. Therefore to understand the thesis we must first understand OOM and polymorphism, and their role in design.

Object Oriented Modeling is a way of abstracting real life issues for problem solving, which is used mainly in Object Oriented Programming (OOP). The term “Polymorphism” refers to the idea that abstractions we create in OOM can take many forms. An instance of data can have multiple representations depends on the functionality on which is focused. This idea gave revolution to the world of computer science because we no longer have to recognize an instance of data by a set of specific descriptions but by the abstract description of the capabilities that this data posses.

The classic example used by many programming texts that explained polymorphism is that both cats and dogs are polymorphic representations of the type animal because they both share the common functionalities of walking, eating, and shouting; however they walk differently, eat differently, and shouting in different sound textures. In computation, “Interface” is the technical instrument that helps to implement polymorphism.

Interface is the meta data that describes the capabilities of a type of data. For example, the interface “can walk” specifies the type of data that bare this mark must have the capability to “walk”, however, how it “walks” is totally depended on the explicit implementation of the data. Polymorphism has made data much more reusable because we can apply the “walk” instruction to any data that has the “can walk” interface, disregard how much the dogs and cats walk differently. By modeling interface into parametric models that we use in architecture design will allow them become more accessible to scripts. This will be more expensively discussed and explained in the chapter 3.

“Design Patterns” is one of the most influential computation literature published in 1995 by the Gang of Four. This book described 23 commonly seen programming problems and proposed corresponding programming design patterns to solve these problem using OOP techniques. Behavior Patterns is a group of patterns that takes full advantage of polymorphism to provide adaptable behavior to computer programs. One of these behavior patterns in called the “strategy pattern”:

The strategy pattern is useful for situations where it is necessary to dynamically swap the algorithms used in an application. The strategy pattern is intended to provide a means to define a family of algorithms, encapsulate each one as an object, and make them interchangeable. The strategy pattern lets the algorithms vary independently from clients that use them. (wikipedia)

My proposed framework is heavily based on the strategy pattern, it provides a flexible structure to the data and allows components, in this case parametric models and scripts, to be more reusable.
2.3 Related Works

Tea House Design (modeling contextual relationships in shape grammar)

This work tries to generate architecture design through constructing relationships amongst conceptual and concrete design components. The goal is to capture feelings provoked by traditional Chinese residential housing into the design of a tea house complex. The design intents are expressed by formulating geometric relationships amongst different design components into shape grammar rules. These rules become instructions on how to generate design.

The project breaks down the design process into modules based on design contexts. The first design context in this project is to design the space module in which people would have tea. The design intent for this context is to formulate a cozy and intimate tea-drinking space. Therefore, the elements we have to consider in this decision making includes but not limits to: seating areas, seating orientations, passage, view, distance between people, gesture of people, etc. They are the Domain Elements in this design context. By defining relationships amongst these elements, a set of grammars are formulated to generate tea house components.

Figure 2.3.2 showing the rules that are used to generate the tea house module

Some generations of rectangular designs
The list of rules shown in figure 2.3.4 depicted the design intentions in the design context of “visual linkage between spaces”. In this context, the domain elements, or things we have to consider in this context, includes: physical barrier between spaces, view orientation, and circulation passages. The rules define in this context specify how each component should place itself in relation to others. As a result, they guarantee all spaces generated by these rules would have a visual penetration to adjacent spaces.

Other design contexts being addressed in this project includes spatial accessibility, twisting impressions of hallway, pictorial qualities of corners and visual penetrations, etc. In the end, by modeling design components and relationships amongst them, this project captured a style into grammars, and this knowledge stored in the grammars can be employed by machine to generate designs which preserves the spatial qualities intended by the designer.
Houses Generated by Patterns (contextual relationships in design descriptions)

The book "Houses Generated By Patterns" by Alexander Christopher presented sixty-seven design patterns to address issues in residential housing design. Each pattern contains three parts: Context, Problem, and Solutions. Context describes the domain in which problem is general found. For example, "kitchen in Latin Family". Depends on the problem being addressed, context can be very abstract, such as "Any low income house in Peru" or can be very specific such as "The roads meet in T-junction, whose angle is as near 90 degrees as possible". Problem is the set of factors we want to consider in design or conditions that we want to improve through design. Solution is the set of literal descriptions of how to carry out the design.

Figure 2.3.6 is one of the pattern from "Houses Generated By Patterns" called "Light On Two Sides Of Every Room". The context in which the problem is found is any room that has windows on one side of the room. The intention behind this pattern is to improve the comfortability affected by glaring effects of the room. The solution is to have windows or skylight in the room in at least two direction. In the description of the the design solution, it clearly stated the contextual elements designer has to be aware of in order to implement the solution. In this case, the direction in which the windows are faced.

Figure 2.3.6

Figure 2.3.7 showing the components of a pattern according to Alexander Christopher.
Context Modeling (embedding contextual awareness in design tools)

In 2001, Fred A Mburu wrote the thesis *Context Modeling* which presented a computation model to allow designers to define and use context in computer aided design. Fred acknowledged the importance of context in design and the lack of design knowledge associated with context in current design tools. He demonstrated his idea through an example to model windows buildings responding the site orientation and view conditions. He designed an interface in AutoCAD to allow designers to define context such was views and orientation, then allow designers to prioritize them in to associative ranks in order to address conflict situations.

Figure 2.3.7 shows the contextual relationships amongst the contextual element “Icon G” and its associated objects “wall A”, “wall B” and “wall C”.

Figure 2.3.8 shows when there is a conflict in the design context between “NorthScope” and “GoodView”. The associative rank is used to pick the winner.

Figure 2.3.9 shows the automated window installation in a house based on the given context.
3 MULTIPLE REPRESENTATIONS OF DATA

3.1 Purpose of Multiple Representation of Data in Architecture Modeling

An architecture design is a manifestation of different layers of considerations and creativities, and there is no single type of description is sufficed in explaining the physical form results from the collective design solutions in different design contexts.

It is our habit today to see designing as a personal endeavor; we discuss it in terms of meanings and impressions: that is to say we tend to focus on how we feel and what we want when we design. Interesting and important as this may be, it does not help us understand designing itself. (Habraken, p43)

Therefore before we can discuss how to model context responsiveness in architecture using discrete data, we must first understand how to represent the data in a way that provides a more flexible representation to design in different context. Essentially we can understand design in two different types of descriptions, “syntactic” description and “semantic” description. “Syntactic” description is what can be understood from the logical processes in which the architecture is designed while “Semantic” description is the subjective feeling that a designer has towards a design, such as aesthetic preference or emotional inputs. Due to the explicit nature of digital data, the discussion of this thesis will focus on the “syntactic” description of architecture design.

George Stiny depicted multiple representation of syntactic description in architecture as “h: L of G -> D where h:L* of G -> D is defined recursively by m+1 rules” in his paper A note on the description of designs. It provided a very explicit representation to the idea of polymorphic nature of design description in different context.

When we design, our brain gives priority to different design focuses; decisions happen in different layers, or design contexts. In Alexander’s A Pattern Language, the most valuable thing is not the patterns that Alexander generalized, but the way of problem solving of have the problem-solution pair, and the systemic linkage between these pairs. This way of thinking unavoidably put a designer to perceive things as logical systems. The same in describing design solutions from different design contexts, the description should be a system that contains a collection of problem domains where each domain contains a problem and a set of solutions. For example, an admissible part called “studio space” can contain domains of “lighting”, “circulation”, “ventilation”, etc. The domain of “lighting” has a problem of n” lux of light is demanded in location L1” and the solution is “open a hold on a wall in location L2”.

A note on description of design suggested that “the description function h (of a design) is a many – one relation” (Stiny,p265). We can easily image a space as result of L of G can have n different description functions (as a result of perceiving it in different contexts):

h1: L of G -> D of lighting  
h2: L of G -> D of acoustic  
h3: L of G -> D of view  
h4: L of G -> D of circulation  
... ...

hn: l of G -> D of other context

Therefore the data which represents architecture design should
have the capability to be recognized not only by the explicit geometric properties of the form, but by the contextual interpretation from which the form came about. For example, in lighting design context, we understand window as a hole in the wall with its translucent property, but in a thermal design context, we understand window as layer of insulation materials (glass, air, and frame).

Due to the similarity of problems exists under the same design context, the design solutions are usually very similar, and they can be solved by the same operation with variations. For example, in the design context of window placement with regard to heat lost reduction, the design solution is most likely to be either reduction in window size, or increment in glazing layers. The solution can be used in school design or residential housing design, the difference is how much a designer reduces the size according to the building type. Therefore, the data which represent the design should also be able to be recognized by any design context alone, which make it more accessible to manipulation by computers. For example, if “window size reduction” is a computer function, any type of space that contains window should be able to subject to this function what kind of space it is. In computer programming, this issue can be handled with the concept of “Interfacing”

![Diagram showing object modeled with interface can be recognized as different type based on contexts](image)

**Figure 3.1.1** This diagram shows an object modeled with interface can be recognized as different type based on contexts
3.2 Implementation of Multiple Representations In Data

In computer programming, it was briefly described in chapter 2, that “Interface” is a kind of meta-data that describes the available ways the data can be interacted by others. A data modeled with interface allows it be recognized not only by the type of data, but also by the interfaces, and a data can be modeled with multiple interfaces which allows it to be recognized in various types. This is a very important concept in thinking about digital data. Interacting with digital data is essentially to modify some specific attributes or variables of the data, and the computer is not smart enough to know what variables it contains without knowing what type of data its working with. For example, it has to know there is a variable call “A” before it can give any value to “A”. The analogy in real life situation is that we can not enter a building if there is no entrance, on the other hand, we can enter any building disregard what building it is if there is an entrance.

Interface in OOP usually describes a specific function that the data who implements the interface must contain. In none-OOP languages, interface is usually refers to a type of variable pointers or function delegates. Since the framework in this thesis is based on an OOP approach, therefore the none-OOP approach of interface will not be discussed. However, it is important to acknowledge its importance in general software design in order to understand why the discussion of interface implementation must precede the discussion of the data model proposed by this thesis.

The following is an example showing how can interface change the way we interact with data:

Given that we have two types of data: “Wall” and “Window”, the following table shows attributes they contain:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Wall</th>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>width</td>
<td>width</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>thickness</td>
<td>translucency</td>
</tr>
<tr>
<td></td>
<td>price / sqf</td>
<td>price / sqf</td>
</tr>
</tbody>
</table>

Since any computer program requires to know the type of data it is working with before it can manipulate the data, computer functions usually requires to know the type of the data before it can be passed into the function, or used by the function. Therefore different type of data requires to be manipulated by different functions. If we want to change the “width” of the “Wall” and “Window”, despite their similar effort it has to go through, for example ( F (value) | “width”=value), the computer needs two separate functions to perform the task separately. In another word, it means a programmer has to write the function twice with redundant syntax such as
(F-wall(value) | the “width” of “Wall” =value)
(F-window(value) | the “width” of “Window” =value)

By using interface, we can reduce the complication of writing two separate functions. For example we define an interface called “Has Width” and this interface specify that any data who implements the interface must contain a function “get width” such that the function returns the width attribute of the data. If this interface in implemented into both “Wall” and “Window”, than we can use one single function such as
(F(value) | the “get width” of “Has Width” =value)
to manipulate the “width attribute of the data disregard whether it’s a “Wall” or “Window”.

Further more, the design of interface is usually context specific. For example, defining interface of each of the common variables shared by “Wall” and “Window” could be very cumbersome, therefore we can define an interface for a group of attributes instead of one, base on the purpose in which they are used. In the design context of resizing architecture components, we can define an interface “Resizable” which specify that the data who implements this interface must contain two functions: “get width” and “get height”, or a single function “set size” which sets both “width” and “height” at the same time. Since architecture design requires multiple representations, while we try to understand the data in the context of “cost estimation”, we can define an interface “cost by square foot” which specifies the data to contain “get width”, “get height” and “get “price / sqf”. As a result, the computer can use single function on any data that implemented “Cost by square foot” interface to perform calculations such as “Total cost=width x height x price/sqf”. The “Resisible” and “Cost by square foot” interface can coexist in one data, and allows the data to be represented differently in different context.

Interface not only allows a data to be represented differently base on different context, it can also be recognized in multiple forms of representation at the same time. Consider the scenario that we are designing in the context of resizing architecture components. Although the actual manipulation to resize the components can be done through the same function, however, how much we want to resize a component also base on other factors such as “what are we trying to resize?”. In this case the “Wall” is going to take forms of both “Wall” and “Resizable” and have operations such as:

If the data is a “Wall”

Resize it by half as a “Resizable”

If the data is a “Window”
Resize it by a quarter as a “Resizable”

Having the concept of interfacing in mind, when we model architecture components, we must consider how the components will be used in various design contexts, and the types of design context involves in a design process.
4 A CONTEXT RESPONSIVE DATA MODEL

4.1 Proposing the Context Responsive Data Model (CRD)

The Context Responsive Data model is comprised of two main components that assemble existing parametric models and scripts into one single object which I called the Intelligent Object (see figure 4.1.1). These two components are the Context Handler and the Intelligent Object container. The Context Handler is basically a set of scripts stored as a Behavior List which drives the behavior of the parametric object (circled in blue in figure 4.1.1) contained by the Intelligent Object container. On the other hand, the Intelligent Object container provides a set of meta-data that describes the interfaces in which this object uses to interact with others; in general terms, it combines a specific parametric model and a context handler into a single entity that is able to act on its own and be acted on.

The “Context Handler” interacts with the parametric model and other intelligent Objects, and it contains a list of scripts shown in blue rectangle which named “Reasoning and Response” that is responsible for the behavior of the parametric object that it associates with.

Figure 4.1.2 is an iteration of the diagram above shows a slightly different data model. In this model, the parametric object is not encapsulated into the intelligent object. The Intelligent Object only contains a behavior list that can potentially act on any parametric objects in the scene. However, it is important to have the behavior list to be encapsulated into a single object so that it will be capable of restructuring itself and be manipulated by
others. (The concept of Behavior List will be discussed in detail in the next chapter.)

Within the Intelligent Object, the Context Handler undertakes the following tasks:

1. Communicate with the parametric object; It is able to inquire parameters and interfaces from the model and manipulate the model.
2. Communicate with other intelligent objects as context elements, analyze the condition of the context element in the appropriate context using script components.
3. Trigger actions and manipulate the parametric model base on context response modeled into the context handler.

The Context Handler is the most important part of the Intelligent Object, and it is the place where most behavior modeling processes take place. It is also the place where "intelligence" being stored and extracted. The Context Handler can be built to address multiple issues or dynamically restructure itself to address different issues base on the situation. They are the constraints that provided linkages amongst intelligent objects in the scene to create constrain networks which are capable of topological change.

Since the Intelligent Object is a set of meta-data instead of actual data, therefore it can be created as a type of object where instance of this type can be created. As a result, all instances of the same type of Intelligent Object can be managed and manipulated at type-specific levels. In architecture design, window scheduling or door scheduling are examples of monitoring components at type-specific levels. Pouring concrete using the same mode is an example of creating instance of the same type.
4.2 Context Modeling With CRD

To model with CRD, the first thing to be aware of is the appropriate level of abstraction which is being modeled, in terms of scale of operable components and the amount of intelligence associated with it. In theory, the CRD is abstract enough that you can make your own decision to how you want to model. The following examples demonstrate modeling with CRD in different abstraction levels.

**Smoking Effect (Introduction to using CRD)**

One of the most important features of the CRD is to be able to use simple scripts as components to build complex operations, and then encapsulate the operation into a single portable object. This example demonstrates a detailed implementation of a smoking graphic effect through modeling a behavior list using CRD.

The first thing is to make a parametric object that will be capable of performing all the actions required to become a smoke particle. In this case, the capabilities include: change of position, size, and translucency. Therefore, before we model the parametric object, we must define three interfaces that will be implemented by the parametric model. This is the list of interfaces and their associated capabilities:

- “Can Update Position” : get or set the position of the model
- “Can Update Size” : get or set the overall size of the model
- “Fade Out”
- “Move to Point on a surface”
- “Delete After Given Time”
"Can update Translucency": get or set the translucency of the model.

The next thing is to build the model. This model is a simple perfect square, in order to link the parametric model to the interfaces, here are the list of parameters associated to the capabilities described in the interfaces:

1. Parameter “Position” refers to xyz coordinate of the lower left corner of the square. It will be updated and inquired through the “Can update Position” interface.

2. Parameter “Size” refers to the edge length of the model and it will be updated and inquired through the “Can Update Size” interface.

3. Parameter “Translucency” refers to the Alpha value of the fill color by which the model is drawn, and it will be updated and inquired through the “Can update Translucency” interface.

Looking at the structure of this model passively, we can understand it as: if the model contains “Can Update Position” then it means there is a parameter called “Position” which I can manipulate through scripts. On the other hand, an active view to this data structure would be: if it contains “Can Update Position” interface, there is a predefined function that allows to update the position of the model. The difference between the passive view and active view is that a predefined function which updates the position of the model can do more than just updating the model. For example, it can print out the new coordinate while updating the position as oppose to only changing the value of “position” parameter. The active view is much more effective in controlling complex situations which will be discussed in later chapters.

Base on the modeling software on market today, it is important to understand the difference between the passive view and active view. For example, CAIA will be easier to be reconfigured to work with CRD in the passive view. Generative Component will be easier to be reconfigured to work with CRD in the active view because of its ability to allow users to define functions and attach them to objects through modeling user interface.

The next step is to turn the parametric model into a smoke particle by modeling the contextual awareness and responsiveness into the behavior of the parametric model. While the usage of behavior list will be discussed in the next section, I will discuss the four behaviors implemented for the smoke particles:

1. “Generating Objects”: creates a model at a given point
2. “Moving Objects Toward Wind Direction”: move the model toward a direction with randomness
3. “Grow In Time”: Increase overall size of the model as time goes
4. “Fade Out In Time”: Decrease the translucency of the model as time goes

The first behavior is a globally applied script which generates smoke particles at a given point while the other two are locally applied scripts that are directly associated to each smoke particle. A Smoke Particle is basically an Intelligent Object composed by the square parametric model that contains the three interfaces described from above and a behavior list that contains three local scripts they associate with.

The function of each of the scripts applied to form the Intelligent Object is very general, and they can be applied differently in other context to achieve different outcomes. The most influential factor that determines what an object will be in the end is the behavior list. Finding the appropriate scripts and
combining them in appropriate orders and arrangements is the most crucial design process in modeling with CRD, the next section will demonstrate how to model context awareness and responsiveness through the modeling the behavioral list.

**Conceptual Design of A House (Introduction to using CRD in architecture design)**

In terms of modeling context awareness for design, the context can be anything, it can be a piece of geometry, a number, a string of words, or a religious believe. However, there is one common aspect in awarding the different types of context. That is to establish a relationship between the contextual elements and the design solution. These relationships will be expressed in the form of behaviors in my Context Responsive Data model.

The Following example shows how to model design concepts with CRD to address abstract architecture design issues. In order to use CRD in architecture design, we have think about architecture design in terms of relationships amongst design components. For example, when we are doing space planing, we are considering the location of a spaces in relations to its adjacent spaces. There are some commonly accepted possibilities for spatial adjacencies such as the living room is usually next to the dinning room, and the dinning room is usually next to the kitchen, etc. If we want to let the computer to propose a space arrangement for a house, we can assign different generation behaviors to each type of space, and let it grow on its own.
In this example, I used a parametrically defined hexagon (figure 4.2.3) as the symbolic object to represent spaces. Behaviors will be designed to generate and modify instances of this type of hexagon to give a space arrangement proposal which can be later interpreted into actual architectural form as illustrated in figure 4.2.4.

Figure 4.2.5 shows three example types of spaces and the allowable spaces that they can generate. Given a site, the sidewalk will trigger the generation of an entrance path, and the entrance path will trigger the generation of other spaces. In order to avoid undesired conditions such as spatial conflicts, redundant generations, etc, each space must be aware of not its surrounding spaces, but also the overall statistics for all the generated spaces.

Another behavior designed into the hexagon is to give it different size and height based on the functionalities of its space type. For example, the living room might be taller and larger than the dining room. Figure 4.2.6 shows some iterations of space arrangement proposal generated by the parametric hexagon and its behaviors.
5 MODELING WITH DYNAMIC BEHAVIOR LISTS

Behavior List is a simple concept that can be used to formulate complex models. Behavior List in software reflects the design ideology expressed by Behavior Patterns from the Design Patterns (1994). It has been widely applied in software design, and it's a very popular approach to implement behaviors in AI agents to react to specific type of environments and fight specific types of enemies.

According to the Gang Of Four's Design Patterns, it can be seen as an example of a prioritized Strategy Pattern. The prioritized Strategy Pattern selects context response base on the priority in which the responses are built into the Behavior list. The public reaction of this game had proved the effectiveness of building prioritized behavioral list as a quick and easy way to model context-specific AI.

By bringing this idea into design tools, we can create simple user interface to let designers to model context awareness by selecting script components from a behavior library, and build complex automation from general scripts.

Because of the linear nature of lists, behavior lists are very easy to construct even for video games. Figure 5.1.1 shows a screenshot of the behavior list in a video game Final Fantasy XII.

Basically it allows players to construct a behavior list out of libraries of contextual elements and responsive actions in order to quickly create an AI for a character to react to specific environments and conditions.

Figure 5.1.2 shows the behavior list used in the AI
implementation for Halo2, a video game. According to *Handling Complexity in Halo 2 AI* (2005)
It was the state of art in game AI programming in 1995. The AI was designer for computer agents that play against human players in gun fight. The behavior list arranges a list of in-war actions by priority, by changing the priority for the actions in the list will result in completely different tactical behavior. For example, by giving higher priority to shooting over healing will result in a more progressive tactic while giving higher priority to dodging will result in a more defensive tactic. Further more events in the game can trigger reconfigurations of the behavior lists or introduce new actions into the lists. As a result, the AI behaviors will adapt to the changing environment.

Behavior list can be designed in many ways to adapt to changing contexts, and the agent which reconfigures the behavior list to adapt to different design context can also be modeled into the Behavior list itself, as expressed in the Chain Of Responsibility Pattern from *Design Patterns*. The Chain Of Responsibility pattern is an idea that a behavior will trigger new behaviors to replace itself or others base on contextual conditions. The following demonstration program shows how to model a behavioral list using general script components to form a design AI that addresses issues in architecture design.

```
F1
F2(F1)
F3(F2(F1))
```

*In the end, by stacking more scripts, we get:*

```
F_n(F_{n-1}(....(F_3(F_2(F_1))))
```

---

The task is to provide a shelter for a habitable space represented by a human figure. There are two parametric models

[Figure 5.1.3](image)

[Figure 5.1.4](image)
applied in this design. One is a black dots which contains a parametric attribute for its position. The other is a quad surface which is parametrically defined base on four relocatable points.

Similar to traditional architecture design, the first design move is to work out a site plan for the shelter, therefore I created a behavior using the idea of cellular automata to generate a grid of black dots around the person (figure 5.4.1). When this behavior is added to the behavior list, it is reapplied over and over again, thus it established a relationship between the dots and the person. In another words, when the person moves, the site plan follows. Designers can focus on placing habitable spaces anywhere they want, and the behavior will figure our a site plan for the given habitable spaces. The next behavior I call “shelter height” is to create the height for the space, and result in a dome-like structure as shown in figure 5.4.1.

The behavior list can be dynamically be reconfigured when the contextual condition changes. I made a behavior called “aware conditions” that restructures the behavior list to replace the “shelter height” behavior with a new one when two habitable spaces are very close to each other and result in a wing-like structure as shown in figure 5.1.5

Each behavior script used in the behavior list is very simple and general, however, there collective effect is intelligent. It demonstrated the ability of this model to form complex automation through building from simple components.
6 A DESIGN PROJECT USING CRD

This chapter demonstrates a simplified implementation of the proposed Context Responsive Data model in a design project. Due to the small scale of the project, I avoided the necessity in creating libraries of interfaces along with the parametric models. Instead, I create a base object which serves as collection of predefined interfaces that covers all possible scenarios in the design, and it is called the “WetObject”.

![Diagram of WetObject](image)

Designers create new parametric objects by adding new attributes to the original WetObject and display graphics through Processing scripts, a popular graphic scripting platform. Once the parametric object is created, designers can write JAVA scripts as individual behaviors to manipulate the parametric object. Any automations. Context awareness, or other constraints can then be captured into the behaviors and be reflected collectively in the behavior list.

The design project focus on the formal characters of spaces through generative methods. It was Jimmy Shen who first had the idea of “Springy Thingy” which uses physical properties embedded in particle spring systems to generate form. Later on, Jimmy Shen and Kaustuv Biswas developed the project into what is called the “wet space”, in which user can generate surface that avoids gravitation repellents by allowing lines of spring to dissolve into gravity. In the spring of 2008, Mark Goulthorpe put together a workshop to recreate the “Springy Thingy” in a more dynamic and interactive way for an exhibition. This is an opportunity to apply the context responsive data model I proposed because each form generation step in this project is about relationships amongst others. Therefore Mikey Fujihara from both course 6 and 11, and I implemented this data model into an explicit data type and designed a service engine to put it in test.

Taken the idea from its former projects, this design starts with a cluster of random repellent being emitted on the empty screen. These repellents can be seen as activities or events that try to occupy an architecture space. Designers than start drawing lines on the screen. The lines are particles linked by springs that subject to gravitational influences. While the spring moves towards the direction of the gravity, they are repelled by the the repellents and forms dome-like spaces which suggests a design solution in the context of generating habitable voids.
Context Awareness In The Camera

Since this project was intended for a public exhibition, we assume not everyone can draw like an artist to produce nice looking spaces, and it takes time for anyone to learn to draw an architecture space using this system, therefore I decided to reduce the responsibility for the designer by shifting the responsibility to the computer.

Since the designer can only draw on a flat screen, therefore the position of the camera plays a big role in defining the virtual location of the lines drawn by designers. Thus, the immediate solution for this problem was to automate the camera in a way that restricts the location where the designer can draw. By prescribing the camera trajectory, the lines drawn by designers will always form a dome like space. Furthermore, the designs should always be able see the generated spaces.

Giving the camera such intelligence is not an easy task, camera position is heavily depended on all the objects in the scene, thus the camera requires many different types of context awareness. For example, in the context of “try to see all the spaces”, the camera has to find the center of mass for all displayed geometries, than position itself so that the center of mass is always near near the center of camera view.
In the context of “forming an architecture space” the camera has to be aware of the location of repellents so that it guarantees all the lines drawn by the designers can be repelled to form a habitable space. Figure 6.3 shows a design where the camera is fully aware of its surrounding so that it moves itself around the emerging repellents to guarantee lines drawn by designers will be repelled into dome like structures. Furthermore it adjusts its own speed to change the density of surface give in order to hierarchy to the spaces. On the other hand, figure 6.4 shows a design where the camera has no intelligence at all.

Many strategies were explored to see how can the

Figure 6.5
camera movement affect the characteristics of the generated forms. Figure 6.5 shows two different types of spaces as a result of different camera movement.

**Context Awareness In The Cladding Materials**

In order to produce interesting real time animations, a few different kinds of intelligence were designed into the cladding members. Each cladding material is smart enough to acknowledge the design and adjust its position and configuration to fit into the design.

When the spaces are form, the grid which represents the space is recognized as symbolic components that is called “place holders”. They act as the blue print of a design and tells where the actual architecture components should go. One of the intelligence given to each individual cladding material is to read the blue print and places itself in the virtual space base on the blue print. What actually happens is that each cladding material will try to find place holders that are not occupied and occupy them. By assigning different strategic behaviors to cladding materials to occupy the place holders, we can achieve very distinctive animation effects.

The cladding materials can read multiple blue prints. When multiple blue prints exists, the cladding materials have to be aware of which blue print it is reading. Specific events can trigger a change of behavior so that the cladding material will try to read another blue print. As a result, they will collectively move from one form to another (Figure 6.6).
All these different kinds of context awareness are modeled individually as behavior scripts. The same cladding material can be subjected to any kinds of behavior. Furthermore, these behaviors can be subscribed to other any other objects that modeled with the corresponding interfaces.
7 CONCLUSION

In a practical level, the Context Responsive Data model basically allows designers to encapsulate procedures into parametric models. The potential of the system is that it can use custom scripts as constraints to effect any parameter in the parametric model, therefore it will be no limitation to the kinds of constraints to be used the model. It provides a potential to lessen the role of predefined geometries and to increase the role if predefined relationships amongst design components.

The Context Responsive Data model is built on top of the technological basis that polymorphism is incorporated into parametric model system. Although polymorphism has found its success in software engineering, however, the viability of introducing polymorphism in architecture modeling is needs to be tested because while polymorphism increases flexibilities of architecture models it also increases the complication in the modeling process.

Further more, although the Context Responsive Data model eliminates many software engineering issues for scripting artists to manage large computation systems, but it still requires certain level of knowledge in Object Oriented Programming in order to understand the difference of scripting in a global level and scripting in a local level where procedures are encapsulated into portable objects. This requirement could exclude a large portion of designers as users.

The future work should incorporate this model in a specific parametric design tool, such as Generative Components, and distribute it to large varieties of designers to be used in actual design scenarios for evaluation, in order to discover the viability and potential of the model.
BIBLIOGRAPHY


