Context Dependent Model Visualization

By

Thomas Almy

Bachelor of Science, Mechanical Engineering,
University of Massachusetts Dartmouth, North Dartmouth Massachusetts, December 1995

Bachelor of Science, Marketing,
University of Massachusetts Amherst, Amherst Massachusetts, December 1985

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of

Master of Science

at the

MASSACHUSSETTS INSTITUTE OF TECHNOLGY

June, 1998

Signature of Author_______________________________
Department of Mechanical Engineering
May 8, 1998

Certified by_______________________________
Professor David Wallace
Esther & Harold Edgerton Assistant Professor of Mechanical Engineering
Thesis Supervisor

Accepted by_______________________________
Professor Ain A Sonin
Chairman, Department Committee on Graduate Students

I dedicate this thesis to my wife Mary R Almy. Any success I've had is due to her unyielding support.

Context Dependent Model Visualization

by

Thomas Almy

Submitted to the Department of Mechanical Engineering on May 8, 1998 in partial fulfillment of the requirements for the degree of Masters of Science

Abstract

As product development tools become more powerful, complex, and distributed, the need to simplify the vast amounts of information they provide becomes increasingly important. In addition, building in or adding features that make the development tool easy to use and understand is vital.

Moreover, different product development participants, such as designers, engineers, and managers, want to quickly locate the variables that impact the areas that interest them most and discover how the changes they make affect the rest of the product development problem. They also want to know how sensitive results are to changes in these variables. Given this capability, designers can quickly learn what they can affect and also what impacts them and to what degree. Determining a causality or dependency chain, as well as providing a sensitivity analysis are methods to help users visualize and understand large amounts of information and intricate relationships.

DOME (Distributed Object-based Modeling and Evaluation), a software design environment under development in the MIT CADlab, is an integrated product development tool that offers users powerful modeling and evaluation capabilities. DOME possesses many notable features and functions to aid the designer so they can make sense of the model and results. DOME uses "entity relationship" graphs as a default view, and employs different "lenses" so that the engineer can evaluate the design problem by applying different criteria. For example, the designer can use a lens to evaluate the performance of the design in terms of cost or safety.

The objective of this work is to add the ability of extracting qualitative information, such as cause and effect, from a DOME design model. This thesis considers three different DOME modeling scenarios when developing the algorithms to implement the features to accomplish this task. The first case is an explicit local DOME model. The second case involves deducing causality only knowing a list of equations and nothing else. The third explores how to extract a causality chain when parts of the model are linked to other software applications. By developing algorithms to deal with these circumstances, individually or combined, and by displaying the results effectively, people from different domains, such as designers and managers, will be able to easily perceive the information and intricate interconnections of complex distributed models in the best way possible.

Thesis Supervisor: David Wallace
Title: Esther & Harold Edgerton Assistant Professor of Mechanical Engineering

Acknowledgments

I would first like to thank my advisor, Professor David Wallace, for his time and support. I was immediately intrigued with Dave's work after he took the time to explain it to a fellow student and myself during an interview we conducted when I first started at MIT. Dave is very open and accepting of new ideas. I felt very flattered that he liked and even adopted some of my suggestions. He was also very easy to work with. I wish continued success for him and hope that his work, the DOME project, is fully embraced by academia and industry.

Next, I would like to thank Professor Warren Seering for hiring me as an RA. I will never forget the phone call I received Sunday evening in late July 1996. I listened to him describe a newly created center for product development. He then asked if I would like to be one of six graduate students to help set the research agenda for the center. He always treated me like I was a customer and was only concerned whether or not I was satisfied. I hope that the center becomes as famous and well known as I believe it will be.

I would also like to thank all the members of the CADlab for tolerating all my questions and giving me valuable feedback when asked. The CADlab members include Nick Borland, Shaun Abrahamson, Tina Savage, Jinpyung Chung, Jun Beom Kim, Francis Pahng, Shaun Meredith, and Nicola Senin. I really felt like I was part of team while working in the CADlab.

Finally, I would very much like to thank my family. My mother was very helpful and did what she could to make my life easier during the past 2 years. Most importantly, I would like to thank my immediate family, Mary and Tom. I have already dedicated this thesis to Mary but she was the reason why this all happened. I also hope that Tom did not miss me too much and that I can offer him a better life by completing this masters program.

Table of Contents

List of Figures

1

Introduction

1.1 Motivation and Goals

The DOME (Distributed Object-based Modeling and Evaluation) project team, working in the MIT CADlab, wishes to offer designers a product design tool to help them reduce product development time and improve product quality. In particular, they hope to provide designers with a tool that offers them much more power, freedom, and flexibility. The success of this undertaking is closely related to how well a completed, or partially completed, software model is understood, how easy it is for a given product development participant to use the tool, and how accurate/reliable the model and results are.

Given the complexity of models that are generated with new design tools, the fact that models are now distributed over networks, and that they can be linked to other high quality specialized design software tools, the need to gain insight into qualitative information, why results are obtained as opposed to what the results are, is more important than ever. For example, a design tool might be able to optimize a design and produce an optimal result, but it is difficult for the user to understand why or how this result was realized. In some cases a designer might want to know which variable affects a particular aspect of the design the most. If a design tool uses specs or utility functions to generate scores, it would be important to know for which values, or combination of values, does the design score dramatically worsen. This problem is compounded when the model is distributed. If the model is linked to several different software applications, referred to as a "black boxes", the explicit causality/dependency chain is lost. The designer does not really know which parent variable influences which dependent variable. In summary, designers would find causality/dependency and sensitivity features very useful.

This thesis will develop and add new capabilities to DOME. Specifically, causality/dependency theory, and design sensitivity analysis theory will be investigated and applied to DOME. "Knowing about causal or dependency relations among variables is essential in handling a number of practical problems such as diagnosis and prediction." [3]

1.2 Vision

The vision is to provide users of DOME with a deeper understanding of a complex distributed product development model. This will be accomplished by applying causality/dependency analysis, and design sensitivity analysis. Figure 1.1 is a graphical representation of this vision.

At the user's request, a number of options will be offered. A full dependency or causality map flowing from a selected variable will be produced, or simply a list of control parameters (i.e. roots, the top-level independent variables, not the complete chain) or influences (i.e. leaves, the very end of the chain, not the complete chain). As will be discussed latter, the user could choose to stop the chain at a number of different points, depending on what they are interested in. The goal is to help the designer better understand and utilize the DOME model by focusing on what matters to them. They will be able to focus in on sections of the model that relate to their areas of interest. The hope is to reduce the amount of time the originator, or first time user, spends getting to know and understand the model.

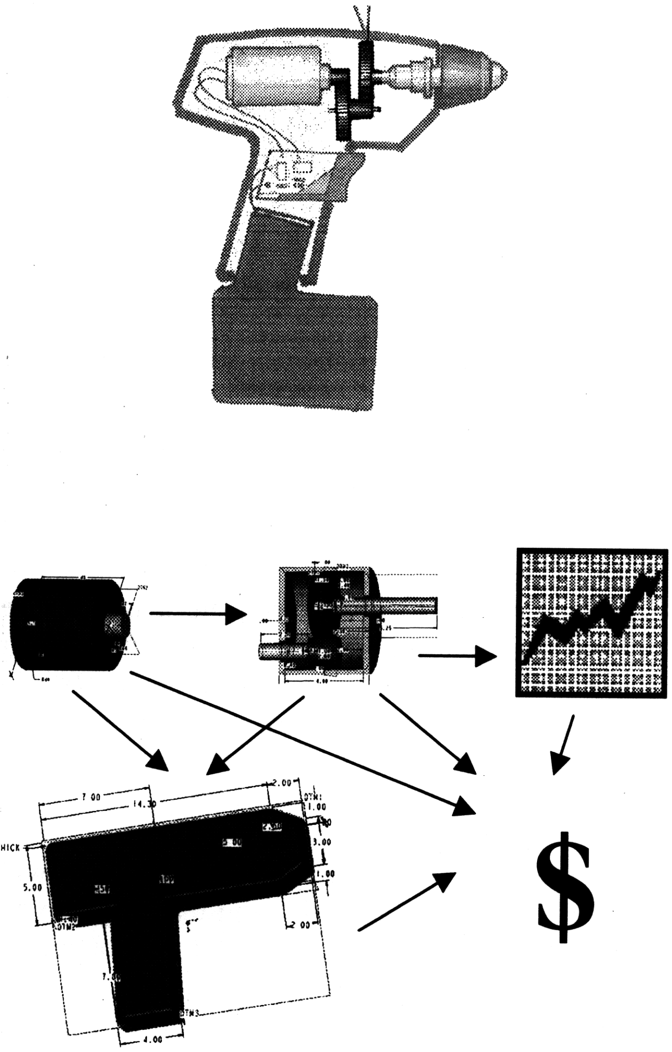


Figure 1.1 The vision of this thesis includes visualizing causality. This figure depicts the causality chain for a power drill.

Additionally, the user, on command, will be able choose a particular variable or design score and capture information on which variables influence them most. This is a sensitivity analysis.

Finally, causality/dependency chains of extremely complex models, such as automotive subsystems, might be difficult to make visible on a computer screen because of the large amounts of data present. For these circumstances, methods of displaying large amounts of information will be offered. An additional goal is to display this information in a way that best suits the particular user. For example, if a manager were using the tool, the information would be displayed so that they could understand it.

1.3 Problem Statement

The goal of this thesis is to define the algorithms that will carry out the above stated vision. The following problems are addressed and resolved:

1.3.1 Causality/Dependency Analysis

First, given a DOME model with an explicit child/parent chain, develop an algorithm that extracts the chain of interest. Second, develop a methodology to extract the chain if it is not made explicit, such as if only equations describing the physical phenomenon are available. Third, develop an algorithm to extract the chain if the model is distributed and linked to other applications. Next, determine which of the above stated scenarios are present and then take the appropriate action. Finally, discover the limitations of each of these approaches.

1.3.2 Sensitivity Analysis

Explore sensitivity analysis theory and ascertain how it applies to DOME. Next, investigate current methods of applying sensitivity analysis and finally, learn what the limitations are.

1.3.3 Visualization

Develop an effective graphical interface to make these features user friendly. Suggest a method of displaying the results. If the analysis produces a great deal of information find methods for displaying this information on a computer screen and look for how the information could be processed so that it make sense to the individual using the application.

1.4 Deliverables of this Thesis

First, an example of a complex DOME design problem model will be presented. It is a model of an LCD computer projector. This LCD example is given to illustrate why these functions are needed to diagnose the model. Furthermore, this example will illustrate how these features help the designer gain insight into the model. Lastly, the model will be used to illustrate how the features will be implemented.

This thesis will also present background information on the theory of causality and sensitivity analysis. In addition, there will be some investigation into data visualization. Next, this thesis will provide algorithms to carry out the vision. Finally, these theories will be integrated into the DOME application.

1.5 Organization of this Thesis

Chapter 2 begins with a brief description of the DOME framework and continues with an illustration of an LCD computer projector design model. This chapter will set the stage for why the features, discussed in previous sections, are needed.

Chapter 3 presents related work. The work of several key researchers in the field of causality and design sensitivity analysis will be discussed. Current theories and methods of data visualization will also be presented.

Chapter 4 deals with how to extract explicit causality/dependency chains from a DOME model and offers proposals for different use scenarios.

The fifth chapter offers algorithms for determining causality/dependency if the model is distributed, the "black box" issue.

The sixth chapter continues with a discussion on sensitivity analysis.

The seventh chapter discusses approaches of establishing causality if only equations are known. The chapter then concludes with an explanation on why this could be useful in the DOME architecture.

Chapter 8 summarizes the work presented in this thesis with conclusions and addresses implementation issues.

2

Background

There is a great deal of literature devoted to the subject of generating qualitative information. Some authors would argue that more emphasis is placed on describing models in terms of equations, focusing on relationships and results rather than how or why they came to be. Iwasaki and Simon stated that "in the world of engineering and science, many are interested in describing things in terms of equations and expounding on the relations that hold among parameters of objects that govern their behavior over time. The idea of causality is rarely made explicit. You should or must know about causality to be able to understand the phenomena." [3]

The need to generate qualitative information while using DOME software became very apparent when the models became large, complex, and especially when they became distributed. Before more is offered on the subject of qualitative analysis, and how it could be incorporated into DOME, the following sections provide a brief description of the DOME framework.

2.1 A Description of DOME

2.1.1 Overview

DOME (Distributed Object-based Modeling and Evaluation), a software application under development in the MIT CADlab [8] [14], is an integrated design tool that provides the user with a distributed and integrated modeling and evaluation environment. The vision behind DOME has broad applicability and encompasses many topics.

One driving notion behind DOME is that product modeling is more than just creating geometric or even solid models of objects. Design can include almost anything. The concept behind the technology is to approach each part/component/aspect of the design as an object, much like objected oriented programming. The objects can then be linked together to provide the user with a systemic view of a design model.

Furthermore, the vision includes creating a general all-purpose tool. Therefore, DOME can be put to use to solve many types of design problems. This includes resolving job shop scheduling problems [24] in addition to a wide array of engineering design problems. DOME objects can encapsulate just about every design activity, such as geometric design, costing, environmental impact, and design for manufacturing. These objects can be then liked together. DOME is very flexible and can be linked to other different applications, which is in step with the distributed vision. Even evaluation schemes can be regarded as objects so users can exploit any assessment or decision method as needed. The concept of objects can even be extended to include people. A design could include an engineer, specializing in a relevant field, to supply technical answers when needed. Their input would then be synthesized into the rest of the model.

Another goal of the creators of DOME is to allow users to quickly build design models. It would not be very beneficial if the user had to spend several months or years, making theses models. DOME models in fact, can be built rather quickly. Section 2.2 presents an example of a real DOME model that required about 35 person days to construct [25]. The ultimate goal of using DOME is to reduce product development time and to increase product quality. Developing the overall model quickly helps in this regard.

The first step in creating a DOME model is to make the assumption that design problems can be broken down into sub-problems. These sub-problems are then seamlessly linked together.

2.1.2 The Module; the Main Building Block of DOME

The main building block of a DOME model is the module. A module can represent a part of the design problem such as a motor (Figure 2.1). A DOME model could also contain modules of other components such as gearboxes, chucks, and rechargeable batteries, as well as more conceptual representations for issues like human factors. Developing these modules and actively linking them together actually facilitates and enhances design.

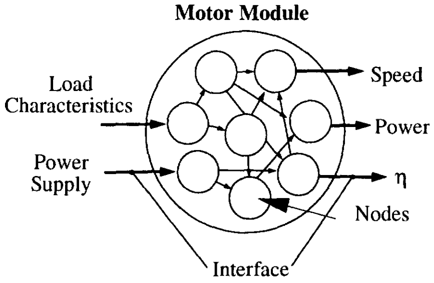
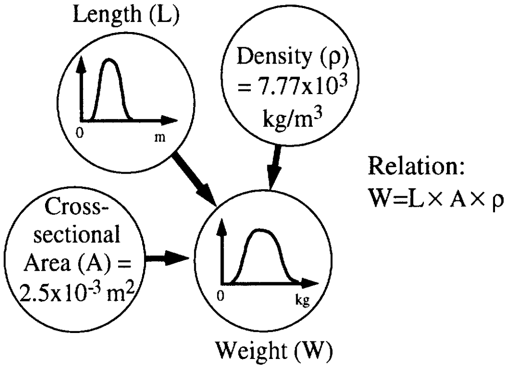


Figure 2.1 Simplified motor module: interface and embedded model [11]

2.1.3 Nodes

These modules in turn encapsulate nodes (Figure 2.1). Nodes contain basic data that can be deterministic or probabilistic, dependent or independent. They can even be linked to other software applications using CORBA [8] [18]. In this way, several different software models representing different pieces of the design can be linked together to form a system. Lastly, nodes can contain computer program, tables, or functions.

Nodes can also be linked together to form embedded models. Figure 2.2 shows how node "Length", that contains a beta distribution, can be multiplied by the node "Cross-sectional Area", containing deterministic value, and then multiplied by "Density" containing a deterministic value of 7.77x10^3 kg/m^3, to generate the distribution called "Weight". This is accomplished by using a Monte Carlo simulation. After accessing one of the nodes, Cross-sectional Area, for example, the user can then manipulate the current value to explore different design possibilities. This change in turn changes Weight. If Weight is connected to other nodes in the design, DOME will then propagate this change throughout. This is accomplished because the modules that encapsulate these nodes can then interact with one another to transmit the information (Figure 2.1). Information and results can propagate from the smallest detail of the design all the way up to the overall model and vice versa.



Length (L)
0
m
Density (ρ) = 7.77x10³ kg/m³
Relation:
W=L× A× ρ
Cross-sectional Area (A) = 2.5x10⁻³ m²
0
kg
Weight (W)

Figure 2.2 A model: variables and relations [11]

2.1.4 Catalogs

DOME modules can be replaceable. A set of similar modules can be grouped together to form a catalog. In this way, DOME has the capability of evaluating discrete module level changes in the design. Figure 2.3 shows an example of an assemblage of DC motor modules.

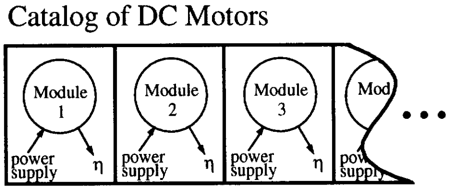
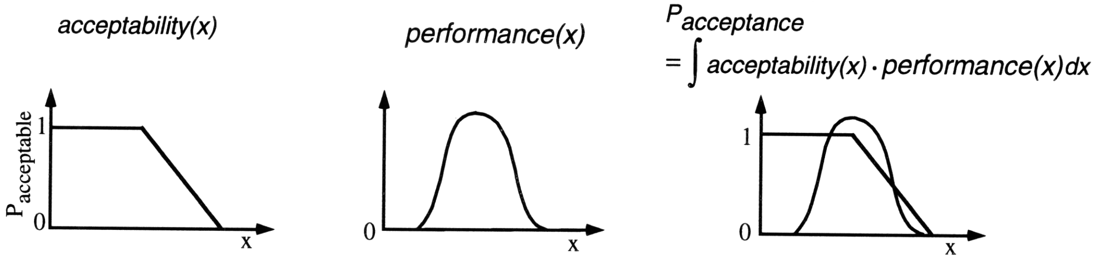


Figure 2.3 Catalog example of DC motors [11]

2.1.5 Evaluation

To evaluate a design, a decision support framework is utilized. Once the performances of the variables are attained they are evaluated against a set of specifications. These specifications are constructed as acceptability functions [16]. Next, all of the scores produced by this evaluation are then taken together to obtain a final score. Many other evaluation methods could be used. An example of an acceptability function is shown in Figure 2.4 (a) and an example of a performance is given in Figure 2.4 (b). This performance might be like the weight distribution obtained in the previous section. The formula for the probability of acceptance if given in 2.4 (c), as well as a graphic depicting the performance overlaid on the spec [16]. If the performance falls outside the acceptability function the performance has a zero probability of being accepted. This outcome would propagate through the entire design and result in a zero score for the whole design. If $P_{acceptance}$ is equal to one, then the probability of acceptance is unity. All results are given between 0 and 1 and indicate a degree of acceptance.



(a) (b) (c)

Figure 2.4 Evaluation Framework [16].

2.1.6 The Complete Design

All of the design modules including catalogs and static modules are encapsulated in the final design. Figure 2.5 shows an example of a top-level view of a DOME design model. The modules are located in the middle part of this figure, the lenses, which will be described in the next section, are located on the far right hand side, and the overall design score is located on the bottom right hand side of the figure.

2.1.7 Features

Dome possesses many notable features. One striking feature of DOME software is the capability of exchanging modules of designs or catalog items over networks or the web. This allows the designer to utilize designs or catalog items from all over the world and try them out in their design models.

Another capability of DOME is a multi-criteria search engine that allows designers to optimize their design [30]. Designers and engineers can optimize the model base on all the parameters or just a chosen few. Likewise, they can optimize independent parameters and catalog items. In addition, the designer will receive not just the "best answer", but also a number of very good

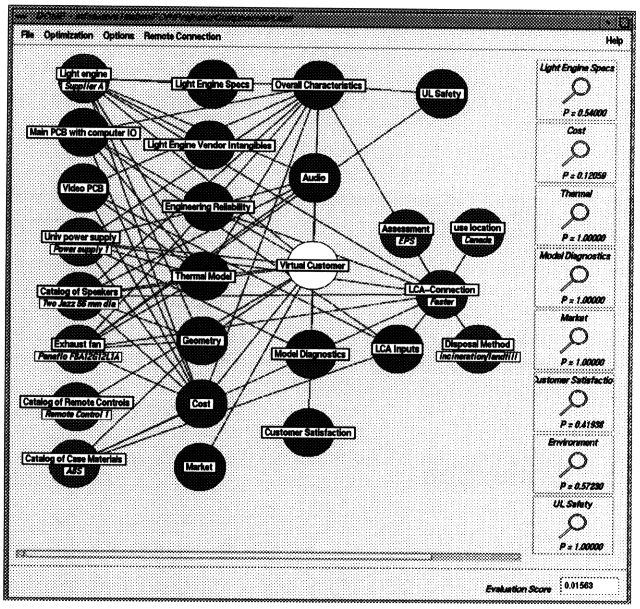


Figure 2.5 Top-level design view of a DOME model

solutions to choose from. This search engine reduces the chances of getting a local maximum and increases the chance of obtaining the optimal solution.

The current version of the DOME GUI (graphical user interface) displays a default graph view that shows the modules as part of an entity relationship graph (Figure 2.5). Each module can then be opened so that the designer can view its encapsulated nodes. More will be said about this in the next section when an example is presented.

Some features that DOME does not presently have are options that produce a complete dependency or causality chain. DOME also does not have the capability of performing sensitivity analysis. These two features would give the designer deeper insight into the model. Before proceeding with this topic, an example DOME design problem is presented.

2.2 An Example of a Distributed DOME Design Model-The LCD Projector

An example product design problem will be used to show what an actual DOME model might look like. This DOME design model will then be used to illustrate why it is necessary for DOME software to include features that can generate causality/dependency trees and perform sensitivity analysis. This example will also be used in later sections to describe how these features will be implemented.

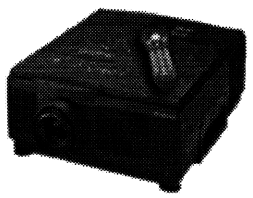
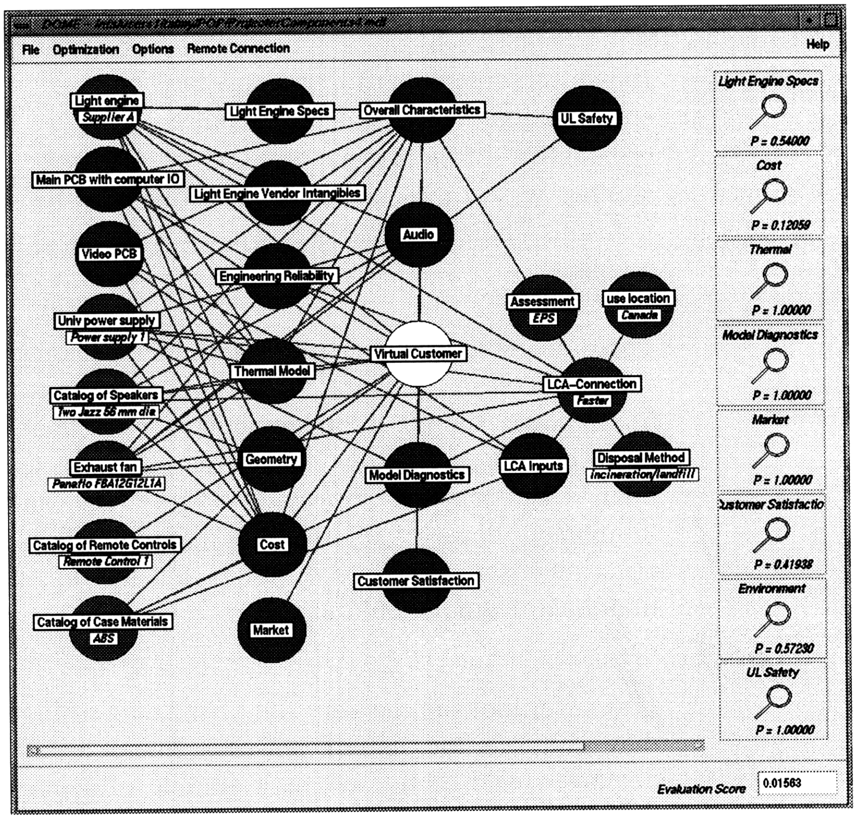


Figure 2.6 An example of an LCD projector

Recently, a proposal was made by an industrial sponsor of the MIT's Center for Innovation and Product Development (CIPD) to utilize DOME software to model an LCD computer projector (Figure 2.6). The goal was to use this DOME model to help the engineers reduce the development time of the projector and improve product quality. A complex distributed DOME model of the projector was then completed by a team of MIT students at the MIT CADlab. A snapshot of the top-level design view is shown in Figure 2.7.



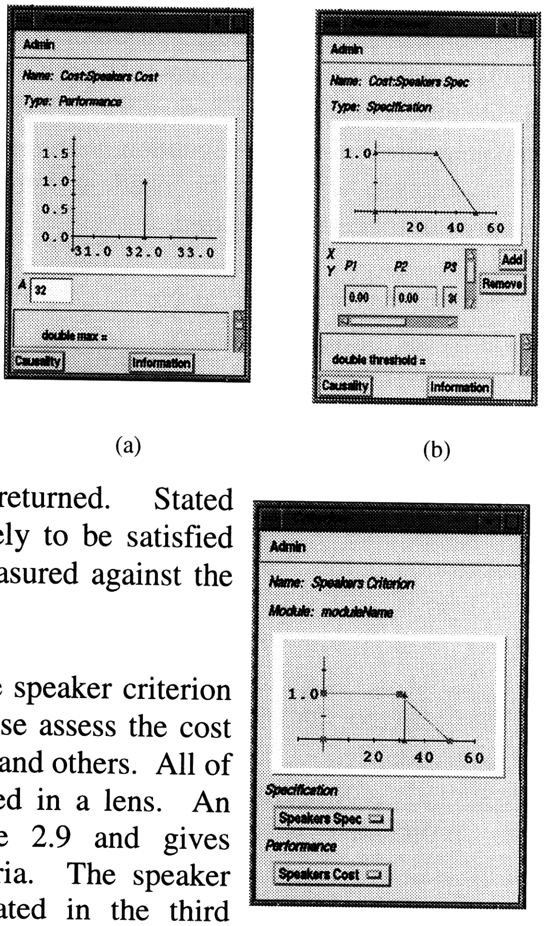
Light Engine Specs P = 0.54000
Cost P = 0.12059
Thermal P = 1.00000
Model Diagnostics P = 1.00000
Market P = 1.00000
ustomer Satisfactio P = 0.41938
Environment P = 0.57230
UL Safety P = 1.00000

Evaluation Score 0.01563

Figure 2.7 The DOME LCD projector model; top-level design view.

The model contains different modules representing aspects of the design. The modules representing all the major components of the projector are located on the left-hand side of Figure 2.7. Some of the modules are from catalogs and others are not. The catalogs include, starting from the top left corner, Light engine, Universal power supply, Catalog of speakers, Exhaust Fan, Catalog of Remote Controls, and Catalog of Case Materials. The names printed in small italicized letters, beneath the catalog names, indicate which particular module is currently selected. For example, two Jazz 56 mm dia is the current speaker configuration from the Catalog of Speakers. Also included in the model are non-catalog, or static modules such as Main PCB (printed circuit board) with computer IO and Video PCB. These eight modules represent what the engineers thought were important sub-problems, and were thus included. All of these modules are linked to other design modules, and this network of linked modules exchange services with one another. The designer can view Figure 2.7 to see what the present state of the design is. In the configuration shown, the design is utilizing the light engine from supplier A, power supply 1, two Jazz 56 mm dia speakers, the Panaflo FBA12G12LIA exhaust fan, remote control 1, and ABS for the case material.

Also included in the DOME model are other modules that perform services to evaluate the design. One example of this is the Cost module located in the lower left-hand side of Figure 2.7. This module receives cost information and, using several criteria, returns scores. For example, the cost of the two Jazz 56 mm dia speakers, $32.00 (Figure 2.8 (a)), is judged against a budget or spec (Figure 2.8 (b)), to formulate a criterion (Figure 2.8 (c)). Figure 2.8 (c) shows the cost superimposed on the specification. From this criterion, a probability of acceptance, 0.90, is returned. Stated another way, the designer is 90% likely to be satisfied with this choice of speakers when measured against the budget.


Admin

Name: Cost:Speakers Cost

Type: Performance

A 32

double max =

(a)

Admin

Name: Cost:Speakers Spec

Type: Specification

X Y	P1	P2	P3
	0.00	0.00	3(

double threshold =

(b)

Admin

Name: Speakers Criterion

Module: moduleName

Specification

Speakers Spec

Performance

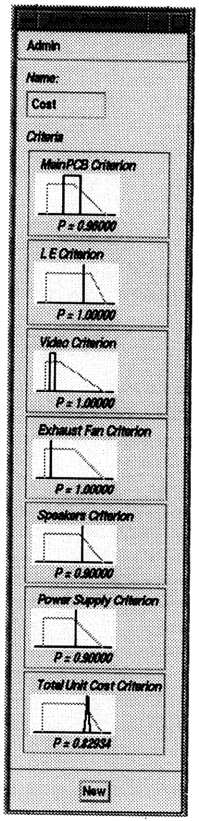
Speakers Cost

(c)

Figure 2.8 Speaker Cost (a), Speaker Cost Spec (b), and Speaker Criterion (c).

There are several other criteria like the speaker criterion encapsulated in the Cost module. These assess the cost of the power supply, light engine, fans, and others. All of these criteria and scores can be viewed in a lens. An open Cost lens is shown in Figure 2.9 and gives thumbnail views of all the cost criteria. The speaker criterion from Figure 2.8 (c) is located in the third thumbnail from the bottom.

If the user looks at the very right hand side of Figure 2.7, they can see the list of lenses. There are 8 lenses displayed, and each contains a graphic of a magnifying glass. These lenses include the Light engine Spec, Cost (Figure 2.9 shows the open cost lens), Environment, Thermal, and more. Each of the criterion scores


Admin

Name:

Cost

Criteria

MainPCB Criterion
P = 0.96000

L E Criterion
P = 1.00000

Video Criterion
P = 1.00000

Exhaust Fan Criterion
P = 1.00000

Speakers Criterion
P = 0.90000

Power Supply Criterion
P = 0.90000

Total Unit Cost Criterion
P = 0.82934

Figure 2.9 Cost lens.

encapsulated in the lens are multiplied together to obtain the overall lens scores which is displayed under the magnifying glass icon. The score *P* = *0 .12059* can be seen under the name Cost in the Cost lens (Figure 2.7). In a like manner, all of the final lens scores are multiplied together to obtain an overall design score, located in the lower right hand side of Figure 2.7. The final design score for this design configuration is 0.01563.

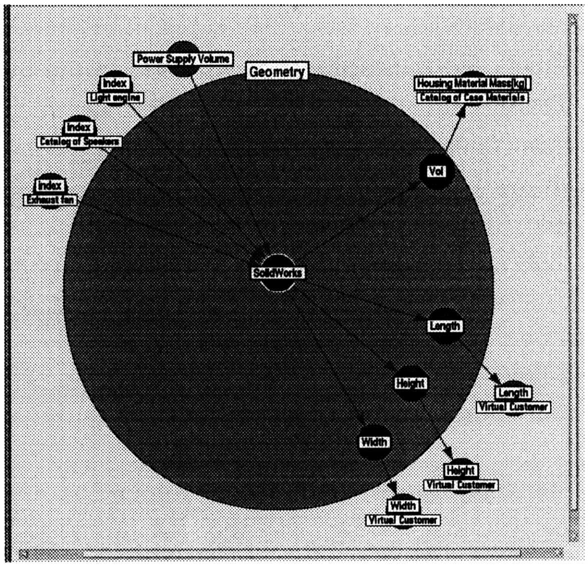


Figure 2.10 View of open Geometry module

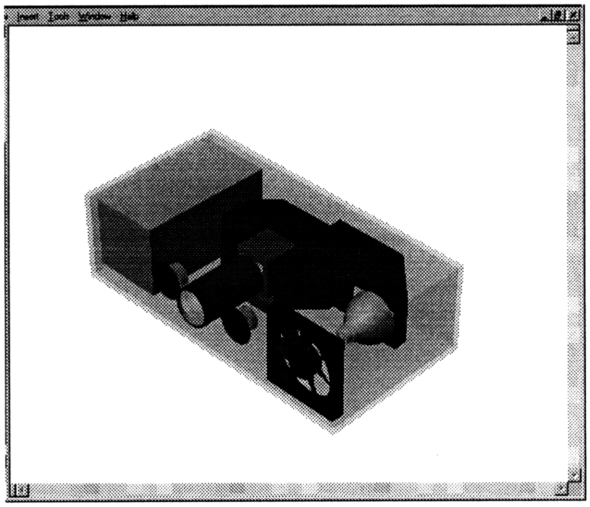
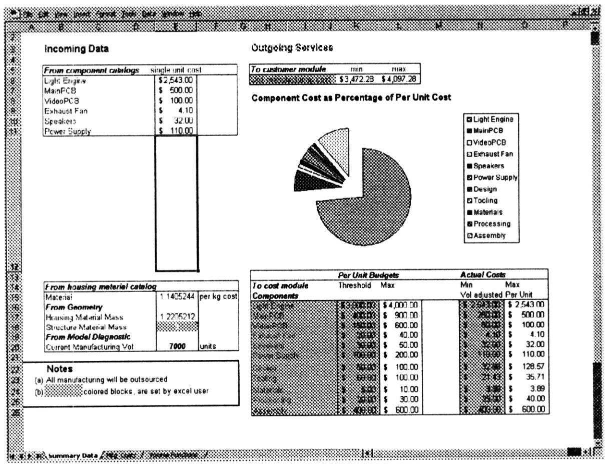


Figure 2.11 Solid Works® model of the LCD projector.

As described in the previous section, DOME has the flexibility of linking to other software applications. Accordingly, several applications were linked to this main model including Team®, an environmental assessment software package, Excel®, Solid Works®, a solid modeling software package, and another DOME model. These applications were linked because each possesses its own specialized functions. Also, the vision behind DOME is to allow designers and engineers from different disciplines to use the software tools they want. To illustrate this software integration, the geometry module from Figure 2.7 is opened (Figure 2.10) to show information entering the SolidWorks node located in the middle of the module. This node is remote and is linked to Solid Works®. The information from Power Supply Volume, index, from Light engine, index, from Catalog of Speakers, and index, from Exhaust fan, are then sent to Solid Works®. The geometry produced by this information can be seen in Figure 2.11. If the designer selects a smaller fan, the solid model would change and, if appropriate, propagate this information to the dependent variables Width, Length, Height, and Vol (Figure 2.10). In addition, if the user opened the Cost module (Figure 2.7), they would see a node called Excel Cost Model. Just like the SolidWorks node, this Excel Cost Model node is linked to an Excel® spreadsheet, which can be seen in Figure 2.12. Any change related to costing would concurrently change the Excel® spread


File Edit View Insert Format Tools Data Window Help
A B C D E F G H I J K L M N O P
Summary Data
Mfg Costs
Volume Functions

Incoming Data

From component catalogs	single unit cost
Light Engine	$2,543.00
MainPCB	$ 500.00
VideoPCB	$ 100.00
Exhaust Fan	$ 4.10
Speakers	$ 32.00
Power Supply	$ 110.00

From housing material catalog		
Material	1.1405244	per kg cost
From Geometry		
Housing Material Mass	1.2205212	
Structure Material Mass		
From Model Diagnostic		
Current Manufacturing Vol	**7000**	units

Notes
(a) All manufacturing will be outsourced
(b) ▨ colored blocks, are set by excel user

Outgoing Services

To customer module	min	max
[illegible]	$3,472.28	$4,097.28

Component Cost as Percentage of Per Unit Cost

	Per Unit Budgets			Actual Costs	
To cost module	Threshold	Max		Min	Max
Components				Vol adjusted	Per Unit
Light Engine	$3,000.00	$4,000.00		$ 2,543.00	$ 2,543.00
MainPCB	$ 400.00	$ 900.00		$ 280.00	$ 500.00
VideoPCB	$ 150.00	$ 600.00		$ 60.00	$ 100.00
Exhaust Fan	$ 20.00	$ 40.00		$ 4.10	$ 4.10
Speakers	$ 30.00	$ 50.00		$ 32.00	$ 32.00
Power Supply	$ 100.00	$ 200.00		$ 110.00	$ 110.00
Design	$ 50.00	$ 100.00		$ 32.86	$ 128.57
Tooling	$ 60.00	$ 100.00		$ 21.43	$ 35.71
Materials	$ 5.00	$ 10.00		$ 3.89	$ 3.89
Processing	$ 20.00	$ 30.00		$ 25.00	$ 40.00
Assembly	$ 400.00	$ 600.00		$ 400.00	$ 600.00

Figure 2.12 Excel spread sheet representing the projector cost model.

sheet. Furthermore, there is node in the module LCA-Connection (Figure 2.7) that is linked to Team® software. This way, the user receives real time environmental impact information as they make changes. Figure 2.13 shows what Team® software looks like. Finally, one might notice the light module called Virtual Customer included in the DOME model (Figure 2.7). This module is linked to another DOME model that addresses customer issues.

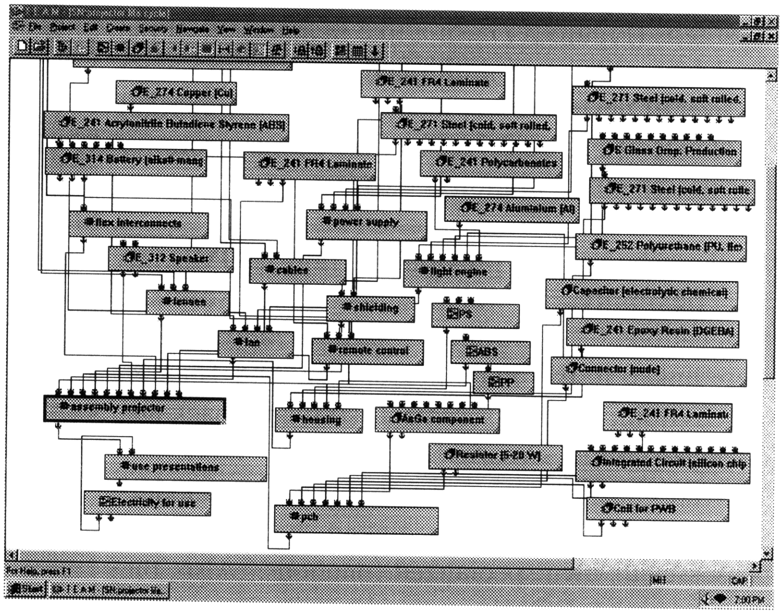


Figure 2.13 Environmental assessment using Team®

The model now gives the designer a systemic view of the LCD Projector. If there is a change, for example, if the designer chooses another light engine from the light engine catalog, they can see how that affects all the other aspects of the design like geometry and cost structure. The final overall design score confirms which design configuration is best. This then gives the designer a powerful tool to help then answer challenging trade off questions. Now evaluations of design iterations will take seconds or minutes instead of days or weeks.

2.3 Why Causality/Dependency and Sensitivity Analysis is Needed

After the LCD model was finished, many of the students examined and tested the model. Over all, the team was satisfied that the model was working well. A short time later, sensitively tests related to this thesis were preformed. The results did not make sense. When debugging the problem, it was found that the information returning from Solid Works® was unusable because only one unchanged value was returned no matter how much the design configuration was changed. Put in another way, node Vol (volume of the case) was not at all sensitive to the changes in the node Power Supply Volume. This was clearly wrong and important to know since many different variables depend on the value of node Vol. Furthermore, this problem had gone unnoticed during validation, but was quickly discover by performing sensitivity analysis. This example illustrates one reason why a sensitivity feature like this is important in the DOME framework.

3

Related Work

This chapter surveys past and present work related to the field of causality and dependency, sensitivity analysis, and information visualization techniques. There are many individuals in the field of qualitative research and visualization. Some of their ideas are useful and can be incorporated into the DOME framework.

3.1 Causality/Dependency Related Research

Herbert A. Simon is a famous researcher in the field of causality. His background is in the area of economics and artificial intelligence. In 1952 he established a method of determining causality solely based on a set of equations [23]. Brown and de Kleer summarized the method concisely by stating: "causal ordering (Simon) places an ordering on the variables. Oversimplifying, when a variable can be solved by simple substitution, it is causally dependent on the antecedents." [12]. Even though Simon's background is in economics, he argues that his approach can still be used in the field of engineering as well. He won a Nobel Prize for economics in 1978.

Yumi Iwasaki collaborated with Simon and produced several papers applying Simon's theory of causal ordering to practical problems [3] [7] [9]. She outlines a very thought provoking example that demonstrates that when one attempts to derive a casual ordering given a set of equations describing a phenomena, one's intuition might mislead them. More will be said on this topic in Chapter 7.

Brown and de Kleer take a different approach to qualitative research. They sum up the differences they have with Iwasaki and Simon by stating: "The difference between Johan de Kleer and John Seely Brown and Simon and Iwasaki concerning causality, modeling and stability, comes from the difference of concerns between engineering and economics. Johan de Kleer and John Seely Brown's notion of causality arises from considering the interconnections of components, not equations. The difference comes from a difference in point of view on the relationship between the structure of a system and the equations that describe its behavior. The difference stems from a difference in explicitness of the mechanisms and structural components underlying economic systems vs. engineered artifacts." [12]

Brown and de Kleer have written several papers on the subject of determining qualitative information from a "mechanism" [10][12]. In [10] they develop a theory they call qualitative physics. The point of qualitative physics is to predict and explain how a device or system behaves in qualitative terms. More precisely, they outline three goals for qualitative physics. The first is to make the theory far simpler than classical physics yet still able to characterize such things as state or momentum. The second is to present easy to understand causal accounts of physical devices. Finally, the third goal is to establish a foundation to enable designers and engineers to build models that will work well with the next generation of expert systems. [10]

In a summary, they employ the notion that physical systems are made up of many relationships in which a change in one or more variables produces a change in another. They go on to explain that the difference between qualitative physics and classical physics is in using continuous variables. In qualitative physics, each variable is described qualitatively by taking on only a small number of values such as a very small positive change, a small negative change, or no change. Further, they introduce what they call a qualitative differential equation:

$$\partial P + \partial A - \partial Q = 0 \qquad \text{Equation 3.1}$$

Part of this equation states that a change in A positively affects Q and negatively influences P. Finally, Brown and de Kleer discuss how qualitative physics helps in determining causality. This part of their research is applied to DOME and will be shown in Chapter 5.

One other important note is that this theory was put into practice and a computer program was created. It is called Envision. Envision has the capability, given an arbitrary device, to predict its behavior and also determine causality. The input to Envision (Figure 3.1) is much like a DOME model. It is a description of a system in terms of the topology and constraints.

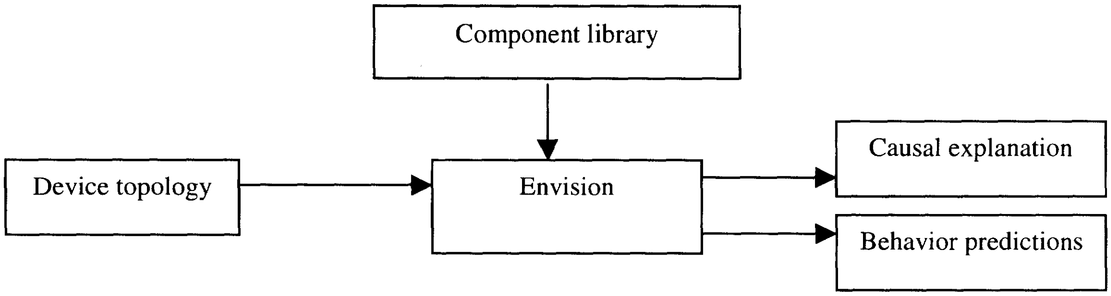


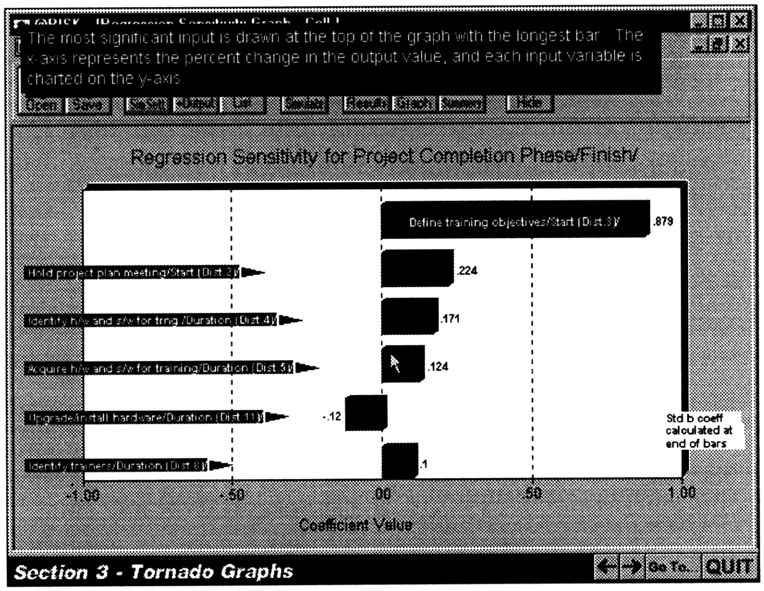
Figure 3.1 Envision [10]

3.2 Sensitivity Analysis

M. Kleiber et al. has written a book [21] on the subject of sensitivity analysis. This book outlines sensitivity analysis for linear systems as well as nonlinear systems. The book also addresses the difficult problem of conducting sensitivity analysis for nonlinear structural systems. Specifically, [21] focuses on solid mechanics issues and offers an approach to develop a sensitivity analysis for non-linear material and kinematic problems. Lastly, the author explains how sensitivity analysis can be used to streamline optimization by making it more efficient.

In [22] the authors develop a method for sensitivity analysis and optimization of discrete-event systems using the score functions (SF) method. They also touch on perturbation theory. The authors of this book demonstrate that the score functions method allows users to assess the performance of a system, produce data regarding all of its sensitivities, and to solve an full optimization problem by conducting simulation experiments.

While searching for software applications that possess a sensitivity capability, a product called PrecisionTree® was discovered [13]. It is an add-in package for Excel®. PrecisionTree® performs decision analysis so that the user can determine the best path to take when confronted with making an important decision. The software can even build a decision tree and influence diagrams. The application, more importantly, can perform sensitivity analyses.



The most significant input is drawn at the top of the graph with the longest bar. The x-axis represents the percent change in the output value, and each input variable is charted on the y-axis.

@RISK - [Regression Sensitivity Graph - Cell]
Open
Save
SimSett
+Output
List
Simulate
Results
Graph
Summary
Hide
Regression Sensitivity for Project Completion Phase/Finish/
Define training objectives/Start (Dist.3)/
.879
Hold project plan meeting/Start (Dist.2)/
.224
Identify h/w and s/w for trng./Duration (Dist.4)/
.171
Acquire h/w and s/w for training/Duration (Dist.5)/
.124
Upgrade/install hardware/Duration (Dist.11)/
-.12
Identify trainers/Duration (Dist.8)/
.1
Std b coeff calculated at end of bars
-1.00
-.50
.00
.50
1.00
Coefficient Value
Section 3 - Tornado Graphs
Go To...
QUIT

Figure 3.2 PrecisionTree®'s tornado diagram.

The authors explain that PrecisionTree® performs a one-way sensitivity analysis by changing one variable of the system at a time. The results of many one-way studies are then compared on the same chart called a tornado diagram (Figure 3.2).

Some professors at MIT have lectured and written about the subject of sensitivity analysis. Professor Odoni has the following to say about sensitivity analysis in the context of a parametric model: "…some of the input parameters and assumptions may be varied systematically over a range of values. How one interprets and uses the results of these sensitivity studies is very much a matter of judgment." [27]

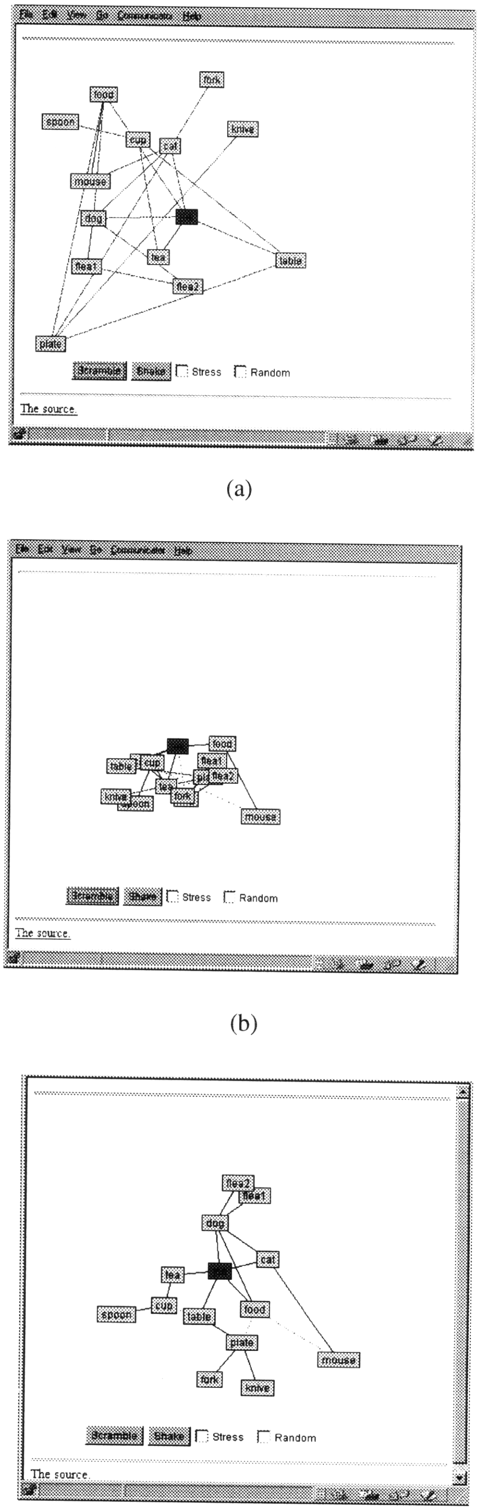
3.3 Visualization

When generating causality graphs it is important to think about how they will be displayed. It would be beneficial to find different approaches and techniques to visualize this data.

3.3.1 The Java Graph Layout Algorithm

A Java graph layout applet [17] has been utilized to deal with the simple issue of laying out a graph. This graph layout algorithm behaves very much like an overly damped first order system. The program works quite well arranging small arbitrary graphs with about twenty to thirty nodes. The lengths of the cords connecting the nodes are minimized to a preset value and the distances between the nodes are maximized. A sequence of screen dumps is given in Figure 3.3 (a) – (c) to show how this algorithm takes an arbitrary graph and continuously iterates until the graph is properly spread out. In the first figure the nodes are unfurled randomly. In Figure (b), the nodes are then pulled together closer and the algorithm continues to execute, minimizing the length of the cords while maximizing the distances between nodes. Figure (c) shows the final layout.

The graph layout algorithm can be used not just as a tool to help the designer lay out a complicated graph, but the algorithm can be modified to help cluster large amounts of data. For example, the nodes can be restricted to certain areas of the screen, which will help in clustering. Furthermore, the predefined lengths between the nodes can also be adjusted. Figure 3.3 (c), for example, shows flea1 and flea2 closer together than cat and mouse. This modification could also be used to cluster other types of nodes as well.



(a)

(b)

(c)

Figure 3.3 The Java graph layout applet. The initial random placement (a) The program executing (b) The final layout (c).

3.3.2 Other Visualization Techniques

Researchers at Xerox PARC are working on several different software applications to visualize data. The cone tree is a noteworthy example of using three-dimensional space to display large amounts of data. The authors of [1], who are also the inventors of this software, argue that being able to handle a large amount of data is a problem in large-scale cognition. "The task of managing and accessing large information spaces is a problem in large scale-cognition." [1] Figure 3.4 shows a screen dump of a cone tree. Animation is used to rotate the cone to help the user perceive the information and to track the relationships without thinking about it.

Using this 3-D approach one can display a great deal of data. However, the cone tree does not work well for directed graphs having multiple children and parents. The cone tree could nevertheless be used to display the graphs produced by causality analysis.

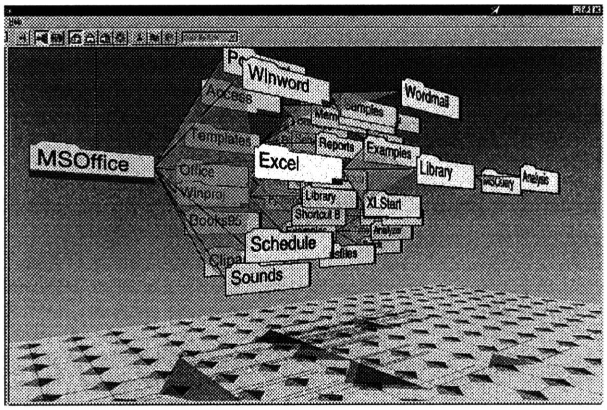


Figure 3.4 The cone tree.

Another application created to visualize data is called the perspective wall [8] (Figure 3.5). The strength of this application is in the ability to visualize linear structures. For example, the detailed front view of the wall might contain a page from a newspaper and the distorted views (the two other sides of the wall, left and right) might contain issues of the paper from different days. In this way, the documented is placed in a context. To view the other articles, animation is used to scroll the documents to the front view. It places the article in the center view and at the same time, places the article in a meaningful context. The perspective wall is good for displaying revisions of files like engineering drawings.

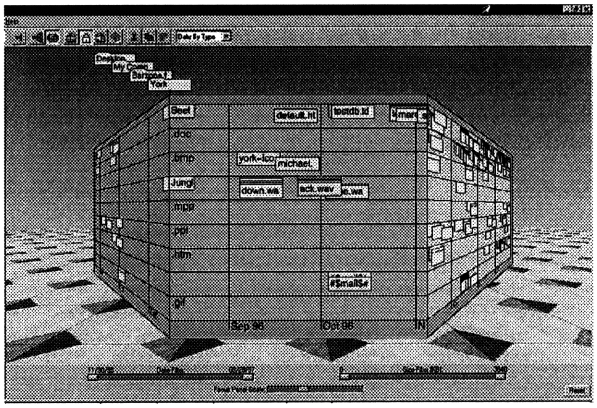


Figure 3.5 The perspective wall.

Yet another visualization approach is the hyperbolic tree [28]. The hyperbolic tree uses what is called a "focus + context" fish eye technique for visualizing and manipulating large trees. This application could be very helpful in displaying causality graphs. Like the perspective wall, the application allocates more display space to what the user is looking at in the center while keeping most all of the other information on the circumference. Again, like the cone tree, animation is used to navigate through the graph. The creators claim to be able to display about 10 times as many nodes effectively as conventional hierarchy viewers.

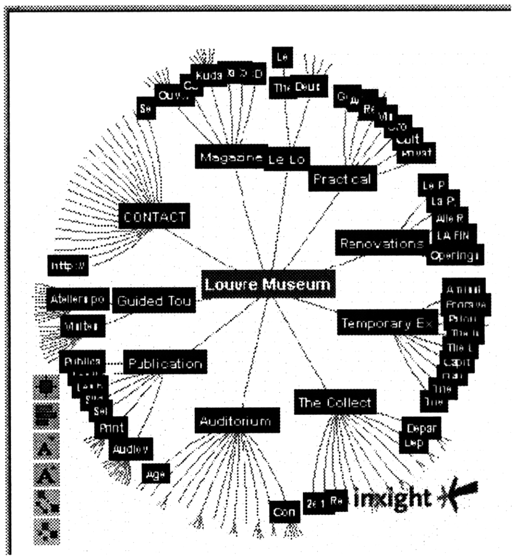


Figure 3.6 The hyperbolic tree.

3.3.3 Clustering Algorithms

Another approach to help visualize an overwhelming amount of data is to cluster it. In [31], for example, the authors deal with the clustering of related items. They employ structure-based and content-based clustering. Content-based clustering examines the content of the element, such as price range or actual cost and uses that attribute to group the nodes. Structure-base clustering, on the other hand, collects the nodes based on characteristics of the structure of the graph. For example, the algorithm consolidates some elements together if they have the same type of link. Many elements might be grouped together if each of their links end at the same destination.

The authors go on to explain how the users can interact with the program to help create the view they want. At this point in their paper they introduce a software tool they developed called Navigational View Builder. This tool gives the users the ability to interactively create overview diagrams. As with the other approaches previously discussed, they also make use of three dimension and interactive animation to help with the visualizations.

In [2] the same authors expand their scope to deal with the clustering of arbitrary graphs. Their aim is to provide users with alternative trees, each giving a different viewpoint to the underlying information to help the user better discern the data. They propose to run an algorithm that locates a root node, one that has no parents, then cluster groups of child nodes while cutting the links between these clusters. The result is a newly formed tree from the arbitrary graph. The algorithm then performs an evaluation to determine how good the structure of the graph is, symmetrical versus asymmetrical, and also rates itself based on how much information was lost.

Finally, in [26] Yang puts forth a methodology to acquire an overall global view of information that is somewhat similar to the others above. She utilizes three dimensions and uses a genetic algorithm to untangle the 3-D graph to present the data in the best way possible. As a result, global information and characteristics are revealed and, at the same time, the low-level details are still accessible.

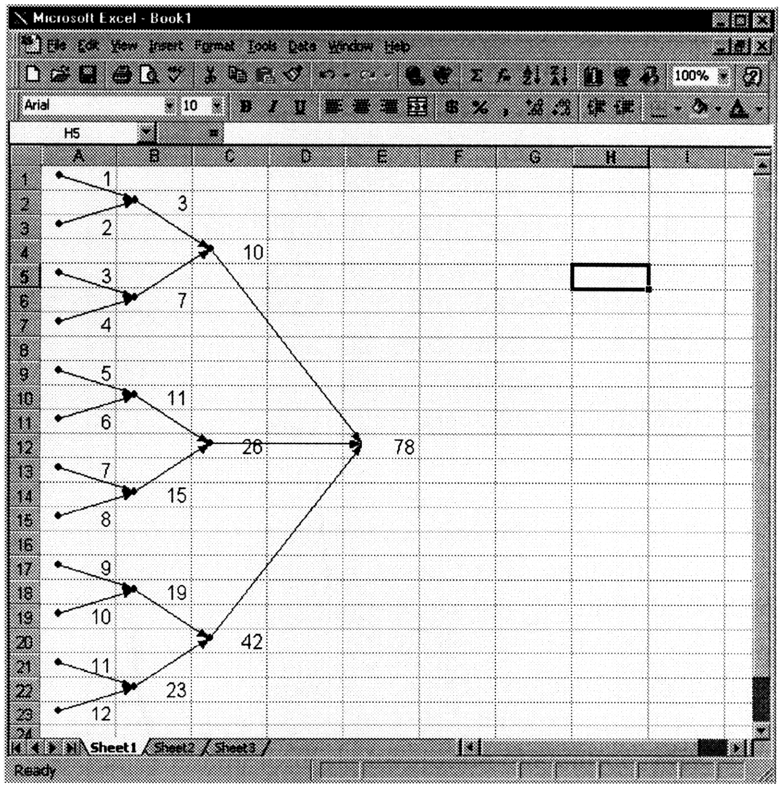
4

Causal Ordering Given a Predefined DOME Model

4.1 Overview

In many instances the mere act of modeling defines a casualty/dependency chain. When a model is built in Solid Works®, for example, entities are linked together so that if there is a change in one dimension, the power supply volume from Chapter 2 for instance, the other dependent dimensions, in this case the enclosure, will change accordingly. Similarly, creating a spreadsheet in Excel® and defining relationships such as cell A8 = cell B8 + cell C8 also imbeds a causal chain.

A question then arises, how does one extract this information from the application? Excel® has an option called Auditing that generates a causality graph when a cell is selected. The two choices Excel® offers are "trace



	A	B	C	D	E
1	1				
2		3			
3	2				
4			10		
5	3				
6		7			
7	4				
8					
9	5				
10		11			
11	6				
12			26		78
13	7				
14		15			
15	8				
16					
17	9				
18		19			
19	10				
20			42		
21	11				
22		23			
23	12				
24					

Figure 4.1 Excel's® "trace precedents" feature.

precedents” and “trace dependents”. These features only display the next level of parents or children, and thus are somewhat limited. For example, clicking on the cell containing 78 (Figure 4.1) produces lines and arrows from the cells containing 10, 26, and 42 only. To generate the other arrows, the user would have to select cells with the numbers 42, 26, and 10, and then repeat the command. To follow a chain all the way back to the beginning or end requires using the command, “trace dependents” for example, over and over again, which becomes very tedious. Figure 4.1 shows an example of a precedent tree generated using the “trace precedents” command ten times.

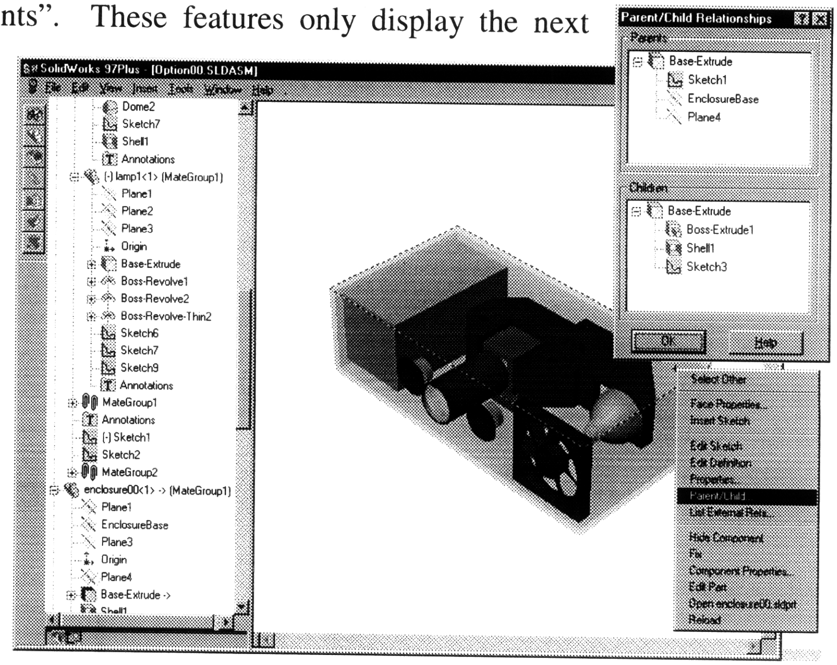


Figure 4.2 Solid Work’s® “Parent/Child” feature.

Similarly, Solid Works® has an option to display parent/child relationships. An example of this is shown in Figure 4.2. The selected object is the top face of the projector’s case, profiled by a dotted line. The popup window on the right displays what this face depends on, and what depends on it. Solid Works®, like Excel®, does not give the user a complete chain and consequently the user can only view the next level parent or child.

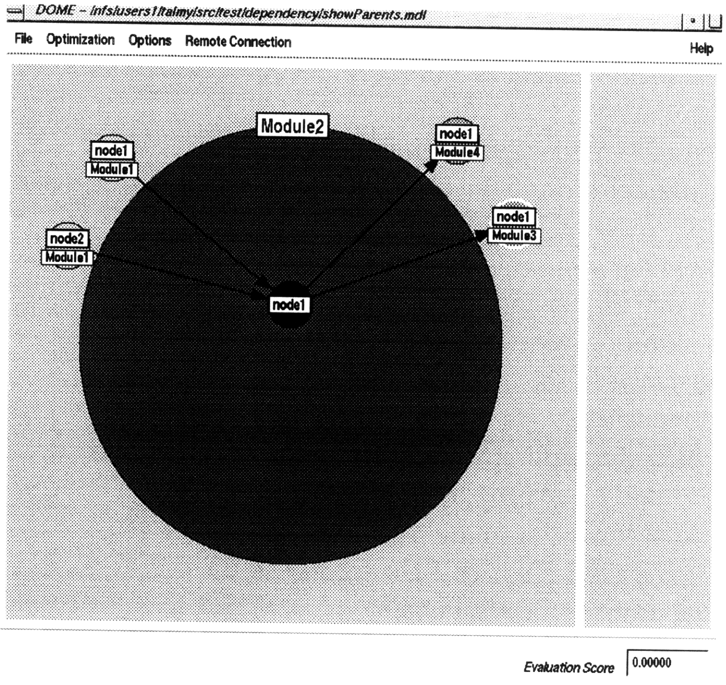

DOME - /nfs/users1/talmy/src/test/dependency/showParents.mdl
File Optimization Options Remote Connection
Help
Module2
node1
Module1
node2
Module1
node1
Module4
node1
Module3
node1
Evaluation Score
0.00000

Figure 4.3. An Open DOME module.

DOME software addresses causality in a similar fashion. Like using Excel® and Solid Works®, the act of modeling in DOME creates a causality/dependency chain. When viewing the model, DOME’s graphical user interface employs a standard or default view that shows the immediate next level child or parent relationship. To go to this view, the user simply opens the module and examines the contents. Figure 4.3 shows an open module containing nodes (see Chapter 2 for an example of the initial default view, Figure 2.5). Figure 4.3 indicates that there are parents because the cords that connect the nodes are unidirectional. The arrows pointing to node1, located in the center, verify that the connected nodes on the left, node1 and node2 from Module1 (the text box beneath the node name describes which module it comes from) are its parents. In the same way, the arrows pointing out of the module to node1 from Module4 and node1 from Module3 indicate that these are its children.

DOME however, does not have the capability of displaying the entire chain of dependency or causality with a click of a button. It takes much searching to find this information. Many steps are needed and the user must jump from a view of the open module (Figure 4.3) to the default view (see Figure 2.5), then back again. At the end, the designer has forgotten the complete chain unless they were recording the information by hand. This method does not give the user a one-time picture of the complete causality chain, and subsequently the user looses an important overall all view of causality/dependency. Also, obtaining this information multiple times would be very difficult to find and tabulate.

When generating qualitative information from the model it is important to generate a causal/dependency explanation whenever the user wishes. The authors of one paper wrote: "one of the central issues of qualitative reasoning is generating causal explanation in response to user's query." [5] Moreover, having the option to display the entire chain, a subset of the chain, or just the end points must be offered. This subject will be discussed in ensuing sections.

4.2 MDL (Modeling Definition Language)

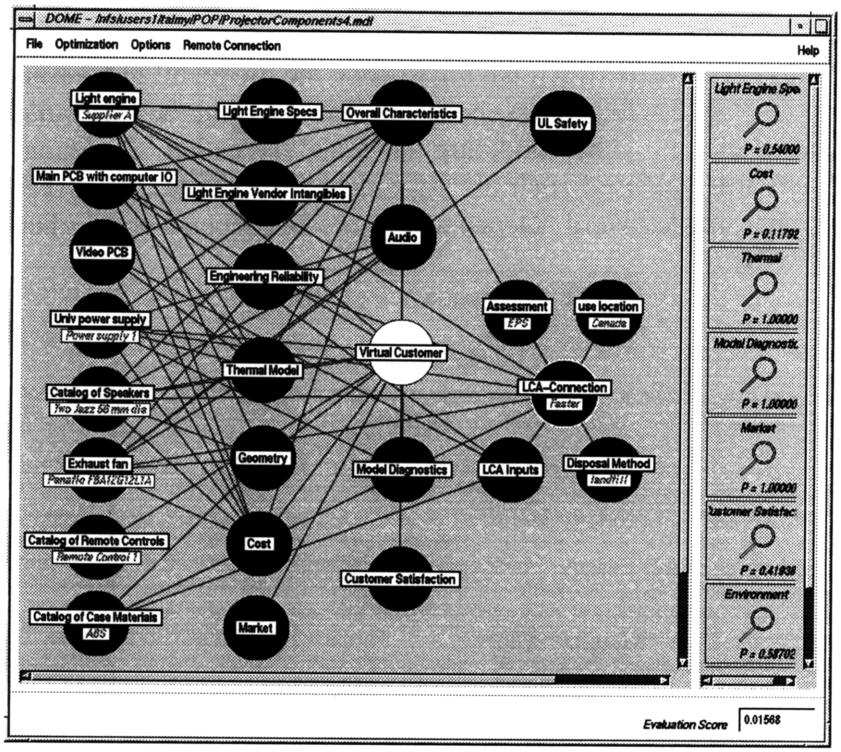
To show how the causality/dependency chain is established in the DOME framework, one can examine the computer code that originates the DOME model. DOME employs a script language to generate the design [4]. Having to write this simplified script language, or code, is a temporary step between having to code the model all in C++ and building the design graphically. Soon DOME will possess the capability of letting the user build a model graphically, by pointing and clicking. The modeling language is called Modeling Definition Language or MDL. Keywords are used such as "Module", for example, to instantiate a module object. In a like manner, the keyword "Dependency" creates dependencies. The following passage is an example of MDL:

```
Module "Dimensions"
(
        Variable "Length"
        (
                Delta {1}
        )
        Variable "Width"
        (
                Delta {1}
        )
)
Module "Area"
(
        SimVariable "Area"
        (
                Dependency "a" (ConnectedTo "Dimensions:Length")
                Dependency "b" (ConnectedTo "Dimensions:Width")
                (output = a * b;)
        )
)
```

The user has defined, in this section of code, two modules called Dimensions and Area. As can be seen, module Dimensions contains two variables, Length and Width. The module Area contains a SimVariable (simulated variable) that takes the Length and Width, and multiplies them together to obtain the output area. In this example one can see the word Dependency used under the keyword SimVariable. This sets the variable "a", located in Area, to be dependent on variable Length from module Dimensions. In the same way, variable "b", located in Area, is set dependent on Width from module Dimensions. The program that creates the graphical user interface uses this information regarding dependencies from the design problem model to display arrows indicating dependency (Figure 4.3). As a model grows, and more and more modules are added, many dependencies are established. The question now becomes how to extract a complete chain of these dependencies from start to finish. To create the proposed features and options as explained in the introduction, the same type of function that generates the first level dependency now will be modified. The function will be changed to provide a complete chain of information and to offer more options on how to display this information.

4.3 Proposed Causality/Dependency Graphs

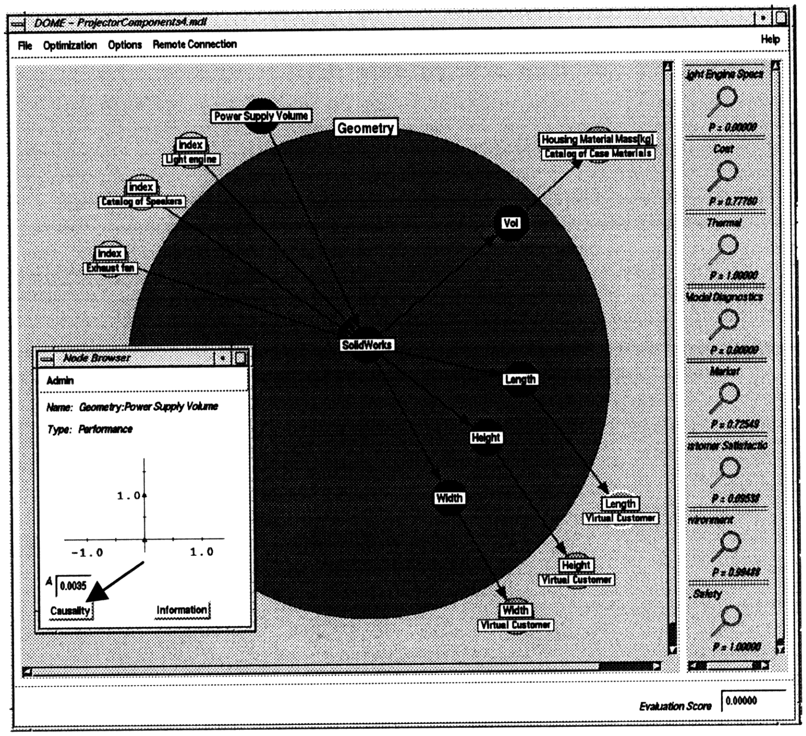
The proposed feature would enable the user to open a module, click on the node of interest, and then select an option to display causality and dependency in a number of different ways. The next figures presents how this vision might be implemented. First, when the user opens a DOME model, they are looking at the default view (Figure 4.4). For this example, the LCD model (See background) will be used. Next, they open a module. The module Geometry is opened for this example (Figure 4.5). Subsequently, the user selects the node/variable of interest. For this example, node Power Supply Volume has been selected. The Power Supply Volume node contains an independent continuous variable.



Evaluation Score 0.01568

Figure 4.4 Default view of the DOME LCD model.

The user would then click on the causality button in the node browser (see arrow Figure 4.5). Next, the user is offered a menu that lists a number choices in the way in which they can view causality (Figure 4.6). There are four main options and two of these have two sub-options. The first option produces a causality chain, the second shows control parameters (the roots or beginning of the chain), the third produces a dependency chain, and the fourth shows a pop-up window of influences (these are nodes located at the very end of the chain). In addition, the user can stop the chain at three different points. The first is to end the chain at the dependent variables and not include information regarding criteria or lenses. The second is to show the same chain and go one step further to include the criteria nodes. The third is to go yet another step to include criteria and the lens information. As can be seen from the diagram (Figure 4.5), Power Supply Volume does not have any parents, so the causality options are not applicable. The user therefore chooses the dependency option. Figure 4.7 shows a pop-up window containing the complete dependency chain, not including criteria nodes or lenses. From this display (Figure 4.7), generated from the design/MDL, the designer can see that Power Supply volume has dependents such as SolidWorks from module Geometry, and Excel Cost Model from module Cost.



Name: Geometry:Power Supply Volume

Type: Performance

Figure 4.5 Open Geometry module.

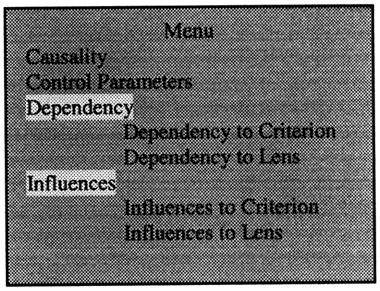


Figure 4.6 Proposed menu options.

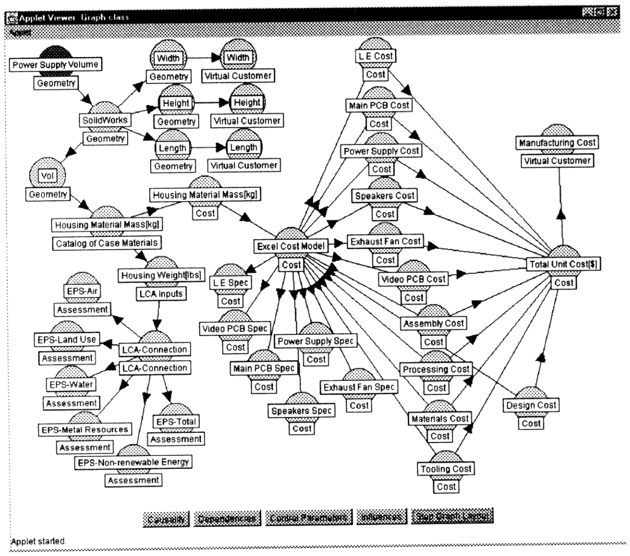
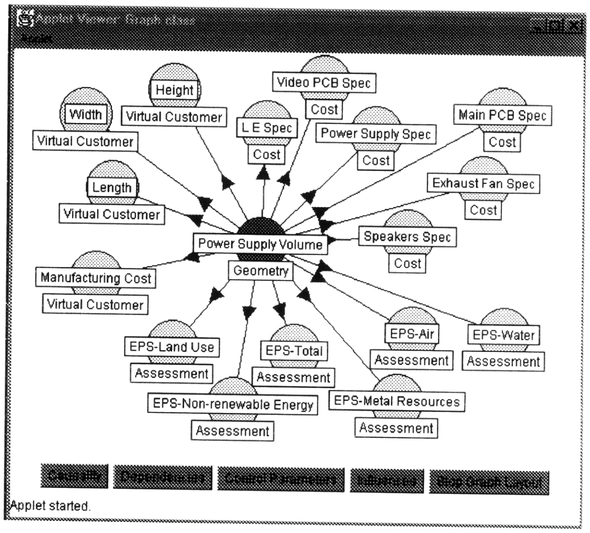


Figure 4.7 Complete dependency chain from Power Supply Volume.

Even though there is much useful information contained in this graphic, displaying the entire chain still might be too confusing to the user. Furthermore, they might just want to know ultimately what dependent variables would be effected by a change in Power Supply Volume. For this case the user has the option of choosing to see only the influences (Figure 4.6). The window in Figure 4.8 now displays the extremities of the Power Supply Volume's dependency chain.

As explained earlier, the user has the option to view chains including criteria and lens information. Figures 4.9 through 4.12 on the following page show the output of selecting these commands. The fact that the chains stop at different points might help the user gain additional insight into the model.



Applet started.

Figure 4.8 Pop up window: What Power Supply Volume influences.

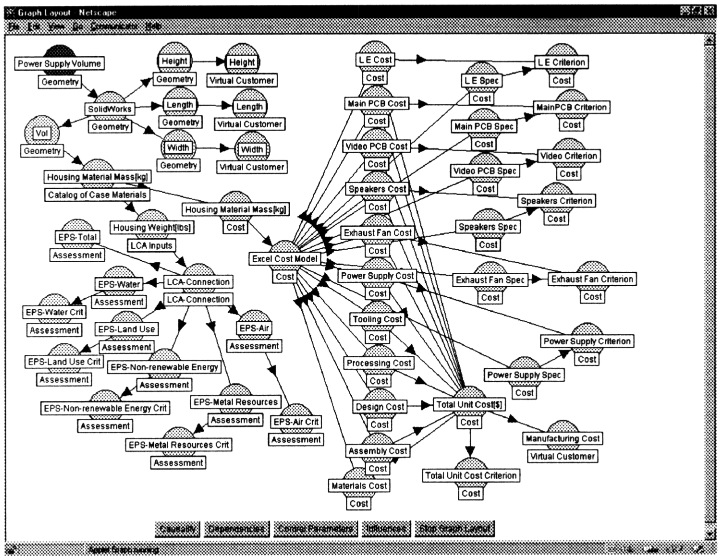


Figure 4.9 Dependency chain for Power Supply Volume including design criteria information.

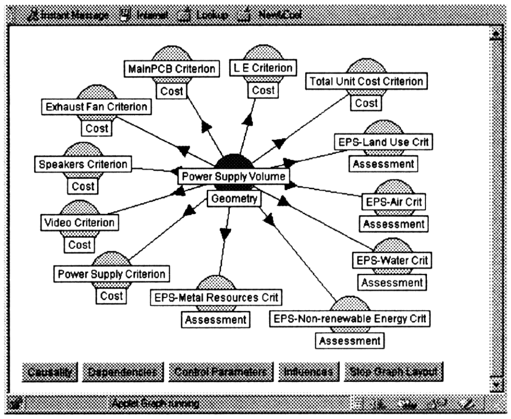


Figure 4.10 The design criteria that Power Supply Volume influences.

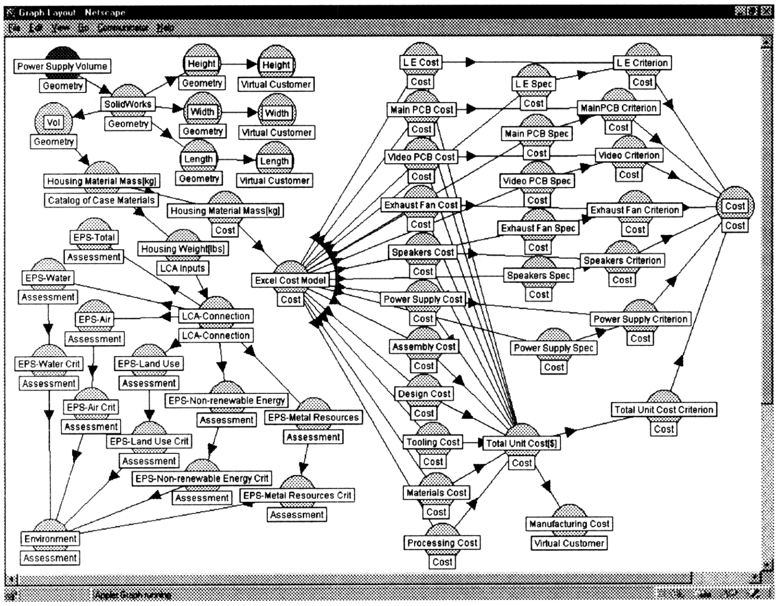


Figure 4.11 Dependency chain for Power Supply Volume including decision lens information.

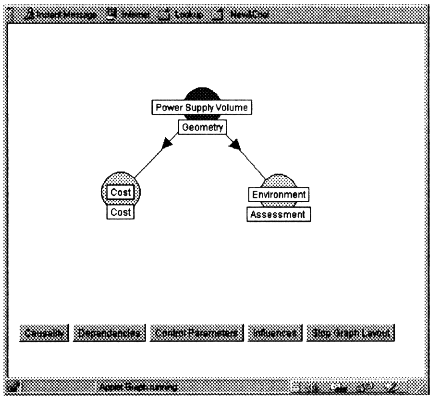


Figure 4.12 The decision lenses that Power Supply Volume influences.

4.4 Generating a Causality Chain Offers Another Point of View

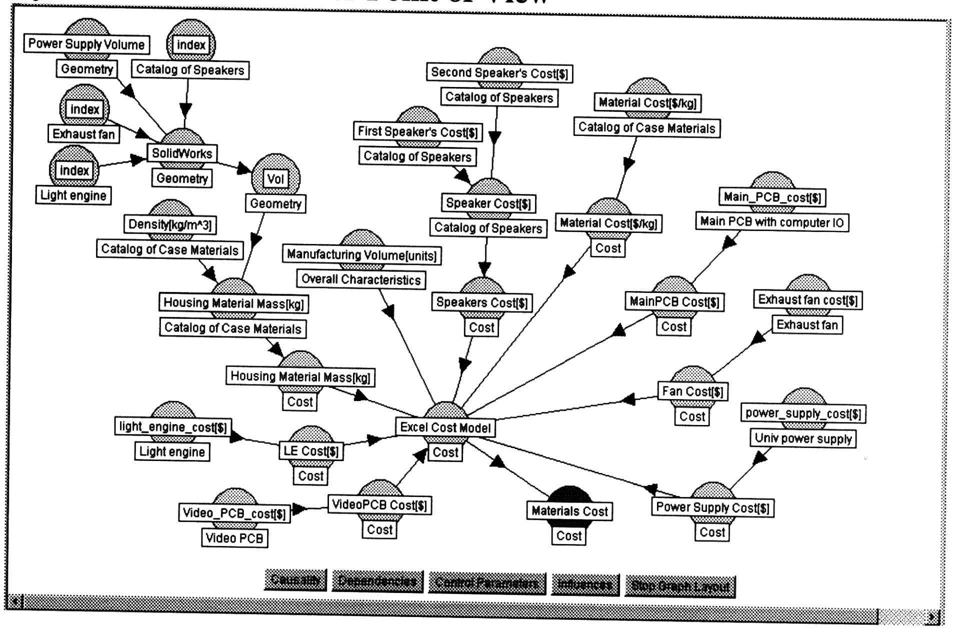


Figure 4.13 Causality chain of Material Cost.

The designer may notice that Materials Cost is one of the variables in the dependency chain located between Excel Cost Model and Total Unit Cost (Figure 4.7). To acquire an alternative view of cause and effect, the user could select node Materials Cost and choose the causality option. That would produce the graph seen in Figure 4.13. This gives the user a different perspective. Previously, looking at the dependency chain, the user could see that Power Supply Volume was one root parent of Material Cost, but now the user recognizes that Materials Cost has several roots or control parameters. Just as before, Figure 4.13 might still be too confusing. The user may just want to view the control parameters (roots) only. Now the designers could choose the Control Parameters option and view the output shown in Figure 4.14.

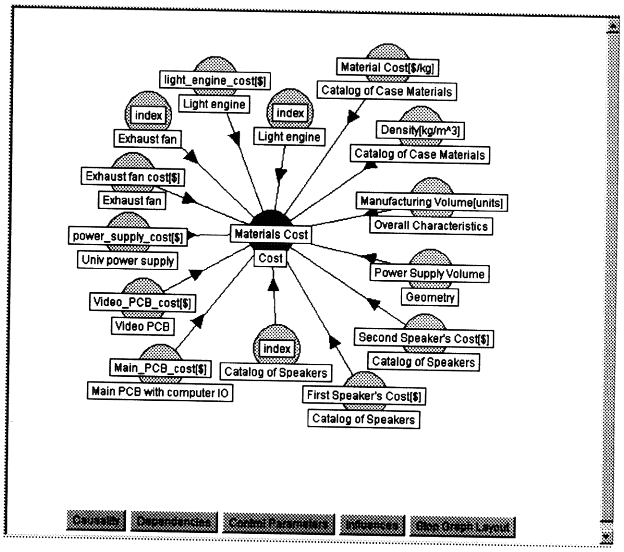


Figure 4.14 Control parameters of Material Cost.

4.5 Using These Features Offers New Insight into the Model.

Being able to instantly create these full dependency graphs provides the user with a great deal of insight. The dependency chain from Power Supply Volume (Figure 4.7) sheds much light on how the model is constructed. A first time user can instantly get the sense what is connected to what. With a few clicks of the mouse, the first time user can very quickly see what influence a particular variable and what that variable effects.

4.6 Using a Recursive Algorithm to Extract the Information

A recursive function is used to generate these chains. The information for parents and children are already data members of each node object. The function merely begins with the selected node and extracts the names of all its parents or children, and repeats this for each of those parents or children. Ultimately the function comes to an end when there are no dependency information left. Next, the information is transferred to a Java graph layout algorithm (see Chapter 3) that produces the pop-up window with the final graphs. The computer code for these functions and algorithms are given in appendix A.

4.7 Limitations

As can be seen from the MDL language, section 4.1, the key word "Dependency" defines a dependency between one variable and the next. For some reason the designer might define a variable with a dependency but not use the variable in an equation. For example, taking the slice of code in section 4.1 with one slight modification is as follows:

```
Module "Dimensions"
(
        Variable "Length"
        (
                Delta {1}
        )
        Variable "Width"
        (
                Delta {1}
        )
)
Module "Area"
(
        SimVariable "Area"
        (
                Dependency "a" (ConnectedTo "Dimensions:Length")
                Dependency "b" (ConnectedTo "Dimensions:Width")
                (output = a;) // previous was (output = a * b)
        )
)
```

As can be seen, variable b is not used in the calculation of "Area". However, using the causality/dependency algorithm just presented would report such a connection. A future improvement to DOME would make it impossible for a designer to create a dependency with out using it in an equation. The one flaw in this reasoning however, is that even if the designer makes use of the variable in an equation, the variable could be multiplied by zero thus producing the same effect. The line could look like:

```
(output = a * (b*0);) // previous was (output = a * b)
```

One last important note, when looking at Figure 4.7, the chain shows that Power Supply Volume connects to Excel Cost Model which then links to a number of specifications; LE spec, Video PCB Spec, Speakers Spec, and three others. Even though it is shown as such, the variable Power Supply volume does not influence these specifications in anyway. The links are correct but the causality is not. If the model is distributed, to Excel® in this case, then one really does not know if parent x affects child y. For these instances an alternative method to determine the true causality must be used. This method will be presented in the next chapter.

5

Causal Ordering Given a Black Box

5.1 Overview

The previous chapter introduced some limitations reporting causality/dependency based on the patents/child relationship contained in the design model/MDL language. This is because, as was explained in the Background section, DOME can be linked to other software applications [11]. In fact, DOME can be linked to other DOME models. The node/module where this connection takes place will be referred to as a "black box". Because of this, one does not truly know if parents of the black boxes influence its children. For example, Figure 5.1, taken from chapter 4, shows that several specifications, LE Spec (the light engine spec), Power Supply Spec, Speaker Spec and others, all from the cost module, depend on the variable Power Supply Volume. This pop-up window was created because that is how the user wrote the MDL file and subsequently that is how the design was created. The other illustration that makes these connections clear is the complete chain shown in figure 4.7. Following the chain (Figure 4.7) forward from the beginning, Power Supply Volume influences SolidWorks, SolidWorks influences Vol, Vol influences Housing Material Mass[kg], from Catalog of Case Materials, Housing Material Mass[kg] influences

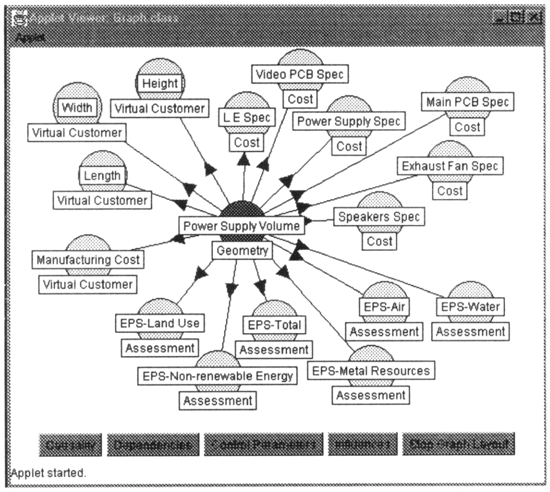


Figure 5.1 Pop-up window of influences from Power supply volume (what Power Supply Volume influences).

Housing material Mass[kg], from module Cost, this Housing material Mass[kg] influences Excel Cost Model, and Excel Cost Model influences each of the specifications. But in fact, the Excel Cost Model, is a black box. In reality, the cost manger, the originator of the Excel® spread sheet, influences the specifications through the Excel Cost Model node. Therefore, The Excel Cost Model node is the only parent to these specifications. The user might be confused and think that they can influence these specifications by changing the value of Power Supply Volume; this is incorrect. Power Supply Volume does not affect any of the six specifications. In addition, there are other nodes (Figure 4.7) that are not dependent on Power Supply Volume. In fact, the only variable Power Supply Volume affects after the Excel Cost Model node in Figure 4.7 is Material Cost. Already one can see how this could be definitely confusing for a first time user of the model.

Now the question arises, how to determine if there is a link between two variables? There are different methods that could be used. One is the token method. There is another method that will be referred to as the perturbation method. This method states that if one changes a value of a variable upstream, and the variable down stream changes, then one depends on the other. This method has roots in the work of Brown and de Kleer [10] [12]. The next section deals with the specific algorithm. In applying this algorithm, many different interesting and important issues are exposed. A different thought process will have to be applied to each and will be dealt with later in this chapter.

5.2 Generating a Dependency Chain through a Black Box

In words, the algorithm for determining dependency through a black box is as follows: First the variable of interest is selected. Power Supply Volume is again selected for this example. Next, determined if the node is from a catalog or not. Then, obtain the list of Power Supply Volume's children. While saving this dependency information, each child is checked to see if they are remote. Currently, node objects do not have data members to indicate whether they are remote or not. Each node does however contain a documentation object. The keyword "remote" was added to the documentation of remote nodes so that the algorithm could ascertain whether or not the node was connected to another application. If the node is not remote, the chain is continued. If the node is remote, then other steps are taken.

The first step is to obtain the list child variables of the remote node. Before proceeding with each link of this part of the chain, the value of the initially selected node, Power Supply Volume, is changed slightly. If Power Supply Volume were a member of a catalog, the whole module would be replace. If there were a corresponding change in one of the children, indicating a dependency, the algorithm would continue with that part of the chain. If there were no affect, that part of the chain would end. This algorithm produces the true dependency chain and the code for this algorithm is located in Appendix A.

5.3 The Resulting Chain after Applying the Perturbation Method

Figure 5.2 shows the results of this algorithm when applied to the Power Supply Volume. This window can then be compared the chain produce in Figure 4.7. This new graph displays surprisingly fewer dependencies than previously thought. Likewise, the new graph of

Influences is shown in Figure 5.3. There is also a marked difference between this window and the previous one shown in Figure 4.8.

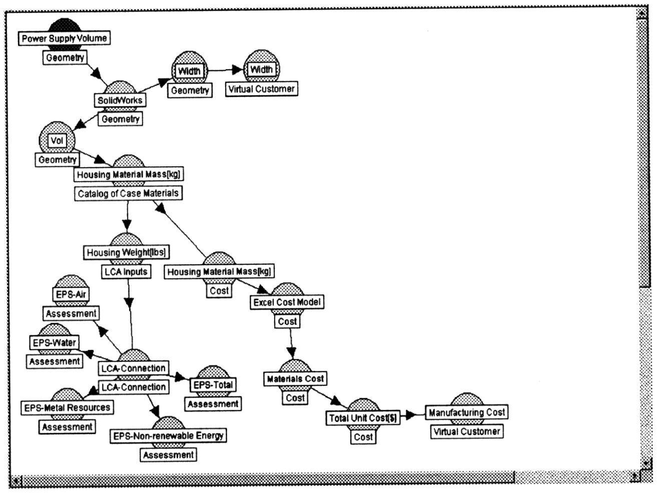


Figure 5.2 Resulting Power Supply Volume chain produced by the perturbation method.

5.4 Generating a Causality Chain through a Black Box

Going up the chain and determining causality can be dealt with in a similar manner as generating dependency. The algorithm begins with the selected node. First information regarding the node's parents is obtained. While doing this, the algorithm checks to see if the parent is remote. If none of the parents are remote, the algorithm continues up the chain until there are no more parents as in Chapter 4.

If one of the parents is remote, the algorithm follows and remembers each individual link up the chains until a root is reached. At this point it is determined if the root is from a catalog or not. If it is not, its value is changed slightly. If it is from a catalog, the algorithm cycles through the catalog items. If the initially select node changes due to a change in the root, then algorithm reports the causal information for that particular link. If there is no influence, then information pertaining to that part of the chain is disregarded. The code for this algorithm is also located in appendix A.

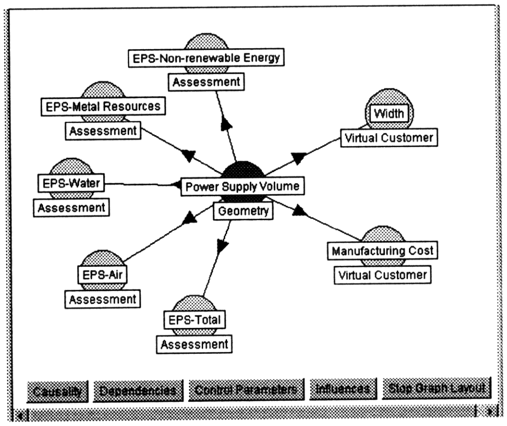


Figure 5.3 Influences from Power Supply Volume using the perturbation method.

5.5 The New Resulting Causality Chain

Figure 5.4 shows the new resulting causality chain. The difference between this tree and the one in Chapter 4 is that it does not include Main PCB cost, Video PBC cost, and Manufacturing Volume[units]. The designer now knows what Material Cost's true parents are. The pop-up window displaying the new set of

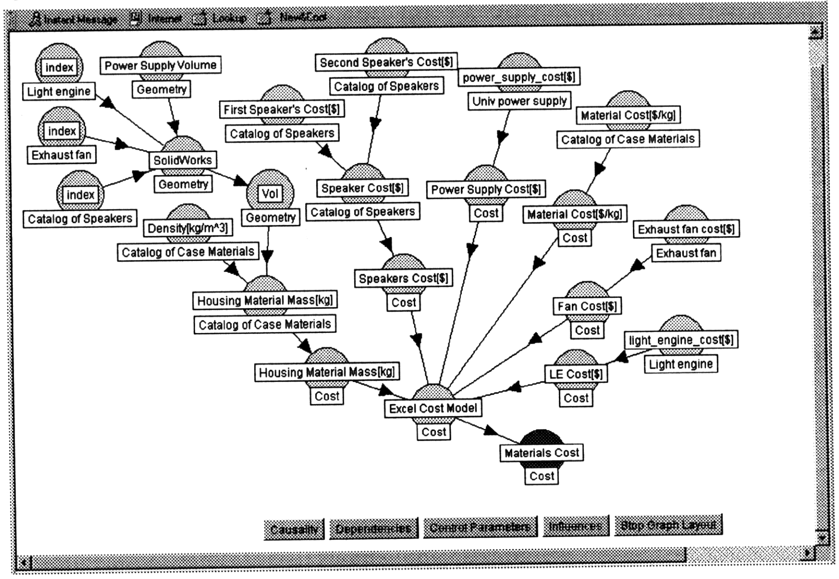


Figure 5.4 New causality chain for Material Cost.

control parameters is shown in Figure 5.5. This figure can also be compared to the graph produced in section 4.2 to reveal measurable differences.

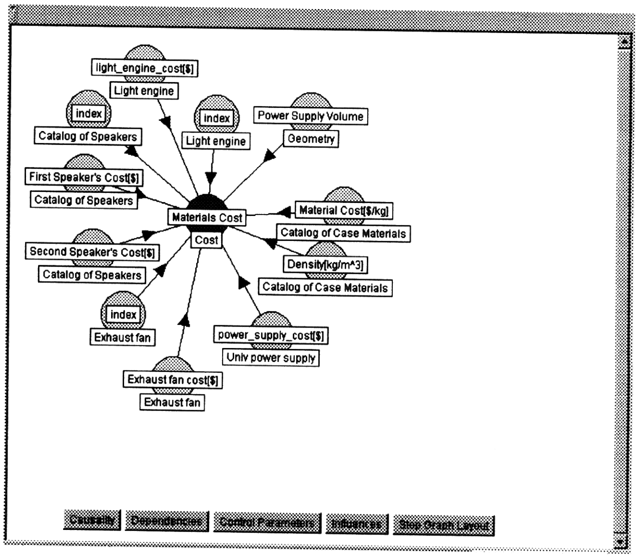


Figure 5.5 Control parameters of Material Cost produced by the perturbation method.

5.6 Limitations

One limitation of the perturbation method is in determining what constitutes a change. To explain, some of the performance distributions contained in the nodes are generated using a method call a parametric Monte Carlo simulation [29]. The distribution contained in node EPS-Air from Figure 5.3 is an example of this. If a root of EPS-Air were changed to the same value, in other words not changed, (if Power Supply Volume is change from .035 to .035, meaning no change) the distribution contained in EPS-Air would change. Such a change would be detected in the algorithm just present. Therefore some interval or tolerance must be established so that one can be sure that a change in a parent results in an actual change in a dependency and that the value is not just varying due to the simulation method.

The way to determine if there has been a change depends on the number of simulations conducted [29]. According to [29], if twenty simulations were completed, the mean of the resulting distribution could vary plus or minus 10 percept. However, if 100 simulation were conducted, the range would be reduced to plus or minus 2 percent. This added constraint can be use to determine if a dependent variable generate by such a simulation method is actually changing or not. For the above example in section 5.3, the variables related to the environment, EPS-Air, EPS-Water, EPS-Metal Resources, EPS-Non-renewable Energy, and EPS-Total were all influenced significantly by a change in Power Supply Volume. Therefore, there is a causality link between them. Interestingly, the sixth dependent variable, EPS-Land Use from 4.2, was not influenced (see Figure 5.3). This demonstrates the usefulness of the perturbation method because in fact there is a problem with the EPS-Land Use distribution. The calculations are not correct and only a value of zero is returned. Returning a zero value in this case is a perfect score. It means that there is no detrimental effect to the environment. The modeler now realizes there is a problem and can take steps to fix it.

5.7 How This Algorithm can be Extended to Carry out Sensitivity Analysis

Another technique to obtain qualitative information from a design model is to determine sensitivity. With the algorithms already presented it would not be too difficult to extend their capabilities to produce such an analysis. In the sections above, the goal was to discern if a child variable is sensitive at all to a change in the parent. It would not be difficult to determine, not just if the child is sensitive, but by what degree. This subject will be covered in Chapter 6.

6

Sensitivity Analysis

Sensitivity analysis is another way of obtaining qualitative information from a design model. Some authors would hold that the ability to carry out this analysis is extremely important. "The sensitivity of a structural system to variation of its parameters is one of the most important aspects necessary for a proper understanding of a systems performance. In fact, it is now widely acknowledged that any reliable approach to practical structural engineering problems should provide the analyst with an assessment of parameter sensitivity" [21].

6.1 Proposed Method of Sensitivity Analysis

Sensitivity is the partial derivative of a one variable with respect to another [21]. In the simplest case, the function describing the system must be defined and sensitivity can be characterized as a small change in the independent variable divided by the resulting change in the dependent variable (the slope), holding all other variables constant. DOME presents a challenging case for sensitivity analysis because although the parameters are known, the functions are hidden. Moreover, the designer might want to know the sensitively over a range of values and not just for one interval. For example, the environmental designer might want to have some information regarding how the Environmental lens score changes as Power Supply Volume changes through a series of values. Moreover, the issue of catalogs must be taken into account.

Therefore a slightly different approach is offered when tying to incorporated sensitivity analysis in the DOME framework. In the background section, professor Odoni suggested that it should be up to the user to determine what to do with the results of a sensitivity analysis: "...some of the input parameters and assumptions may be varied systematically over a range of values. How one interprets and uses the results of these sensitivity studies is very much a matter of judgement" [27].

The overall goal of sensitivity analysis, in the context of DOME, is to convey information on the degree to which a parent variable or catalog affects the selected dependent variable, criterion, or lens score. To implement this in the DOME framework the following algorithm is proposed.

First, after selecting the variable, criterion, or lens score, obtain a list of its control parameters. The algorithm presented in Chapter 4 has the capability of doing this. Second, vary one parent variable at a time while keeping the rest of the model constant. If one or more parents are from a catalog, then cycle through the entire catalog including all hierarchies. If the parent is not from a catalog, then vary its value (vary its value if it is a deterministic variable or move the mean if it is a distribution) plus or minus 10 percent of its present value.

This algorithm produces data including both the values of the changed independent parent variable, and the corresponding outcome values of the dependent variable. Figure 6.1 shows lens scores corresponding to each module selection from the catalog Assessment. The lens scores will be presented in ascending order.

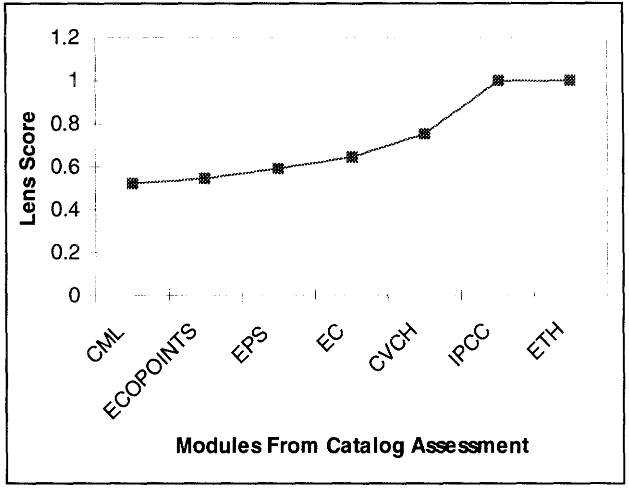


Figure 6.1 Environmental Lens score of every module in the catalog Assessment.

As stated earlier, the aim of this analysis is to shed some insight into sensitivity. Therefore, some value should be used to characterize the data obtained from a graph like the one in Figure 6.1. Accordingly, a proposal is made to use the slope of a best-fit line through the data to indicate sensitivity. Then, all of the slopes from the graphs of all the control variables will then be put together in one chart.

To display these results a tornado diagram is used (See Chapter 3). A complete sensitivity analysis example is given in the next section. The code that detects the control parameters and generates the data points such as in Figure 6.1 is located in Appendix A.

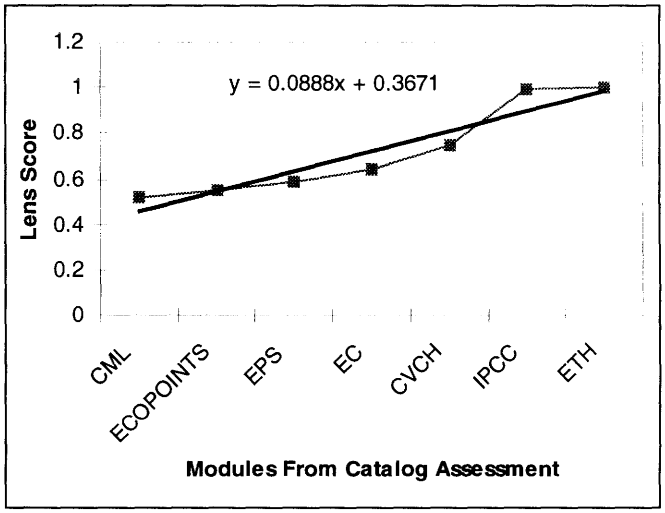


Figure 6.2 Best-fit line through Figure 6.1's data points.

6.2 Proposed Use Scenario: Sensitivity Analysis of the Environment Lens

To perform a sensitivity analysis the user should be able to select a variable, which could include nodes, dependent variables, specs, criteria, or lenses, and choose the option for sensitivity analysis from a menu. As with other features, the user might want to display sensitivity analysis in several forms. A pop-window should then be displayed to show the results of the analysis.

In this section the lens Environment has been selected to illustrate the sensitivity feature. As said previously, the slope values were used to gain an understanding of the sensitivity. Figure 6.3 shows the results of a sensitivity analysis performed for the Environmental lens (The parent variable Power Supply Volume was not included in this graph because its slope was extremely large and would obscure the other variables in the tornado diagram).

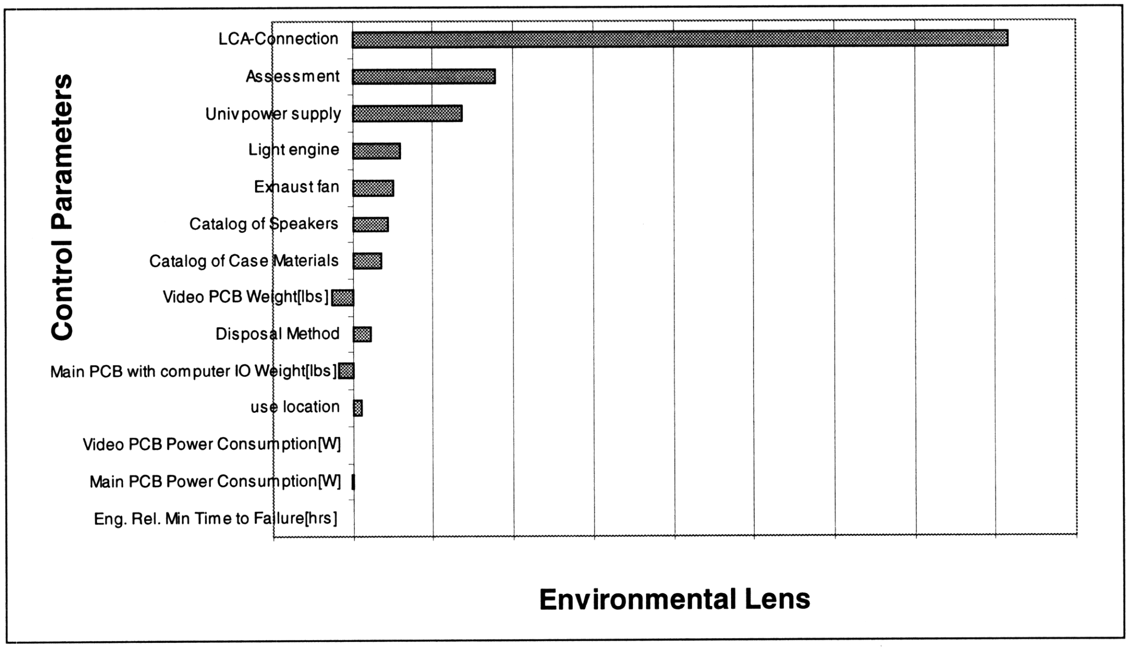


Figure 6.3 Sensitivity analysis for the environmental lens (not including the parent variable Power Supply Volume).

Interestingly this diagram shows that the lens score is more sensitive to some of the assessment modules than some of the modules representing LCD components. There is a reason why the lens is so much more sensitive to LCA-Connection than the rest. For this DOME model, LCA-Connection is a catalog that contains modules that encapsulate Team®. The catalog contains three different modules. The first module is called Nothing and performs no environmental assessment. The other catalog choices utilize two different impact assessments, one is fast and the other is detailed. Therefore the Environmental lens is very sensitive to this catalog because it is changing from no assessment to some assessment.

On further inspection, it may be interesting to notice that there are three variables that do not seem to affect the lens score at all. These are the Eng. Rel. Min Time to Failure[hrs], Main PCB Power Consumption[W], and Video PCB Power Consumption[W]. This information could be very valuable to the user.

Just like the causality/dependency case the user should be offered some alternative ways to view this information. As an example, two more tornado diagram were produced. One includes just the catalogs and the other includes only the variables. One important point to mention is that for this proposed adaptation of sensitivity analysis, the catalogs will always have a positive slope. This is because the data is always presented in ascending order. Therefore the user only sees the magnitude of sensitivity with regard to catalogs. Since the data corresponding to the variables is not sorted, the best-fit line could have a positive or negative slope. Therefore the designer can discover if the parent is positively affecting or negatively affecting the lens score.

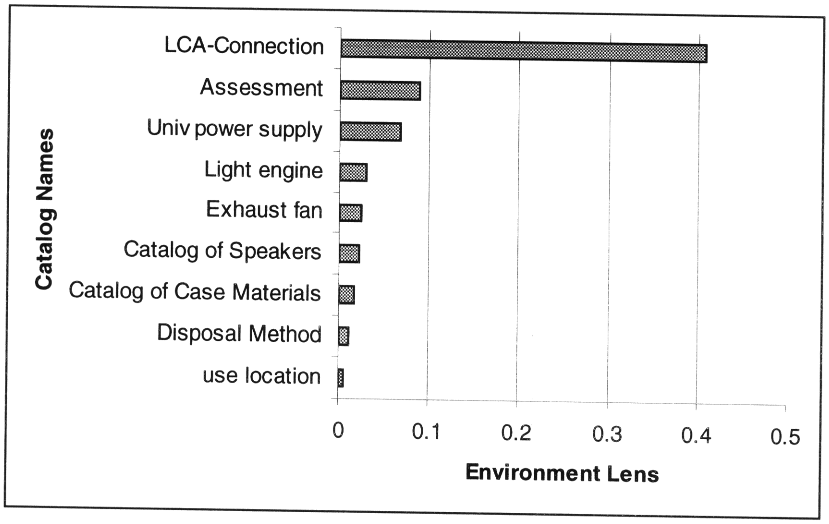


Figure 6.4 Sensitivity analysis of the environmental lens (Variables from catalogs only).

Figure 6.4 shows just the catalogs items from the previous sensitivity analysis. In the same way, Figure 6.5 illustrate the sensitivity of variables not from catalogs. It is interesting to see that Main PCB weight and Video PCB weight both effect the lens score in a negative way. Finally, as mentioned before, the last three variables have a negligible effect on the lens score.

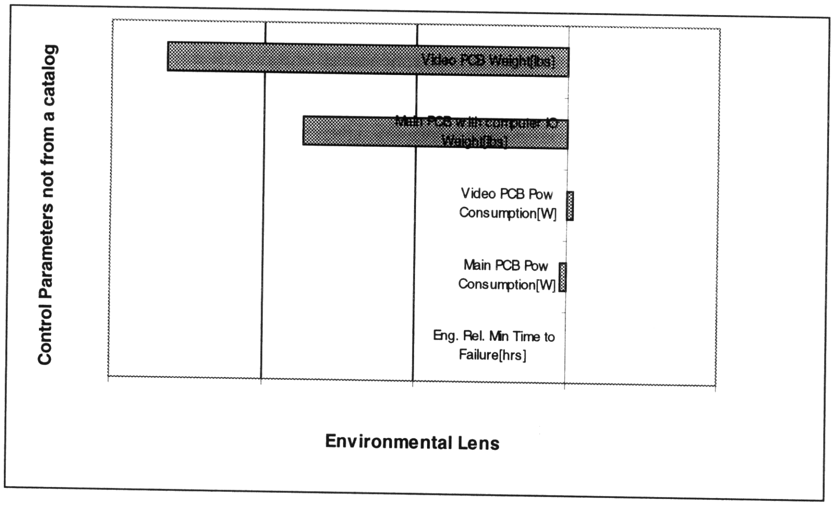


Figure 6.5 Sensitivity analysis of the environmental lens (Variables not from catalogs).

All in all, the designer now has some more insight into the model. Future work will include a full implementation of this feature in to the DOME framework.

7

Causal Ordering Given Only Equations

The prior sections outline methods to extract causality/dependency information from a DOME design model. These methods generated causality/dependency graphs given a local DOME model or a DOME model with remote connections. There is a third possibility, which is to be able to determine causality based solely on a set of mathematical equations.

In the beginning of a design process, when the engineer is first trying to model something, they usually start with a set of mathematical equations. Alternatively, the designer might utilize a set of equations somebody else developed. In the future, a DOME user may have the ability to build a module containing only mathematical equations and not specify any dependencies. For this case, DOME software would have to be able to manipulate the equations in such a way so that the module can be linked to other modules in the design when needed.

For these cases it would be convenient if there were an algorithm that could produce a causal ordering based on a set of equations. An algorithm such as this could really aid the designer because they may mistakenly assign the wrong causal ordering to a given set of equations. This chapter first presents an example of why this tool might be beneficial and then concludes with how it can be useful in the context of DOME.

7.1 The Bathtub Example

The following is a bathtub example that was taken from [3] and illustrates the usefulness of being able to establish causality knowing only a set of equations.

To begin, an imaginary designer has the desire to model a bathtub to ascertain the optimal configuration. First the designer thinks about how to model this object. To start, the tub needs to be a certain size and include some kind of drain. If the designer were considering a bathtub with no stopper, they would have to think about flow rates and drain diameters.

The designer now has to relate these facts together to form some sort of model of the bathtub. The following is a picture of this system.

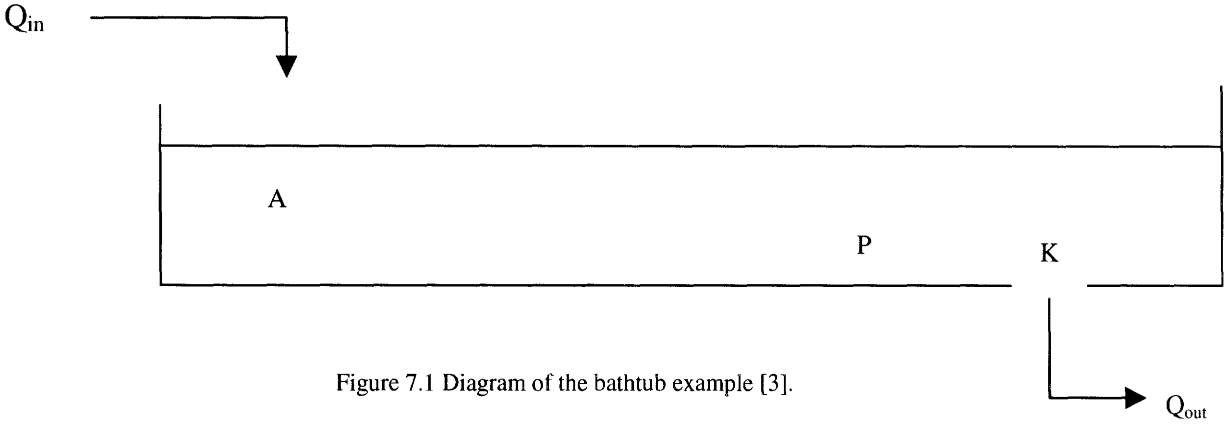


Figure 7.1 Diagram of the bathtub example [3].

Next, the designer lists the variables and relates them in the form of equations. Q_{in} and Q_{out}, represent input and output flow rates. Variable A will represent the volume of water in the tub and K gives the size of the drain opening. The designer makes one more relation by realizing that the pressure at the bottom of the tub relates to the out going flow rate. Now the designer uses these variables in equations. First they set Q_{in} as a constant. The equation for this relation is as follows:

$Q_{in} = c_1$, c_1 is a constant.

Next, because the designer wants to keep the water level steady, they set the inflow rate equal to the outflow rate:

$Q_{out} = Q_{in}$.

Then the designer sets the size of the drain:

$K= c_2$, c_2 is a constant.

The next equation describes the output flow rate as being proportional to the pressure.

$Q_{out} = c_3KP^{1/2}$, c_3 is a constant.

The last equation describes the pressure as being proportional to the depth of water

$A= c_4P$, c_4 is a constant.

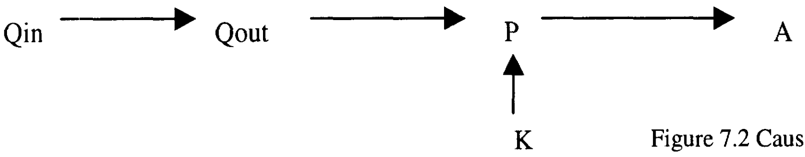


Figure 7.2 Causal ordering for the bathtub example. [3]

The resulting causal ordering, shown in Figure 7.2, obtained by using the causal ordering methodology presented in [23] is a bit counter intuitive. It shows that the output flow rate directly depends on the input flow rate, the pressure depends on the output flow rate and drain size, and the water volume depends on the pressure.

One would think that adding water to the tub would increase the volume A, which would increase the pressure P, which would then increase the output Q_{out}, like in Figure 7.3.

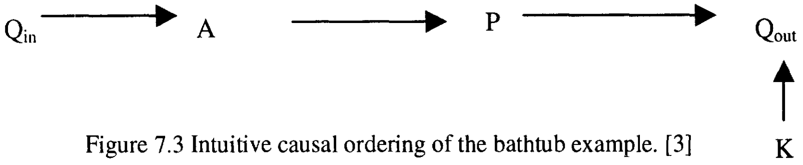


Figure 7.3 Intuitive causal ordering of the bathtub example. [3]

Therefore a designer might not guess the correct causal ordering as shown in Figure 7.2. To confirm that figure 7.2 is the correct representation, the reader must recognize that this system is in equilibrium. These equations do not depict a transient system.

To explain, we begin by abruptly opening the drain while the system is at equilibrium. The direct response will be that Q_{out} will increase. However when equilibrium is eventually restored, Q_{out} must be equal to Q_{in}, otherwise the system would not be in equilibrium.

Therefore changing the size of the drain only affects the equilibrium values of the pressure P and the volume of water A but not Q_{out}. Consequently, the equilibrium value of Q_{out} cannot be dependent on the pressure or the volume of water. This conclusion is accurately portrayed in figure 7.2 but not in figure 7.3.

This type of methodology could be useful to the designer to help them understand the causal ordering in a given set of equations.

7.2 Implementation of Causal Ordering

A computer program was develop to carry out the algorithm in [23] and is found in Appendix A. The program parses all the equations available like the ones presented in the previous section and, using Simon's algorithm, returns a causal ordering.

Taking the equations from above for example, the input file would look like this:
$Q_{in} = c_1$

$Q_{out} = Q_{in}$
$K=c_2$
$Q_{out} = c_3KP^{1/2}$
$A= c_4P$

The program produces the following output (the arrow indicate causality):

Q_{in} -> Q_{out}
Q_{out} -> P
P -> A
K -> P

Two more interesting examples are given to demonstrate the capability of this methodology implemented as a computer program.

The input equations are:

$y_1 + 12y_2 + 13y_3 + 18z_8 + 19z_9 +10 = 0$
$y_1 + 22y_2 + 24y_4 +28z_8 + 20 = 0$
$y_3 + 37z_7 + 39z_9 + 30 = 0$
$y_4 + 45y_5 + 46z_6 + 48z_8 + 40 = 0$
$y_2 + 55y_5 + 58z_8 + 50 = 0$
$z_6 + 60 = 0$
$z_7 + 70 = 0$
$z_8 + 80 = 0$
$z_9 + 90 = 0$

The causal ordering output is:

z_7 -> y_3
z_9 -> y_3
y_3 -> y_1,y_2,y_4,y_5
z_6 -> y_1,y_2,y_4,y_5
z_8 -> y_1,y_2,y_4,y_5
z_9 -> y_1,y_2,y_4,y_5

The input equations are:

$y - 2 = 2(x - 1)$,
$2y = 4$,
Sin [5z] – y * x = -2

Causal ordering:
y -> x
x -> z
y -> z

7.3 How this Could be Useful in DOME

This methodology is somewhat limited because it only works on a set of self-contained (N equations and N unknowns) set of equations. There are a few cases in which a complex model is made up of such a set of equations. There are many instances in which a module contains a computer program that uses if then statements. In such cases this approach would not work. An addition problem is that if some DOME objects are remote, one may not be able to access the set of equations and therefore not know the causality.

However, this method could be helpful to a designer in the initial stages of design. Also, in subsequent versions of DOME, the user might be able to enter only a list of equations in a module. Being able to determine causality, the application could know how to connect such a module by knowing what the inputs and outputs are of a given set of equations.

8

Summary/Conclusions

8.1 Summary

This thesis presented methods on how to obtain qualitative information from a DOME design model with the objective of attaining several goals. The first was to provide the DOME user with tools or features to help them understand how and why results were reached and not merely what they are. The second was to give the designer an alternative point of view of a model to help them gain valuable insights. Third, these tools were offered to help reduce the amount of time needed for a first time user to understand and navigate through a large distributed design problem. The fourth goal was to offer features to assist in validating and diagnosing a complicated distributed model. In addition to these goals, it was also determined that the algorithms developed to extract this information would have to be robust enough to handle different types of DOME models such as a completely local DOME model or one that is linked to other software applications.

The thesis began by first furnishing a brief description of the DOME framework. In this chapter, the reader learns about the creators' vision for DOME and of the objected oriented approach to design. From the discussion, an explanation of how a DOME model could be linked to other applications was offered. Next, an example of a complex distributed DOME design problem of an LCD computer projector was presented. Finally, it was shown through an anecdotal experience why these features are needed and how they could be employed as a diagnostic tool.

In the next chapter, some research was conducted into related fields of cause and effect, and sensitivity analysis. The authors of the papers surveyed argued that being able to extract qualitative information from a physical model is extremely important and offered their approaches. Later in this chapter, some current work in the field of visualization was also reviewed. As a result of this search, a Java graph layout algorithm was discovered and subsequently utilized to view the output of the causality algorithm. The next sections examined some other promising visualization software applications that could possibly be incorporated into the DOME framework. Finally, the chapter concluded with a discussion of clustering.

Chapter 4 began by showing the causality features of two applications, Excel® and Solid Works®, and then showed how limited they were. The next passage described how DOME displays causality and how the dependency is established by the way the designer writes the Modeling Definition Language. The algorithm for the first scenario of extracting the causality directly from the instantiated design object was presented next. After this, some use scenarios were outlined and a suggestion for the interface was made. The chapter concluded by discussing the limitation of using such an algorithm for a DOME model that was linked to other applications. This is because in such cases it is not known if a parent variable, linked to a remote node/module, is in fact influencing its children.

As a result of the limitations listed in Chapter 4, the next chapter outlined a new method to discover if a parent variable truly influences a child variable. The process was referred to as the "perturbation method." In summary, this method uses the idea that if a parent variable is changed and there is a resulting change in a child variable, then there is causality. Using the LCD projector as a test case, it was shown that there was a big difference between the causality graphs that were produced using this perturbation method compared to those generated in Chapter 4. Knowing the true causality is obviously crucial for the designer. One important discovery made during the implementation of this routine was that some variables' distributions changed when the model was updated even if there was no change in an upstream variable. This was because some of these distributions were generated by a parametric Monte Carlo simulation. Because of the noise present, it was important to include some type of tolerance to be sure that a change in a parent variable really caused a change in a dependent variable.

After creating a function that determines if a dependent variable is sensitive at all to a change in a parent variable, it was not too difficult to extend these capabilities to perform a sensitivity analysis. Since sensitivity analysis is another method of extracting qualitative information from a DOME design model, the function was enhance to provided such a service. Chapter 6 made some proposals on how to perform sensitivity analysis and also presented an interesting example of a sensitivity analysis carried out on the environmental lens score.

Finally, Chapter 7 explained how to determine causality from a set of equations that described a physical phenomenon. An illustration was presented to show how someone could be misled to formulate the wrong causality given an example set of equations. The chapter also suggested how this functionality could someday be adopted into the DOME framework.

8.2 Conclusions

In conclusion, it was demonstrated that these qualitative features are very important options to include in a complex distributed model such as those produced using DOME. In Chapter 2, it was shown how a sensitivity analysis quickly uncovered the fact that the Solid Works model returned bogus information each time the model was changed. By using the perturbation method in Chapter 5, it was also shown that fewer variables were dependent on the selected parent variable than previously thought. Likewise, the sensitivity analysis example presented in Chapter 6 also shows how the environmental engineer might gain valuable insights into what affects the environmental score the most.

Another finding was that care must be taken when using the perturbation method. In some cases, there might be some noise in a calculated distribution and therefore it might look like it is changing when in fact it is not. The size of the tolerance used to detect a change, and the number of simulations utilized to generate distributions might have to be change from model to model.

In the future, DOME design problems could grow larger and larger and be linked to dozens of software applications. If an automobile design was selected, for example, the model could be tremendous. Many of the product development participants will need these features to help them understand the model. First time users will also need to quickly gain a sense of what is connected to what or to perform diagnostics.

8.3 Future Work

One of the goals of this thesis was to make the algorithm flexible enough to determine if, while generating a causality chain, it encountered a linked software application. When the algorithm senses a remote node/module it initiates the perturbation method. It may be desirable to use this method in every case. This is because formulas may not ever really make use of a parent variable and it may be important for a designer to know this. The trade off here is that in one case the designer will be certain of causality for every type of DOME model. One the other hand, this approach could be very time consuming and the chance that a designer might link to a parent variable and not use it in an equation is low.

Other future work could include improving the algorithms to make them more efficient. One reason for this is because it took a relatively long time to generate the results of the sensitivity analysis presented in Chapter 6. Also some of the visualization techniques presented in Chapter 3 could be utilized to view output of the causality analysis. As DOME models grow, the problem becomes understanding, visualizing, navigating, and getting the information one wants when one needs it. In the same way, tailoring this output to suit the individual using the application could also be a worthwhile task.

Lastly, there are currently two versions of DOME. One GUI is written in motif and the other was created using Java. The next step would be to fully integrate these features in the Java version of DOME.

References

[1] Robertson G.G., J.D. Mackinlay, and S.K. Card, Cone Trees: Animated 3D Visualizations of Hierarchical Information, Proceedings of HHI '91, p. 189-194.

[2] S. Mukherjea, J. Foley, S. Hudson, Visualizing Complex Hypermedia Networks Through Multiple Hierarchical Views. Graphics, visualization & usability center, College of computing, Georgia Institute of Technology, CHI' 95 Mosaic of Creativity.

[3] Causality and Model Abstraction, Yumi Iwasaki Herbert A. Simon, Corporate Source Stanford University Palo Alto CA USA, Artificial Intelligence v 67 n 1 May 1994 p 143- 194 ISSN: 0004-3702 CODEN: AINTBB Publication year 1994.

[4] Borland, N., "Building a design using DOME: MDL Manual," in http://cadlab.mit.edu/~nborland/documentation/home.html, 1997.

[5] Toward Generating Causal Explanation: Qualitative Simulation with Association Mechanism to Quantitative Information Masanori Akiyoshi Shogo Nishida.

[6] Causal Ordering and Identifiableability In the book Models of Discovery and other topics in the methods of Science Herbert A. Simon Carnegie-Mellon University Pittsburgh Penn. USA Carnegie-Mellon University Pittsburgh Penn. USA.

[7] Causal Ordering in a Mixed Structure Yumi Iwasaki Department of Computer Science Carnegie Mellon University Pittsburgh, Pennsylvania 15213.

[8] Pahng, K. F., Senin, N., and Wallace, D. R., 1998, "Distributed modeling and evaluation of product design problems," to appear in Computer Aided Design.

[9] Iwasaki, Y., and Simon, H. A. Causality in Device Behavior. *Artificial Intelligence* 29 1986.

[10] de Kleer, J. and Brown, J. S., A Qualitative Physics Based on Confluences, *Artificial Intelligence* 24 (1984) 7-83.

[11] Pahng, F., "Modeling and Evaluation of Design Problems in a Network-Centric Environment," MIT, Feb, 1998.

[12] de Kleer, J. and Brown, J. S., Theories of Causal Ordering, *Artificial Intelligence* **29** (1986) 33-61.

[13] http://www.palisade.com/html/ptree.html.

[14] Senin, N., Borland, N. and Wallace, D. R., 1997, "Distributed modeling of product design problems in a collaborative design environment", CIRP International Design Seminar Proceedings: Multimedia Technologies for Collaborative Design and Manufacturing, October 8-10, 1997, Los Angeles, California.

[15] Rao, R., J.O. Pedersen. M. A. Hearst, J. D. Mackinlay, S. K. Card, L. Masinter, P. Halvorsen, and G.G. Robertson. Rich Interaction in the Digital Library, Communications of the ACM, April 1995/Vol 38, No. 4.

[16] Kim, J. B. and Wallace, D. R., 1997, "A Goal-oriented Design Evaluation Model," Proceedings of ASME Design Engineering Technical Conference, DETC97/DTM-3878.

[17] http://www.javasoft.com/applets/js-applets.html.

[18] Siegel, A. K., *CORBA*: *Fundamentals and Programming*: OMG, 1996.

[19] K. Utting and N. Yankelovich. Context and Orientation in Hypermedia Networks. ACM Transactions on Office Information Systems, 7(1):58-84, 1989.

[20] C. Neuwirth, D. Kauffer, R. Chimera, and G. Terilyn. The Notes Program: A Hypertext Application for Writing from Source Texts. In Proceedings of Hypertext '87 Conference, pages 121-135, Chapel Hill, NC, November 1987.

[21] Kleiber, M., Antunez, H., T. D. Hien, P. Kowalczyk. Parameter Sensitivity in Nonlinear Mechanics, Theory and Finite Element Computations John Wiley & Sons 1997.

[22] Rubinstein, R. Y., A. Shapiro, Discrete Event System : sensitivity analysis and stochastic optimization by the score function method. Chichester, New York, Wiley, C1993.

[23] Simon, H. A., On the definitions of causal relation, *J. Philos.* **49** (1952) 517-528.

[24] Wasserlein, H., "A Representations for Optimizing Scheduling Problems", Undergraduate thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1997.

[25] Wang, P. H., "Benchmarking a Collaborative Concurrent Computer Design Tool" Masters thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1998.

[26] Yang, J.A., Intellectual Genealogies: Visualizing Histories in Citation Graphs, MIT, Department of Mechanical Engineering, March 1997.

[27] Odoni A., Apostolakis G., Engineering Risk-Benefit Analysis ("ERBA"), MIT school of Engineering, class lecture notes, Feb 5 1997.

[28] Lamping J., Roa R., and P. Pirolli. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. Xerox Palo Alto Research Center 3333 Coyote Hill Road, Palo Alto, CA 94304.

[29] Borland, N., Integrating Environmental Impact Assessment into Product Design, Masters Thesis, Massachusetts Institute of Technology, June 1998.

[30] Gruninger, T. and D. Wallace, "Multimodal Optimization using Genetic Algorithms," MIT CADlab 1996.

[31] S. Mukherjea, J. Foley, and S. Hudson. Interactive Clustering for Navigating in Hypermedia Systems. *In Proceedings of the ACM European Conference of Hypermedia Technology*, pages 136-144, Edinburgh, Scotland, September 1994.

Appendix A

Casualty functions

```
#include <iostream.h>
#include "FileScan.h"
#include "GAO.h"

short _DebugOn = 0;

void findCausalityChain(GNode *);
void findRoots(GNode *);

void findDependencyChain(GNode *);
void findDependencyChainCrit(GNode *);
void findDependencyChainLens(GNode *);

void findLeaves(GNode *);
void findLeavesCrit(GNode *);
void findLeavesLens(GNode *);

VarNode *  origCurrentVarNode = NULL;
VarNode *  savedVarNode = NULL;

GNode * currentGNode;
GNode * origCurrentGNode;

Design * D = NULL;

ofstream outputF("MaterialsCost.html");

void main()
{
 char filename[200];

 cerr<<"Input file name:";
 cin>>filename;

 ReadFile a(filename, XXXTrue);
 a.Scan();

 D = a.newGAO()->design();

 Module * currentMod = NULL;

 VarNode *  currentVarNode = NULL;

 currentMod = D->designModule("Geometry");

 currentGNode = & currentMod->varNode("Power Supply Volume");
 origCurrentGNode =  &currentMod->varNode("Power Supply Volume");

 findDependencyChain(currentGNode);
 findDependencyChainCrit(currentGNode);

 findDependencyChainLens(currentGNode);


 findLeaves(currentGNode);

 findLeavesCrit(currentGNode);

 findLeavesLens(currentGNode);


 currentMod = D->designModule("Cost");
 currentGNode = & currentMod->varNode("Materials Cost");
 origCurrentGNode =  &currentMod->varNode("Materials Cost");


 findCausalityChain(currentGNode);

 findRoots(currentGNode);


 currentMod = D->designModule("Assessment");
 currentGNode = & currentMod->lens("Environment");
 origCurrentGNode =  &currentMod->lens("Environment");

 findCausalityChain(currentGNode);

 findRoots(currentGNode);


 currentMod = D->designModule("Assessment");
 currentGNode = & currentMod->criterion("EPS-Air Crit");
 origCurrentGNode =  &currentMod->criterion("EPS-Air Crit");

 findCausalityChain(currentGNode);

 findRoots(currentGNode);
}

void findCausalityChain(GNode * currentGNode)
{
 int i;
 int nParents = 0;

 Array<GNode *> GNodeParentArray;

 cerr << "The class type is"<< currentGNode->classType() << endl;

 nParents = currentGNode->nparents();

 if ( nParents == 0)
   {
    return;
   }
 GNodeParentArray =  currentGNode->parents();

 for(i = 0; i < nParents; i++){

  outputF <<GNodeParentArray[i]->name()<< "‘"<<
        D->connectedDesignModuleName(((VarNode * )GNodeParentArray[i]))
        <<"?"
        <<currentGNode->name()<< "‘" <<D->connectedDesignModuleName((VarNode *)currentGNode)<<",";
  findCausalityChain(GNodeParentArray[i]);
 }
}

void findRoots(GNode * currentGNode)
{

 int i;
 int nParents = 0;

 Array<GNode *> GNodeParentArray;

 nParents = currentGNode->nparents();

 if ( nParents == 0)
  {
   outputF <<currentGNode->name()<<"‘"<<D->connectedDesignModuleName((VarNode * )currentGNode)
   <<"?"
        <<origCurrentGNode->name()<<"‘"<<
        D->connectedDesignModuleName((VarNode * )origCurrentGNode)<<",";

         return;
  }
 GNodeParentArray = currentGNode->parents();
 for(i = 0; i < nParents; i++)
  {
   findRoots(GNodeParentArray[i]);
  }
}

void findDependencyChain(GNode * currentGNode)
{

 int i;
 int nChildren = 0;

 Array<GNode *> GNodeChildArray;

 nChildren = currentGNode->nchildren();

 if ( nChildren == 0)
  {
   return;
  }

 GNodeChildArray = currentGNode->children();
 for(i = 0; i < nChildren; i++){

  if ( currentGNode->classType() !=  22 && GNodeChildArray[i]->classType() !=  22 )
   {
        outputF<<currentGNode->name()<< "‘" <<D->connectedDesignModuleName((VarNode *)currentGNode)
         <<"?"<< GNodeChildArray[i]->name()<< "‘"
          << D->connectedDesignModuleName(((VarNode * )GNodeChildArray[i]))<<",";
   }
  findDependencyChain(GNodeChildArray[i]);
 }
}

void findDependencyChainCrit(GNode * currentGNode)
{

 int i;
 int nChildren = 0;

 Array<GNode *> GNodeChildArray;

 nChildren = currentGNode->nchildren();

 if ( nChildren == 0 )
  {
   return;
  }

 GNodeChildArray = currentGNode->children();

 for(i = 0; i < nChildren; i++){

  if (GNodeChildArray[i]->classType() != 13)
   {
        outputF<<currentGNode->name()<< "‘" <<D->connectedDesignModuleName((VarNode *)currentGNode)
         <<"?"<< GNodeChildArray[i]->name()<< "‘"
          << D->connectedDesignModuleName(((VarNode * )GNodeChildArray[i]))<<",";
   }

  findDependencyChainCrit(GNodeChildArray[i]);
 }
}

void findDependencyChainLens(GNode * currentGNode)
{

 int i;
 int nChildren = 0;

 Array<GNode *> GNodeChildArray;

 nChildren = currentGNode->nchildren();

 if ( nChildren == 0)
  {
   return;
  }
 GNodeChildArray = currentGNode->children();

 for(i = 0; i < nChildren; i++)
  {
   outputF<<currentGNode->name()<< "‘" <<D->connectedDesignModuleName((VarNode *)currentGNode)
        <<"?"<< GNodeChildArray[i]->name()<< "‘"
          << D->connectedDesignModuleName(((VarNode * )GNodeChildArray[i]))<<",";

   findDependencyChainLens(GNodeChildArray[i]);
  }
}

void findLeaves(GNode * currentGNode)
{

 int i;
 int nChildren = 0;

 Array<GNode *> GNodeChildArray;

 nChildren = currentGNode->nchildren();

 if ( nChildren == 0  && currentGNode->classType() != 22 && currentGNode->classType() != 23 &&
currentGNode->classType() != 13)
  {

   outputF<<origCurrentGNode->name()<< "‘" <<
   D->connectedDesignModuleName((VarNode *)origCurrentGNode)
    <<"?"<< currentGNode->name()<< "‘"
    << D->connectedDesignModuleName((VarNode *)currentGNode)<<",";
   return;
  }
 GNodeChildArray = currentGNode->children();

 for(i = 0; i < nChildren; i++){

  if (GNodeChildArray[i]->classType() == 22 && nChildren == 1 )
   {
        outputF <<origCurrentGNode->name()<< "‘" <<
        D->connectedDesignModuleName((VarNode *)origCurrentGNode)
          <<"?"<< currentGNode->name()<< "‘"
            << D->connectedDesignModuleName((VarNode *)currentGNode)<<",";
   }
  findLeaves(((VarNode * )GNodeChildArray[i]));
 }

}

void findLeavesCrit(GNode * currentGNode)
{

 int i;
 int nChildren = 0;

 Array<GNode *> GNodeChildArray;
 nChildren = currentGNode->nchildren();

 if ( nChildren == 0 && currentGNode->classType() == 22)
   {

    outputF<<origCurrentGNode->name()<< "‘" <<
    D->connectedDesignModuleName((VarNode *)origCurrentGNode)
            <<"?"<< currentGNode->name()<< "‘"
              << D->connectedDesignModuleName((VarNode *)currentGNode)<<",";
         return;
   }
 GNodeChildArray = currentGNode->children();

 for(i = 0; i < nChildren; i++){

   if (GNodeChildArray[i]->classType() == 22)
     {
            outputF <<origCurrentGNode->name()<< "‘" <<
            D->connectedDesignModuleName((VarNode *)origCurrentGNode)
              <<"?"<< GNodeChildArray[i]->name()<< "‘"
                << D->connectedDesignModuleName(((VarNode *)GNodeChildArray[i]))<<",";
     }
   findLeavesCrit(GNodeChildArray[i]);
 }

}

void findLeavesLens(GNode * currentGNode)
{

 int i;
 int nChildren = 0;

 Array<GNode *> GNodeChildArray;

 nChildren = currentGNode->nchildren();

 if ( nChildren == 0 && currentGNode->classType() ==  13)
   {
    outputF<<origCurrentGNode->name()<< "‘" <<
    D->connectedDesignModuleName((VarNode *)origCurrentGNode)
      <<"?"<< currentGNode->name()<<"‘"
              << D->connectedDesignModuleName((VarNode *)currentGNode)<<",";
    return;
   }
 GNodeChildArray = currentGNode->children();

 for(i = 0; i < nChildren; i++)
   {
    findLeavesLens(GNodeChildArray[i]);
   }

}
```

Perturbation functions

```
#include <iostream.h>
#include "FileScan.h"
#include "GAO.h"

short _DebugOn = 0;

void findDependencyChain(GNode *);
void findRoots(GNode * );

void findCausalityChain(GNode *);

GNode *  origCurrentGNode = NULL;
Catalog * currentCat = NULL;

double oldNodeVal = 0.0;

GNode *  storedNodesFromFindRoots[20];

int count = 0;

Design * D = NULL;

ofstream outputF("blackBox.html");

void main()
{

  for (int h = 0; h < 20; h++)
    {
     storedNodesFromFindRoots[h] = NULL;
    }

  char filename[200];
  int nModules, nParents;

  cerr<<"Input file name:";
  cin>>filename;

  ReadFile a(filename, XXXTrue);
  a.Scan();

  D = a.newGAO()->design();

  nModules = D->nDesignModules();

  Module * currentMod = NULL;

  GNode *  currentGNode;

  currentMod = D->designModule("Geometry");
  currentGNode = & currentMod->varNode("Power Supply Volume");
  origCurrentGNode = & currentMod->varNode("Power Supply Volume");
```

```
 oldNodeVal = ((VarNode *)origCurrentGNode)->position();
 cerr << "The value of " << origCurrentGNode->name() << " is: " << oldNodeVal << " end" << endl;

 findCausalityChain(currentGNode);
 }

void findCausalityChain(GNode * currentGNode)
{
 int i;
 int nChildren = 0;

 Array<GNode *> GNodeChildArray;

 nChildren = currentGNode->nchildren();

 if ( nChildren == 0)
  {
   return;
  }

GNodeChildArray = currentGNode->children();

 for(i = 0; i < nChildren; i++){

  if ( currentGNode->classType() !=  22 && GNodeChildArray[i]->classType() !=  22 )
   {
        cerr <<currentGNode->name()<< "‘" <<D->connectedDesignModuleName((VarNode *)currentGNode)
         <<"?"<< GNodeChildArray[i]->name()<< "‘"
          << D->connectedDesignModuleName(((VarNode * )GNodeChildArray[i]))<<",";
   }

  findCausalityChain(GNodeChildArray[i]);
 }
}

void findDependencyChain(GNode * currentGNode)
{
 XXXString DesignModuleName;
 XXXString NodeName;

 GNode * tempGnodePtr = NULL;

 int i;
 int nParents = 0;
 int nRecords = 0;

 Array<GNode *> GNodeParentArray;

 nParents = currentGNode->nparents();

 if ( nParents == 0)
  {
   cerr << "I have node " << currentGNode->name() << " from module " <<
        D->connectedDesignModuleName((VarNode * )currentGNode)<< endl;
   return;
  }
  GNodeParentArray = currentGNode->parents();

  cerr << "The current node " << ((VarNode *)currentGNode)->name() << " from design module "
   << D->connectedDesignModuleName((VarNode * )currentGNode)<< ", is  " <<
    ((VarNode *)currentGNode)->documentation().url()<< endl;

  if (((VarNode *)currentGNode)->documentation().url()  == "Remote")
   {

    findRoots(currentGNode);

    for(int j = 0; j < 14; j++)
         {

           DesignModuleName = D->connectedDesignModuleName(((VarNode * )storedNodesFromFindRoots[j]));
           NodeName = storedNodesFromFindRoots[j]->name();

         if (!D->designModuleIsStatic(D->connectedDesignModuleName(((VarNode *
                 )storedNodesFromFindRoots[j]))))
           {
            currentCat = D->catalogAssociatedToDesignModule(D->connectedDesignModuleName(((VarNode *
                 )storedNodesFromFindRoots[j])));

            nRecords =  currentCat->nRecords();

            for(int n = 0; n < nRecords; n++)
                 {
                  D->replace(DesignModuleName, &(currentCat->record(n)));

                  D->evaluate();

                  if (oldNodeVal != ((VarNode *)origCurrentGNode)->position())
                    {
                     n = nRecords;
                    }
                  oldNodeVal = ((VarNode *)origCurrentGNode)->position();
                 }
           }
          else
           {
            ((VarNode * )storedNodesFromFindRoots[j])->moveTo(((VarNode * )storedNodesFromFindRoots[j])-
                 >position()*.1 +
                                                              ((VarNode *
                 )storedNodesFromFindRoots[j])->position());
           }
         }
   return;
  }

 for(int k = 0; k < nParents; k++){
  cerr <<GNodeParentArray[k]->name()<< "‘"<< D->connectedDesignModuleName(((VarNode *
)GNodeParentArray[k]))
   <<"?"
        <<currentGNode->name()<< "‘" <<D->connectedDesignModuleName((VarNode *
)currentGNode)<<","<< endl;
  findDependencyChain(GNodeParentArray[k]);
 }

}

void findRoots(GNode * currentGNode)
{
 int i;
 int nParents = 0;

 Array<GNode *> GNodeParentArray;

 nParents = currentGNode->nparents();

 if ( nParents == 0)
  {
   storedNodesFromFindRoots[count] = currentGNode;
   count++;
   return;
  }
 GNodeParentArray = currentGNode->parents();
 for(i = 0; i < nParents; i++)
  {
   findRoots(GNodeParentArray[i]);
  }
}
```

Sensitivity analysis

```
#include <iostream.h>
#include "FileScan.h"
#include "GAO.h"

short _DebugOn = 0;

void findDependencyChain(GNode *);
void findRoots(GNode * );
GNode *  origCurrentGNode = NULL;
Catalog * currentCat = NULL;

double oldNodeVal = 0.0;

const int numberOfStoredRoots = 500;
const int numberOfSavedModNames = 500;

GNode *  storedNodesFromFindRoots[numberOfStoredRoots];

XXXString savedModuleName[numberOfSavedModNames];
XXXLens * origCurrentLens = NULL;

Module * currentMod = NULL;

int Scount = 0;
int count = 0;

Design * D = NULL;

ofstream outputF("FinalSens100.html");
ofstream outputG("SENSRESULTS100.HTML");

void main()
{
 for (int h = 0; h < numberOfStoredRoots ; h++)
   {
    storedNodesFromFindRoots[h] = NULL;
   }

 for (int o = 0; o < numberOfSavedModNames; o++)
   {
    savedModuleName[o] = "x";
   }

 char filename[200];
 int nModules, nParents;

 cerr<<"Input file name:";
 cin>>filename;

 ReadFile a(filename, XXXTrue);
 a.Scan();

 D = a.newGAO()->design();
 nModules = D->nDesignModules();

 GNode *  currentGNode;

 currentMod = D->designModule("Assessment");
 currentGNode = & currentMod->lens("Environment");
 origCurrentLens =  &currentMod->lens("Environment");

 oldNodeVal = origCurrentLens->evaluate();
 cerr << "The value of " << origCurrentLens->name() << "  is: " << oldNodeVal << " end" << endl;
 outputG << "oldNodeVal is " << oldNodeVal<< endl;
 outputG << endl;

 findDependencyChain(currentGNode);
}

void findDependencyChain(GNode * currentGNode)
{
 XXXString DesignModuleName;
 XXXString NodeName;

 GNode * tempGnodePtr = NULL;

 int i;
 int nParents = 0;
 int nRecords = 0;

 Array<GNode *> GNodeParentArray;

 nParents = currentGNode->nparents();

 GNodeParentArray = currentGNode->parents();

 findRoots(currentGNode);

 for (int s = 0; s < numberOfStoredRoots; s++)
   {
    if (storedNodesFromFindRoots[s] != NULL)
           {
            outputF <<"The node is: "<< storedNodesFromFindRoots[s]->name()<<" from module "<<
              D->connectedDesignModuleName(((VarNode * )storedNodesFromFindRoots[s]))<< endl;
           }
   }

 for(int j = 12; j < 15; j++)
   {
    if (storedNodesFromFindRoots[j] != NULL )
           {
            outputF<<"Root number "<< j <<" is node "<< storedNodesFromFindRoots[j]->name()
              <<" from design module "<<D->connectedDesignModuleName(((VarNode *
                    )storedNodesFromFindRoots[j]))<< endl;

            DesignModuleName = D->connectedDesignModuleName(((VarNode * )storedNodesFromFindRoots[j]));
            NodeName = storedNodesFromFindRoots[j]->name();
```

```
        if (!D->designModuleIsStatic(D->connectedDesignModuleName(((VarNode *
                )storedNodesFromFindRoots[j]))))
         {

          currentCat = D->catalogAssociatedToDesignModule(D->connectedDesignModuleName(((VarNode *
                )storedNodesFromFindRoots[j])));

          nRecords =  currentCat->nRecords();

          int defualtRecordIndex =  currentCat->recordIndex(D-
                >recordAssociatedToDesignModule(DesignModuleName));

          for(int n = 0; n < nRecords; n++)
                {
                 if (!(DesignModuleName == "LCA-Connection" && n == 2))
                   {

                     D->replace(DesignModuleName, &(currentCat->record(n)));

                     D->evaluate();

                     currentMod = D->designModule("Assessment");

                     origCurrentLens =  &currentMod->lens("Environment");

                          <<", Current Record is "<<D->recordAssociatedToDesignModule(DesignModuleName)-
                                  >name()
                                          << "and the current record is " << n << endl;
                   }
                }

          D->replace(DesignModuleName, &(currentCat->record(defualtRecordIndex)));

          currentMod = D->designModule("Assessment");
          origCurrentLens =  &currentMod->lens("Environment");

         }
        else
         {

          double holdNodeVal = ((VarNode * )storedNodesFromFindRoots[j])->position();

          for (int y =0; y < 5; y++)
                {
                 ((VarNode * )storedNodesFromFindRoots[j])->moveTo(holdNodeVal*(.04*y) + holdNodeVal);
                }

          for (int x =0; x < 5; x++)
                {
                 ((VarNode * )storedNodesFromFindRoots[j])->moveTo(holdNodeVal - holdNodeVal*(.04*x));
                }
         }
        }
    }
  }
```

```
void findRoots(GNode * currentGNode)
{

 int i;
 int nParents = 0;

 int check = 0;

 Array<GNode *> GNodeParentArray;

 nParents = currentGNode->nparents();

 if ( nParents == 0)
  {
   for (int m = 0; m < numberOfSavedModNames; m ++ )
        {
         if (savedModuleName[m] != "x")
          {
           outputF << "savedModuleName[m] is " << savedModuleName[m] << endl;
          }

         if (savedModuleName[m] ==  D->connectedDesignModuleName((VarNode * )currentGNode) &&
           !D->designModuleIsStatic(D->connectedDesignModuleName(((VarNode * )currentGNode))))
          {
           check = 1;
          }
        }

   savedModuleName[Scount] = D->connectedDesignModuleName((VarNode * )currentGNode);
   Scount++;

   if (check != 1 )
        {
         storedNodesFromFindRoots[count] = currentGNode;
         count++;
        }

   check =0;
   return;
  }

 GNodeParentArray = currentGNode->parents();
 for(i = 0; i < nParents; i++)
  {
   findRoots(GNodeParentArray[i]);
  }
}
```

Causality from a set of equations

```
import java.io.*;
import java.util.*;
import symantec.itools.awt.Matrix;

class ConMatrix {
  public int rows, columns;
  public double[][] element;

 public static void main(String[] args)
  {
       String[] equation1 =  {"K"};
       String[] equation2 =  {"Qin"};
       String[] equation3 =  {"Qin","Qout"};
       String[] equation4 =  {"K","Qout","P"};
       String[] equation5 =  {"A","P"};

       String[][] equation1to5 = new String[5][];

       equation1to5[0] = equation1;
       equation1to5[1] = equation2;
       equation1to5[2] = equation3;
       equation1to5[3] = equation4;
       equation1to5[4] = equation5;

        String[] all_vars = new String[5];
        int counta = 0;

        boolean no_have_this_var = false;

        for (int i = 0; i < 5; i++)
          for (int j = 0; j < equation1to5[i].length; j++)
          {
             for (int k = 0; k < all_vars.length; k++)
             {
                if (!(all_vars[k] == equation1to5[i][j]))
                {
                   no_have_this_var = true;

                }
                if ((all_vars[k] == equation1to5[i][j]))
                {
                   no_have_this_var = false;
                   break;
                }
             }
             if (no_have_this_var)
             {
                all_vars[counta] = equation1to5[i][j];
                counta++;
                no_have_this_var = false;
             }
          }
         String [][] Matb = new String[5][5];
```

```
     for(int i = 0; i < 5; i++)
    {
        Matb[i][0]= all_vars[i];
        System.out.println("all_vars[" + i + "] = " + all_vars[i]);
    }

  int [][] MatC = new int [5][5];

  int [][] MatA = new int[5][5];

  for (int i = 0; i < 5; i++)
     for (int j = 0; j < equation1to5[i].length; j++)
     {
        for(int k = 0; k < 5; k++)
        {
           if (equation1to5[i][j] == Matb[k][0])
           {
              MatA[i][k] = 1;
           }
        }
     }
  for(int i = 0; i < 5; i++)
     for(int j = 0; j < 1; j++)
    {
        System.out.println("Matb[" + i + "][" + j + "] = " + Matb[i][j]);
    }
     for(int i = 0; i < 5; i++)
     for(int j = 0; j < 5; j++)
    {
        System.out.println("MatA[" + i + "][" + j + "] = " + MatA[i][j]);
    }
       if (isLowerTriangular(MatA))

    for(int i = 0; i < 5; i++)
     for(int j = 0; j < 5; j++)
      {
        if (MatA[i][j] == 1)
        {
            for(int k = 1; k < 5; k++)
               for(int l = 0; l < 5; l++)
               {
                  if (MatA[k][l] == 1)
                  {
                    int rember_j_pos = 100000;
                    for (int m = 0; m < 5; m++)
                    {
                        if ( MatA[k][m] == 1)
                        {
                           rember_j_pos = m;
                        }
                    }
                  }
               }
        }
      }
}
```

```
public static boolean isLowerTriangular(int [][] matrix)
         {
          for (int r = 0; r < matrix.length; ++r)
          {
             for (int c = 0; c < matrix[0].length; ++c)
             {
                if (r < c)
                {
                   if (matrix[r][c] !=0)
                      return false;

                }
             }
           }
          return true;
         }

  public static int [][] transpose(int [][] matrix)
  {
     int [][] result = new int [matrix.length][matrix.length];
     for (int r = 0; r < matrix.length; ++r)
     {
        for (int c = 0; c < matrix[0].length; ++c)
        {
           result[c][r] = matrix[r][c];
        }
      }
   return result;
   }
     public static int [][] shuffleRows(int [][] matrix)
  {
     int [] numbers_used = new int[matrix.length];
     for (int i = 0; i < numbers_used.length; i++)
     {
        numbers_used[i]= 100000000;
      }
     boolean Tr = true;
     boolean random_repete = false;

     int [][] result = new int [matrix.length][matrix.length];
     int count = 0;
     int rand = (int)(Math.random()* 5) ;
     numbers_used[0] = rand;

     for (int r = 0; r < matrix.length; ++r)
      {
       if (r != 0)
        {
           rand = (int)(Math.random()* 5) ;
           while (Tr)
           {
              for (int  i = 0; i < count; i++)
              {
                 if (numbers_used[i] == rand)
                 {
```

```
           random_repete = true;
          }
          if (i == (count - 1) && random_repete == false)
          {
            Tr = false;
            numbers_used[r] = rand;
          }
        }
        if (random_repete)
        {
          rand = (int)(Math.random()* 5) ;
          random_repete = false;
        }
      }
    }
    count++;
    Tr = true;
    for (int c = 0; c < matrix[0].length; ++c)
        {
          result[r][c] = matrix[rand][c];
        }
  }
return result;
}
 public void Matrix(String s) {
     Vector row = new Vector();
     Vector col = new Vector();
     s = s + " ;";
     int i = s.length();
     int j;
     int rowCounter = 0;
     int colCounter = 0;
     String sData = new String();
     Double fl;
     char sChar;
     for (j = 0; j<i; j++) {
            sChar = s.charAt(j);
            if ( (sChar==’ ’)|| (sChar==’,’)|| ( (int) sChar==9)||
                   (sChar == ’;’)||( (int) sChar == 13)||( (int) sChar == 10) ) {
                   fl = new Double(0);
                   try {
                          boolean testSpace = true;
                          int ii;
                          for(ii=0;ii<sData.length();ii++){
                                 testSpace=testSpace&&(sData.charAt(ii)==’ ’); }
                          if(testSpace==false){
                                 fl = new Double(sData);
                                 col.addElement(sData); }
                          sData = new String();
                   }
                   catch (Exception e) {
                          sData = new String();
                   }

                   if ( ( (sChar == ’;’)||( (int) sChar == 13)||( (int) sChar == 10) ) &&
                    !col.isEmpty() ) {
```

```
                row.addElement(col);
                rowCounter = rowCounter + 1;
                sData = new String();
                colCounter = col.size();
                col = new Vector();
            }
        }

        else {
            if ((Character.isDigit(sChar))||(sChar==’.’)||(sChar==’-’)) {
                // allow only digit and decimal point characters
                sData = sData + sChar;     // append to string
            }
        }

    }
    rows = rowCounter;
    columns = colCounter;
    element = new double[rows][columns];
    col = new Vector();
    Double d = new Double(0);
    for (j = 0; j<rows; j++) {
        col = (Vector) row.elementAt(j);
        for (i = 0; i<col.size(); i++) {
            d = new Double((String)col.elementAt(i));
            element[j][i] = d.doubleValue();
        }
    }
    }
}
```

Java Graph Layout

```
/*
 * @(#)Graph.java	1.5 97/07/31
 *
 * Copyright (c) 1994-1996 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license to use,
 * modify and redistribute this software in source and binary code form,
 * provided that i) this copyright notice and license appear on all copies of
 * the software; and ii) Licensee does not utilize the software in a manner
 * which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind. ALL
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
 * IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
 * NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE
 * LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING
 * OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
 * LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER
 * CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF
 * OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control of
 * aircraft, air traffic, aircraft navigation or aircraft communications; or in
 * the design, construction, operation or maintenance of any nuclear
 * facility. Licensee represents and warrants that it will not use or
 * redistribute the Software for such purposes.
 *
 * Changed by: Chris Bucchere, 30-July-1997
 *
 */

import java.util.*;
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

class Node {
   double x;
   double y;

   double dx;
   double dy;

   boolean fixed;

   String lbl;
}

class Edge {
   int from;
   int to;
```

```
  double len;
}

class GraphPanel
 extends Panel
 implements Runnable, MouseListener {
  Graph graph;
  int nnodes;
 Node nodes[] = new Node[100];

  int nedges;
  Edge edges[] = new Edge[200];

  Thread relaxer;
  boolean stress;
  boolean random;

  GraphPanel(Graph graph) {
    this.graph = graph;
    addMouseListener(this);
  }

  int findNode(String lbl) {
    for (int i = 0 ; i < nnodes ; i++) {
      if (nodes[i].lbl.equals(lbl)) {
        return i;
      }
    }
    return addNode(lbl);
  }
  int addNode(String lbl) {
    Node n = new Node();
    n.x = 10 + 380*Math.random();
    n.y = 10 + 380*Math.random();
    n.lbl = lbl;
    nodes[nnodes] = n;
    return nnodes++;
  }
  void addEdge(String from, String to, int len) {
    Edge e = new Edge();
    e.from = findNode(from);
    e.to = findNode(to);
    e.len = len;
    edges[nedges++] = e;
  }

  public void run() {
    while (true) {
      relax();
      if (random && (Math.random() < 0.03)) {
        Node n = nodes[(int)(Math.random() * nnodes)];
        if (!n.fixed) {
          n.x += 100*Math.random() - 50;
          n.y += 100*Math.random() - 50;
        }
      graph.play(graph.getCodeBase(), "../audio/drip.au");
    }
    try {
      Thread.sleep(100);
    } catch (InterruptedException e) {
      break;
    }
  }
}

synchronized void relax() {
  for (int i = 0 ; i < nedges ; i++) {
    Edge e = edges[i];
    double vx = nodes[e.to].x - nodes[e.from].x;
    double vy = nodes[e.to].y - nodes[e.from].y;
    double len = Math.sqrt(vx * vx + vy * vy);
    double f = (edges[i].len - len) / (len * 3) ;
    double dx = f * vx;
    double dy = f * vy;

    nodes[e.to].dx += dx;
    nodes[e.to].dy += dy;
    nodes[e.from].dx += -dx;
    nodes[e.from].dy += -dy;
  }

  for (int i = 0 ; i < nnodes ; i++) {
    Node n1 = nodes[i];
    double dx = 0;
    double dy = 0;

    for (int j = 0 ; j < nnodes ; j++) {
      if (i == j) {
        continue;
      }
      Node n2 = nodes[j];
      double vx = n1.x - n2.x;
      double vy = n1.y - n2.y;
      double len = vx * vx + vy * vy;
      if (len == 0) {
        dx += Math.random();
        dy += Math.random();
      } else if (len < 100*100) {
        dx += vx / len;
        dy += vy / len;
      }
    }
    double dlen = dx * dx + dy * dy;
    if (dlen > 0) {
      dlen = Math.sqrt(dlen) / 2;
      n1.dx += dx / dlen;
      n1.dy += dy / dlen;
    }
  }

  Dimension d = getSize();
```

```
    for (int i = 0 ; i < nnodes ; i++) {
      Node n = nodes[i];
      if (!n.fixed) {
        n.x += Math.max(-5, Math.min(5, n.dx));
        n.y += Math.max(-5, Math.min(5, n.dy));
        //System.out.println("v= " + n.dx + "," + n.dy);
        if (n.x < 0) {
          n.x = 0;
        } else if (n.x > d.width) {
          n.x = d.width;
        }
        if (n.y < 0) {
          n.y = 0;
        } else if (n.y > d.height) {
          n.y = d.height;
        }
      }
      n.dx /= 2;
      n.dy /= 2;
    }
    repaint();
  }

  Node pick;
  boolean pickfixed;
  Image offscreen;
  Dimension offscreensize;
  Graphics offgraphics;


  final Color fixedColor = Color.red;
  final Color selectColor = Color.pink;
  final Color edgeColor = Color.black;
  final Color nodeColor = new Color(250, 220, 100);
  final Color stressColor = Color.darkGray;
  final Color arcColor1 = Color.black;
  final Color arcColor2 = Color.pink;
  final Color arcColor3 = Color.red;

  public void paintNode(Graphics g, Node n, FontMetrics fm) {
    int x = (int)n.x;
    int y = (int)n.y;
    g.setColor((n == pick) ? selectColor : (n.fixed ? fixedColor : nodeColor));
    int w = fm.stringWidth(n.lbl) + 10;
    int h = fm.getHeight() + 4;
    g.fillRect(x - w/2, y - h / 2, w, h);
    g.setColor(Color.black);
    g.drawRect(x - w/2, y - h / 2, w-1, h-1);
    g.drawString(n.lbl, x - (w-10)/2, (y - (h-4)/2) + fm.getAscent());
  }

  public synchronized void update(Graphics g) {
   Dimension d = getSize();
   if ((offscreen == null) || (d.width != offscreensize.width) || (d.height != offscreensize.height)) {
    offscreen = createImage(d.width, d.height);
    offscreensize = d;
```

```
      offgraphics = offscreen.getGraphics();
      offgraphics.setFont(getFont());
    }

    offgraphics.setColor(getBackground());
    offgraphics.fillRect(0, 0, d.width, d.height);
      for (int i = 0 ; i < nedges ; i++) {
        Edge e = edges[i];
        int x1 = (int)nodes[e.from].x;
        int y1 = (int)nodes[e.from].y;
        int x2 = (int)nodes[e.to].x;
        int y2 = (int)nodes[e.to].y;
        int len = (int)Math.abs(Math.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)) - e.len);
        offgraphics.setColor((len < 10) ? arcColor1 : (len < 20 ? arcColor2 : arcColor3)) ;
        offgraphics.drawLine(x1, y1, x2, y2);
        if (stress) {
          String lbl = String.valueOf(len);
          offgraphics.setColor(stressColor);
          offgraphics.drawString(lbl, x1 + (x2-x1)/2, y1 + (y2-y1)/2);
          offgraphics.setColor(edgeColor);
        }
      }

      FontMetrics fm = offgraphics.getFontMetrics();
      for (int i = 0 ; i < nnodes ; i++) {
          paintNode(offgraphics, nodes[i], fm);
      }
      g.drawImage(offscreen, 0, 0, null);
  }

//1.1 event handling
public  void mouseClicked(MouseEvent e) {
}

public  void mousePressed(MouseEvent e) {
  double bestdist = Double.MAX_VALUE;
  int x = e.getX();
  int y = e.getY();
  for (int i = 0 ; i < nnodes ; i++) {
    Node n = nodes[i];
    double dist = (n.x - x) * (n.x - x) + (n.y - y) * (n.y - y);
    if (dist < bestdist) {
      pick = n;
      bestdist = dist;
    }
  }
  pickfixed = pick.fixed;
  pick.fixed = true;
  pick.x = x;
  pick.y = y;
  repaint();
  e.consume();
}

public  void mouseReleased(MouseEvent e) {
  pick.x = e.getX();
  pick.y = e.getY();
  pick.fixed = pickfixed;
  pick = null;
  repaint();
  e.consume();
 }

 public  void mouseEntered(MouseEvent e) {
 }

 public  void mouseExited(MouseEvent e) {
 }

 public  void mouseDragged(MouseEvent e) {
  pick.x = e.getX();
  pick.y = e.getY();
  repaint();
  e.consume();
 }

 public  void mouseMoved(MouseEvent e) {
 }

  public void start() {
    relaxer = new Thread(this);
    relaxer.start();
  }
  public void stop() {
    relaxer.stop();
  }
}

public class Graph
 extends Applet
 implements ActionListener {
  GraphPanel panel;

  public void init() {
    setLayout(new BorderLayout());

    panel = new GraphPanel(this);
    add("Center", panel);
    Panel p = new Panel();
    add("South", p);
    p.add(new Button("Scramble"));
    p.add(new Button("Shake"));
    p.add(new Checkbox("Stress"));
    p.add(new Checkbox("Random"));

    String edges = getParameter("edges");
    for (StringTokenizer t = new StringTokenizer(edges, ",") ; t.hasMoreTokens() ; ) {
      String str = t.nextToken();
      int i = str.indexOf('-');
      if (i > 0) {
        int len = 50;
        int j = str.indexOf('/');
```

```
       if (j > 0) {
         len = Integer.valueOf(str.substring(j+1)).intValue();
         str = str.substring(0, j);
       }
       panel.addEdge(str.substring(0,i), str.substring(i+1), len);
     }
   }
   Dimension d = getSize();
   String center = getParameter("center");
   if (center != null){
     Node n = panel.nodes[panel.findNode(center)];
     n.x = d.width / 2;
     n.y = d.height / 2;
     n.fixed = true;
   }
 }

 public void start() {
   panel.start();
 }
 public void stop() {
   panel.stop();
 }

public void actionPerformed(ActionEvent e) {
 Object arg = e.getSource();
 if (arg instanceof Boolean) {
  if (((Checkbox)arg).getLabel().equals("Stress")) {
   panel.stress = ((Boolean)arg).booleanValue();
  } else {
   panel.random = ((Boolean)arg).booleanValue();
  }
  return;
 }
 if ("Scramble".equals(arg)) {
  play(getCodeBase(), "../audio/computer.au");
  Dimension d = getSize();
  for (int i = 0 ; i < panel.nnodes ; i++) {
   Node n = panel.nodes[i];
   if (!n.fixed) {
    n.x = 10 + (d.width-20)*Math.random();
    n.y = 10 + (d.height-20)*Math.random();
   }
  }
  return;
 }
 if ("Shake".equals(arg)) {
  play(getCodeBase(), "../audio/gong.au");
  Dimension d = getSize();
  for (int i = 0 ; i < panel.nnodes ; i++) {
   Node n = panel.nodes[i];
   if (!n.fixed) {
    n.x += 80*Math.random() - 40;
    n.y += 80*Math.random() - 40;
   }
  }
  }
 }

 public String getAppletInfo() {
  return "Title: GraphLayout \nAuthor: <unknown>";
 }

 public String[][] getParameterInfo() {
  String[][] info = {
    {"edges", "delimited string", "A comma-delimited list of all the edges.  It takes the form of 'C-N1,C-N2,C-N3,C-
NX,N1-N2/M12,N2-N3/M23,N3-NX/M3X,...' where C is the name of center node (see 'center' parameter) and NX
is a node attached to the center node.  For the edges connecting nodes to eachother (and not to the center node) you
may (optionally) specify a length MXY separated from the edge name by a forward slash."},
    {"center", "string", "The name of the center node."}
  };
  return info;
 }
}
```