MANAGEMENT OF INTEGRATED PRODUCT AND PROCESS MODELS THROUGH AUTOMATED DECOMPOSITION

by

SHAUN M. ABRAHAMSON

B.Sc. Electro-Mechanical Engineering
University of Cape Town, South Africa, 1995

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

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

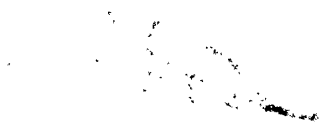
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[February 1999]
January 1999

Signature of Author.....
Department of Mechanical Engineering
January 15, 1999

Certified by.....
David Wallace
Esther and Harold E. Edgerton Associate Professor of Mechanical Engineering
Thesis Supervisor

Accepted by.....
Ain A. Sonin
Chairman, Department Committee on Graduate Students



MANAGEMENT OF INTEGRATED PRODUCT AND PROCESS MODELS THROUGH AUTOMATED DECOMPOSITION

by

SHAUN M. ABRAHAMSON

Submitted to the Department of Mechanical Engineering
on January, 1999 in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Mechanical Engineering

ABSTRACT

A method is proposed to assist participants in the design of complex integrated models for engineering design. The approach builds on the DOME (Distributed Object-based Modeling and Evaluation) framework, which is intended to facilitate the creation of integrated models by allowing groups of designers to link services of models encapsulated by distributed objects. The aim is to provide a tool for users to understand dependencies between sub-models through the visualization and evaluation of the resulting integrated model structures. A tool is designed and implemented and is used to analyze DOME model networks and provide decomposition feedback for resource allocation during the design process.

Integrated modeling concepts are introduced and an industrial case study is used to describe a service exchange network approach to system modeling. Then, decomposition methods are discussed and classified by dependency types as serial, functional, informational and intellectual. The need for decomposition support to create integrated models through the services of distributed objects, like those used in DOME, is evaluated and illustrates the need for decomposition tools.

Thesis Supervisor: David Wallace

Title: Ester and Harold Edgerton Associate Professor of Mechanical Engineering

ACKNOWLEDGEMENTS

The author would like to thank a number of people who have made this thesis possible. My family at home in South Africa and here in the United States, who have supported me in every way possible throughout my time at MIT. Prof. David Wallace for his guidance and patience and for giving me the opportunity to explore ideas. To all the member of the Computer Aided Design Lab at MIT for providing a stimulating and entertaining work environment.

Particular thanks to the following people for support with the software development: Benjamin Linder, Mathew Wall for support with the DOME components. Nick Borland for general assistance throughout my stay in the Lab. Thanks to those who were involved in the construction of the case study model: Tom Almy, Inis Sosa and Tina Savage and members of the Polaroid team. Also thanks to the following people for assisting in the development of many of the ideas and concepts: Nichola Senin, Nader Sabbaghian, Samir Patil and members of he Axiomatic Design Group. Work reported in this thesis was supported (in part) by the MIT Center for Innovation in Product Development under NSF Cooperative Agreement Number EEC-9529140.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Problem Statement

The DOME (Phang 1997) software framework is being developed to support the synthesis of mathematical system models from the numerous sub-models used in the product development process. The aim of this work is to provide tools for users to understand dependencies between sub-models through visualization and evaluation of the resulting integrated model structures. This information should allow detailed analysis of product development process and subsequently, better resource allocation as shown in Figure 1-1.

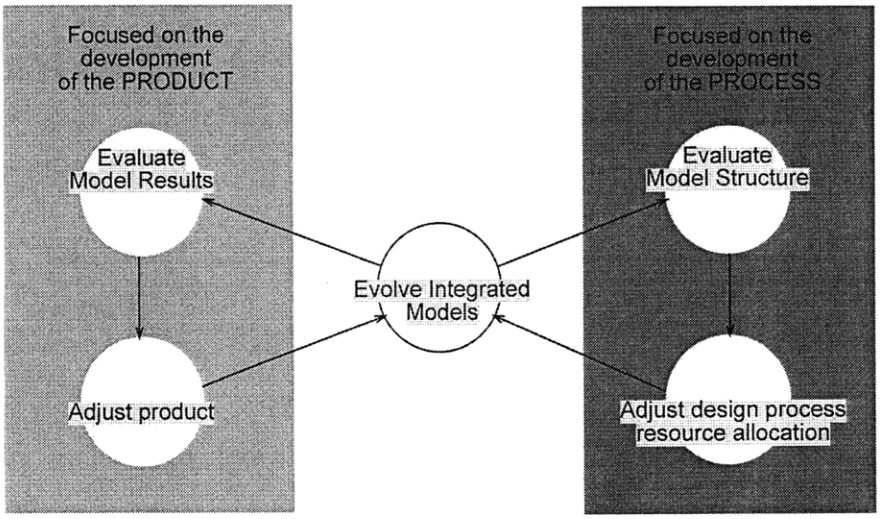


Figure 1-1 Integrated model evaluation for product and process feedback. The left side of the process shows the role of the integrated model for making decisions about the design object. The right side reflects the objective of this work, providing feedback about the design process by evaluating integrated models.

1.2 Problem Overview

There are a number of approaches to address the problem of integrating models in the product design and development process. The objectives are generally the same, that is, integration of models, which represent different considerations to facilitate global, rather than local trade-offs.

Although there are a number of aspects that distinguish the various approaches, one of the fundamental issues is problem representation. There is a continuum marked by two distinct poles. On the one end, there are those who wish to build common representations in which all participants can represent their problems (STEP 1999). The other pole is defined by those who allow each model define the representation, which allows them to best represent the problem they are considering. Mechanisms are then provided to locally map one representation to another (Cutkosky, 1994). The DOME framework is based on the latter.

DOME provides mechanism for users to create models within heterogeneous environments or native to the DOME environment. Various mechanisms are provided to allow organization of sub-models, using an object-based approach. Users are then able to make model services available to other users who can subscribe to these services. As services are connected using locally defined relationships, a service network is created which can then be evaluated as a complete simulation.

Evaluation takes place by running the various sub-models according to the relationships and service connections defined by each of the users. Independent variables can be adjusted and variables can be evaluated to determine an overall score. In this way, the locally defined models can be evaluated from a global perspective. The evaluation can be automated since the number of scope and scale of the model rapidly becomes intractable for any single person or group.

In addition to the complexity of the model operation, there are a number of challenges related to the tractability of the resulting network. The organization of the global network structure is referred to as the **decomposition**. This refers both to how models are reduced to sub-models and how models are integrated to form integrated models. Although it has been acknowledged in a variety of fields that decomposition and relationships are the basis of problem solving and modeling (Minsky 1986, Pinker 1997), little is understood about how these activities can be supported for individuals and even less about multi-disciplinary groups.

The approach used in this work is to identify domain independent types of decomposition. This is used to isolate mechanisms needed to support decomposition activities and the construction of models. **Heterarchies** are identified as a likely emergent structure and as a result the work was focused on providing tools to support the navigation and understanding of such structures. Two tools are created to meet these objectives.

The first is a visualization tool that facilitates the navigation of large heterarchies by providing mechanisms for adjusting levels of abstraction. The other is an extensible graph analysis algorithm, which can be used to organize sub-models according to their relative **connectedness**[1] within the scope of the network under consideration. These tools are combined to create an approach, which allows real-time responses to resource allocation during construction of integrated models.

[1] **Connectedness** refers to the degree of interdependence between a given element and others

Although there are a number of approaches suggested for managing the design process, they suffer from lack of timely, detailed information about the process state. The information quality is a function of the effort expended to collect the data, as can be determined from an example collection process. In the creation of a design process outline for a small (less than 20 parts) automotive subsystem, which takes 6 person (Dong 1998) weeks to collect, the data can be used once during the three month design process. The proposed method makes process information available in real-time, allowing responses to the dynamics of the design process. Analysis of the integrated model structure is used to produce a design process view, without additional overhead.

1.3 Overview of the thesis document

The thesis begins by describing integrated modeling approaches for product design and development. The DOME (Distributed Object-based Modeling Environment) approach is introduced and an example industrial problem is used to review current capabilities and limitations. This is followed by a broad discussion of decomposition intended to provide an overview of the scope of the problem The result is a classification of decomposition types.

The current DOME implementation is then discussed, identifying mechanisms, which support different decomposition types. This identifies the need for additional support forming the requirements for a prototype decomposition tool. An implementation of an analysis and visualization tool is then presented, reflecting on different analysis and visualization approaches.

A design process management process is then introduced. Current limitation of design process management are explained and particular areas are identified which can be addressed through the use of the DOME modeling environment and the integrated visualization and analysis tool. Finally conclusions are drawn, focusing particularly on a number of possible directions for future work.

2 BACKGROUND

This section consists of two main parts. Section 2.1 introduces ideas of integrated modeling in general and section 2.3 explains DOME (Distributed Object-based Modeling and Evaluation) framework in particular.

2.1 Overview of integrated modeling

Predicting product performance and the ability to make informed decisions are critical to improving the product design and development process. In particular, integrated modeling and decision making can improve quality and reduce development time through the elimination of costly design, build, test and refine cycles. The objective of integrated modeling is to facilitate real-time coupling between design attributes derived from product design models. Figure 2-1 contrasts the physical and analytical aspects of design process modeling with respect to degrees of integration. Integrated analytical models, both large in **scope**[2] and **scale**, have traditionally been considered infeasible. There are a number of reasons for this, as expressed in the following statements (Cutkosky 1994).:

1. It is very difficult, if possible, to formulate a complete explicit product development model that can be solved, both because of its size, complexity, and its dynamic, uncertain and evolving nature
2. The different product development domains use different tools, data models, and often even different data management systems.
3. All necessary data are not available to a single entity, due to either consolidation logistics or proprietary issues.

It is clear that the scope and scale of the effort is large and there are a number of ongoing projects attempting to solve parts of the problem. Some information technology issues are being resolved through new technologies intended to enable integration of heterogeneous systems. Further, in certain areas, there is increasing effort to form open standards and collaboration between vendors, for example in Computer Aided Design (CAD), Computer Aided Machining (CAM) and Product Data Management (PDM).

[2] Scope refers to the number of different disciplines and analysis, while scale refers to the overall size in terms of the number of people and software tools.

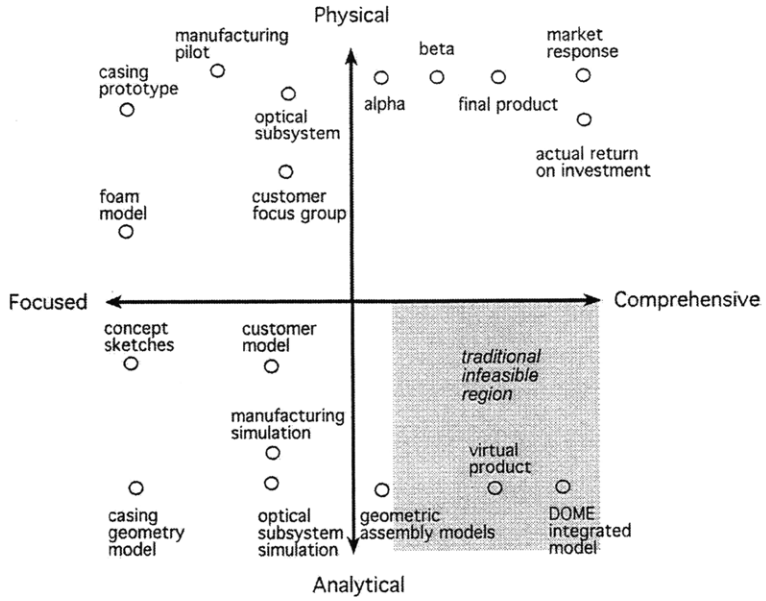


Figure 2-1[3] The role of DOME in product development. The comprehensive analytical model has traditionally been considered infeasible. This is in part because of the difficulty in modeling some elements of the design, but also because of the integration difficulties associated with connecting different sub-models.

The following sections will show how integrated modeling is being addressed in the DOME framework. We begin by presenting our definition of the requirements for integrated modeling. Within this context, different approaches are discussed and the DOME implementation is introduced. DOME is then explained in the context of the development of an integrated model for a Liquid Crystal Video Projector.

2.2 Discussion of approaches to integrated modeling

A number of groups have described requirements for integration frameworks (Molina, Akman Cutkosky, Tomiyama). Based upon these readings and work with industry collaborators, the developers of the DOME framework feel that the key requirements for an integrated product development system include:

1. *Low barriers to participation:* information system architectures that allow distributed users in different environments to participate in a transparent manner without investing significant time for training to learn systems, tools, or modeling languages outside of their own expertise.

[3] Adapted from Product Design and Development, Ulrich and Eppinger 1995

2. *Domain specific choice of tools:* allowing each design participant/expert use the tools, representations, simulations, heuristics or models which are most suitable within their domain.
3. *Flexibility at the user level:* flexibility to allow for the spontaneous and robust growth, extension, change, revision and reuse of integrated models, tools, or resources to solve evolving or new problems.
4. *Mixing of level of detail and quality of models:* incorporate a seamless mix of detailed models and incomplete or approximate models to support both top down and bottom up design.
5. *Real-time simulation:* provide the ability to explore the design solution space, concurrently elicit trade-offs or causality between disciplines and goals, and monitor design evolution—in both a manual and automated fashion.
6. *Decision support:* support decision making through visualization and evaluation.
7. *Support different collaboration models:* accommodate both tight and loose collaboration ranging from close colleagues to outside consultants or suppliers while respecting a diverse set of intellectual property and work-sychronization needs.

Theses requirements are focused on **participants** in the process, who fulfill the various disciplinary roles. These people must have the capability to affect the modeling environment without having to work through intermediaries such as system administrators. The approach supports individuals and groups of people as well as individual and networked computation and the combinations of these elements. The following combinations capture this view (adapted from Branki 1995):

1. communication between designers
2. communication between designers and tools
3. communication between tools

We expect that the people and applications will observe and represent the world in a way that is amenable to the tasks which they wish to perform. One of the integration challenges is facilitating interaction of designers and tools *with different **views** and data **representations***. Another critical aspect of integration is support for model construction and decision support, since as we move to incorporate multiple domains, it becomes increasingly *likely that the resulting **solution spaces**, will **be intractable***. Providing tools that allow users to represent desired states or outcomes are important; as are means to assist users in finding promising solutions.

2.2.1 Integration and data model representations

Integration can be approached in a variety of ways, however it can be viewed as a continuum, bounded by two approaches. One end *requires participants to represent their views in a common way*, which can then be understood by other people or machines. The other end integrates by translating between elements, which wish to communicate. An example of the first, are standards such as STEP, which create a general data representation in which people can communicate. An example of the second are ontologies (Cutkosky 1994) used to define exchanges between entities, with *no central definition*. Somewhere in the continuum are approaches such that use intermediate representations such as *qualitative process theory* (Tomiyama 1994), which allow integration once concepts from the new modeling domain has been mapped to the overall representation. The representation issue is at the heart of the decomposition issue which will be introduced in the section 3.

The DOME framework supports a model that allows users to work with the tools they are familiar with and as a result, there is **minimal prescribed representation**. Part of the motivation for this, is that we expect that representations for different domains will evolve in directions that best serve the particular domain, making a central representation intractable. Following a similar approach to that of Cutkosky, we allow users to wrap applications and **publish** interfaces. Rather than use ontologies for translation, we provide a publish/subscribe mechanism through a **service marketplace**. Services can be published and then manipulated in a graphic programming environment and used by other users and tools.

It is worth mentioning that integration through language specification provides a number of advantages for automation of aspects of the integration. Richer representations facilitate more elaborate processing. For example, Tomiyama et al use qualitative process theory as an integrative language requiring each sub-model to be mapped to this global representation. This holds the potential of automating the integration step, however the designer is required to understand the global view as well as the mapping of the tool. It is not clear what overhead is involved in the creation of the mapping.

A further consideration is the representation is uncertainty. At different stages in the evolution of a model, some parts will be uncertain. For example, we might be able to state exactly the geometry of an object, but we might not be certain about the process, which will be used to manufacture it or the consistency of its material properties. Similarly, we might have a model of a given resolution, that is correct within a certain range, such as a first order estimate. We view this as essential aspect of the design process, therefore we have generalized probabilistic representations, allowing users to represent uncertainty and facilitating the mixing of different levels of modeling detail.

Design is increasingly a distributed activity, including distributed data storage and computation as well as network access. In most cases, it is likely that when dealing with multiple data owners, there is little hope for central data storage. This issue is tightly coupled to security concerns as well as different representation schemes. This in turn means that no system can expect to have a **complete model view**, making centralized approaches such as blackboard architectures less desirable. The network capabilities and appropriate mechanisms are built into the DOME modeling environment, such that as long as services can be published on a network, they can be included in the modeling environment and therefore be integrated.

Once integration is achieved, it becomes essential to provide decision support tools, given the anticipated size of resulting integrated models. Although there are numerous optimization schemes, most require a classification of the problem beforehand. Also, in incomplete models, the focus will be on identifying potential solution, rather than optimal ones. The current DOME approach makes use of genetic algorithms to identify areas if interest within the solution space. Users are able to specify independent variables and specify criteria for different metrics, which are then used by the GA to define objective functions. The modular architecture, means that objects encapsulating other optimization or search strategies might also be used.

Although integrated models, promise a great deal, a number of issues conspire to make them unmanageable, like with any large software system (De Pauw 1997) At the heart of this problem is understanding the relationships between different parts of the model. If one considers that there are multiple participants creating different parts of the model, the overall structure is continually evolving, creating the need for a capability to monitor and understand the form at any time. This can be referred to broadly as **decomposition**. The case study in the following section will introduce the idea and section 3 is devoted to a more comprehensive discussion.

2.3 Description of DOME in the context of the Polaroid Project

This case study introduces the implementation of the DOME integrated modeling environment. The case is used to demonstrate the **scale** and **scope** of integrated modeling efforts and affords an opportunity to explain the current features of the DOME framework. The particular challenges of integrated modeling relate to the management of large models, which have been *evolved by multiple people.* A review of the implementation process that it is *unlikely that any individual or group will understand the complete model structure.* This results in a number of model management challenges.

The development of an integrated model is explained for a Liquid Crystal Projector. In the following sections, the model evolution is explained as per the timeline in Figure 2-2.The integrated model begins with a broad statement of the design objectives, considering initial marketing data as well as existing and potential critical component suppliers. The model evolves to address a variety of technical, supplier, marketing, environmental, cost and financial questions, through the participation of multiple suppliers, consultants and engineers and the connection of services from their respective models.

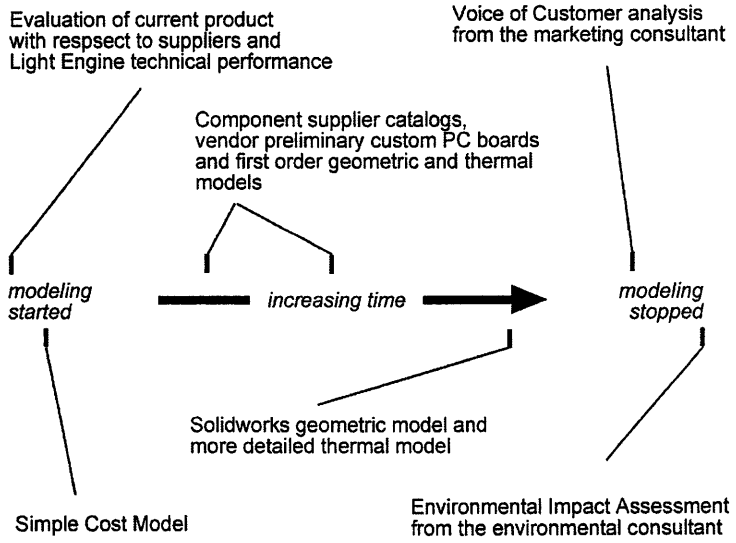


Figure 2-2 The timeline for the evolution of the integrated modeling effort. Modeling begins with an evaluation of current product attributes and continues through a variety of expansion stages culminating in an integrated model including customer, supplier, engineering, financial, costing and environmental sub-models.

Figure 2-2 shows the model structure for the initial objectives, which were to evaluate the performance of the current *light-engine* component against new alternatives. The circles represent **modules**, or containers, which encapsulate data and models. The lines connecting the modules reflect information paths. This representation reflects the object-based nature of the framework. The light-engine module contains data pertaining to 3 different suppliers. The other modules contain the light-engine technical performance **specifications** and light-engine vendor **metrics**, respectively. In this case, the modules are created by different individuals.

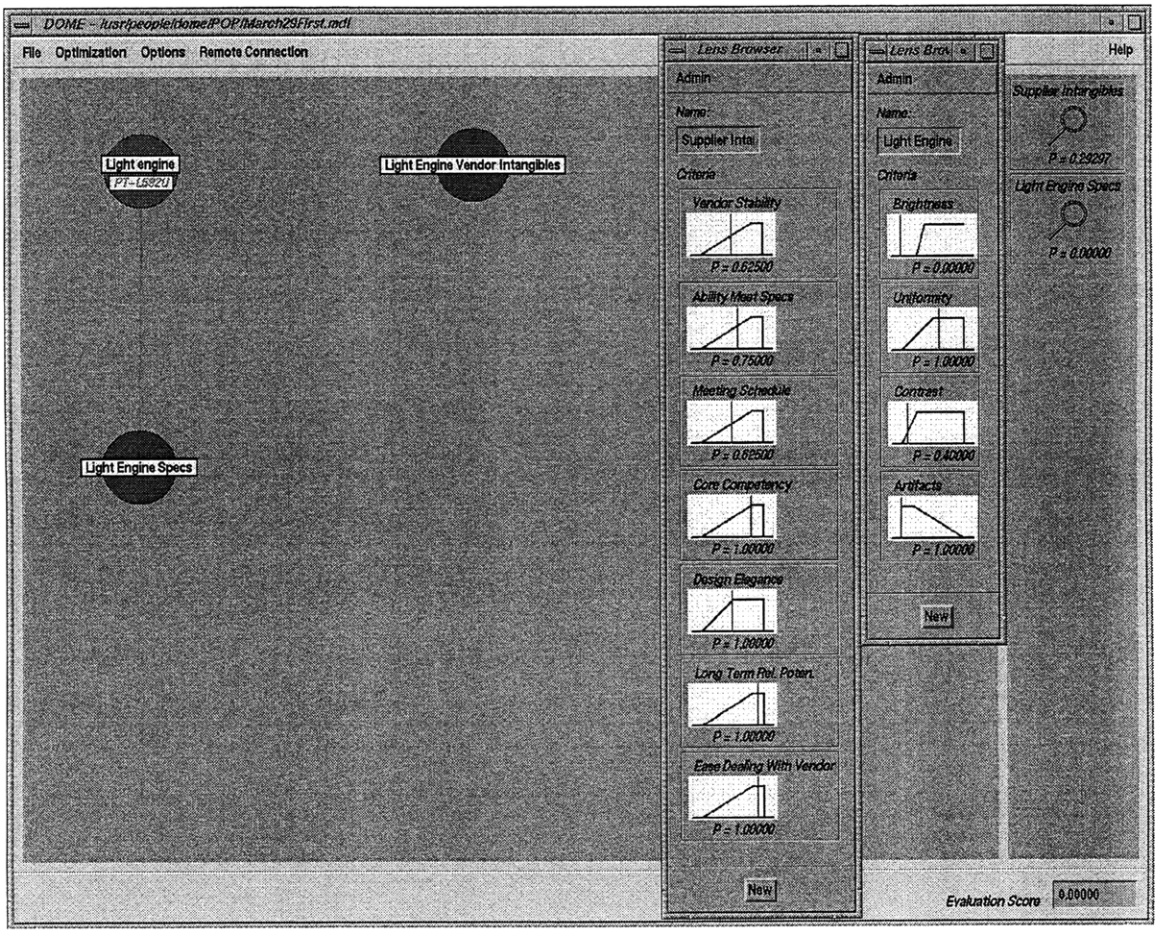


Figure 2-3 Polaroid model first state in the DOME graphical user interface. On the left are modules, connected by arcs, illustrating information exchanges. Right of center are two panels which display evaluations of various attributes with respect to specifications.

The two panels on the right show **evaluation** scores of various parameters with respect to the light-engine specifications. These displays are a specific type of module, which provide **evaluation services**, calculating and displaying the relationship between the current **design-state** and various **objective functions** (specifications and metrics). The vertical lines represent the state of the design, while the sloping forms represent the specifications expressed as probability distributions. Figure 2-4 shows the addition of a cost module for the assessment of cost attributes of the light engine.

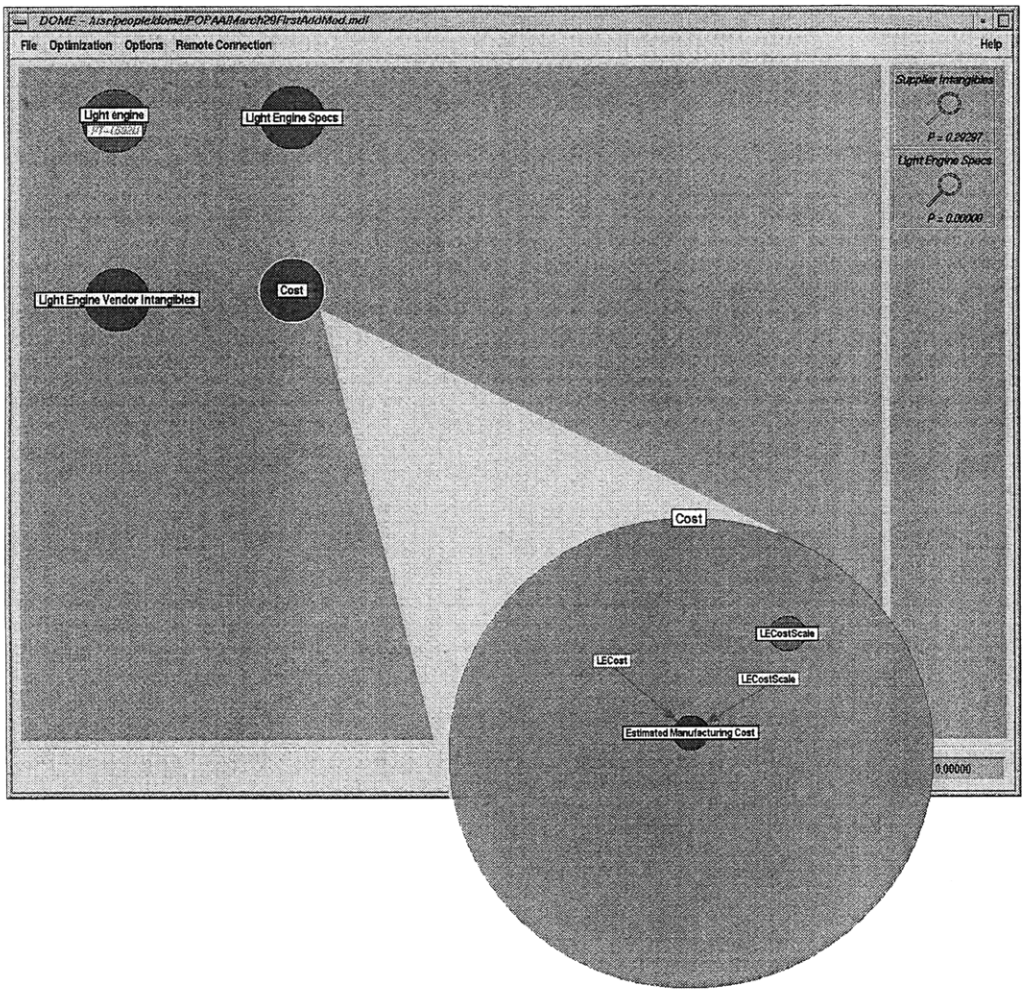


Figure 2-4 Addition of a simple cost model. The image shows the addition of a simple cost model to extend the capabilities of the evaluation.

2.3.1 Creating inter- and intra-module service exchanges using relations

We add a simple cost estimator based on the light engine cost and scale factor intended to represent the effect of different sales volumes on the costs. A **relation** is formed as follows:

(Predicted Manufacturing Cost) = (LECost) + (LECostScale)

LECostScale is created locally, as is the relation result **(Predicted Manufacturing Cost)**, however an additional service is required to provide the light-engine cost which is "owned" by another module. It is important to note that while this might simply be a data point obtained from a supplier, it might also represent the result of a mathematical computation. This might in fact be the result of the suppliers' model, which computes a predicted cost based on combinations of features (as might be the case if the component is in the process of being designed). The relationship statements[4] might consist of control flow statements or algebraic functions. For example, the scaling factor for the light-engine cost, is conditional on certain production volumes.

When multiple relations are created, a change occurring in part of the model results in the network re-evaluating the requisite relations. The result is a recursive update of the **service network**. In this way different model components react to changes in real-time and the connected modules behave as an integrated system. The behavior of the system is *not determined centrally, but is emergent, based on locally defined relations.* This mechanism has been refined in the latest implementation and is discussed in detail in Section 4.1.1.

[4] The relationships can be any C++ style statement

2.3.2 Modules for evaluating the current state of the design model

Evaluation of the design is carried out by comparing attributes of the current design-state to the specifications provided by the various participants in the design process. Although a number of multi-attribute utility-based methodologies exist, a **satisficing** methodology (Kim 1997) is used. The evaluation can be thought of as a comparator, which takes in a specification and a performance attribute and then produces an output which reflects the relationship between them (in this case a score from 0 to 1.0).

When a new Light-Engine vendor is selected, the **lenses** and the propagation mechanism cause an update to reflect the new scores for this selection. This update happens because the lens is using services from the light engine catalog. In this case, the relation is defined by a criterion, which compares the stated performance to the desired performance specification. In addition to the graphic view provided of this evaluation, the results are used by a decision module, which calculates a multi-attribute evaluation, providing an overall view of the design state. The evaluation process is shown in Figure 2-5.

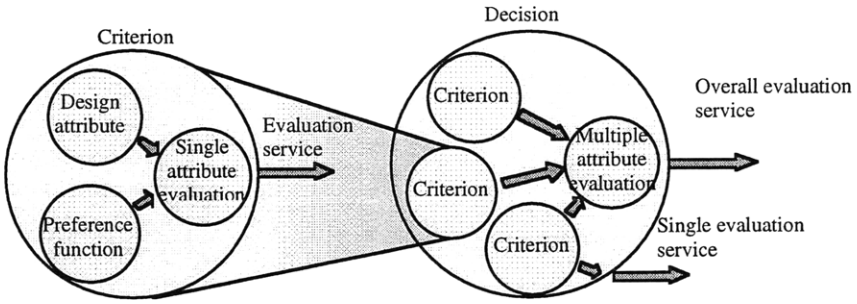


Figure 2-5 Operation of lenses and criterion in DOME models. This graphics show how attributes are evaluated with respect to specifications, the results of which are used to compute a multi-attribute utility for the given state. Specifications are presented in the form of probabilistic representations and compared to the attributes (in whatever form the are presented).

2.3.3 Probabilistic Representation and Generalized Variables

Probabilistic representation allows the representation of uncertainty encountered in different stages of the design. An example might be first order calculation results for initial estimates such as the inputs for the cost model. In the case of the detailed cost model, some of the inputs to the model are uncertain, such as sales volume. This does not prevent us from using the model; rather, we represent the variables appropriately and expect to have uncertain results. Other examples of the application include **service quality**, where data is passed through an "uncertainty filter", to account for confidence. Computation using probability is, however, transparent to the user; the system selects appropriate solution techniques, depending on the distribution types and the relationship between variables as shown in Figure 2-6 below.

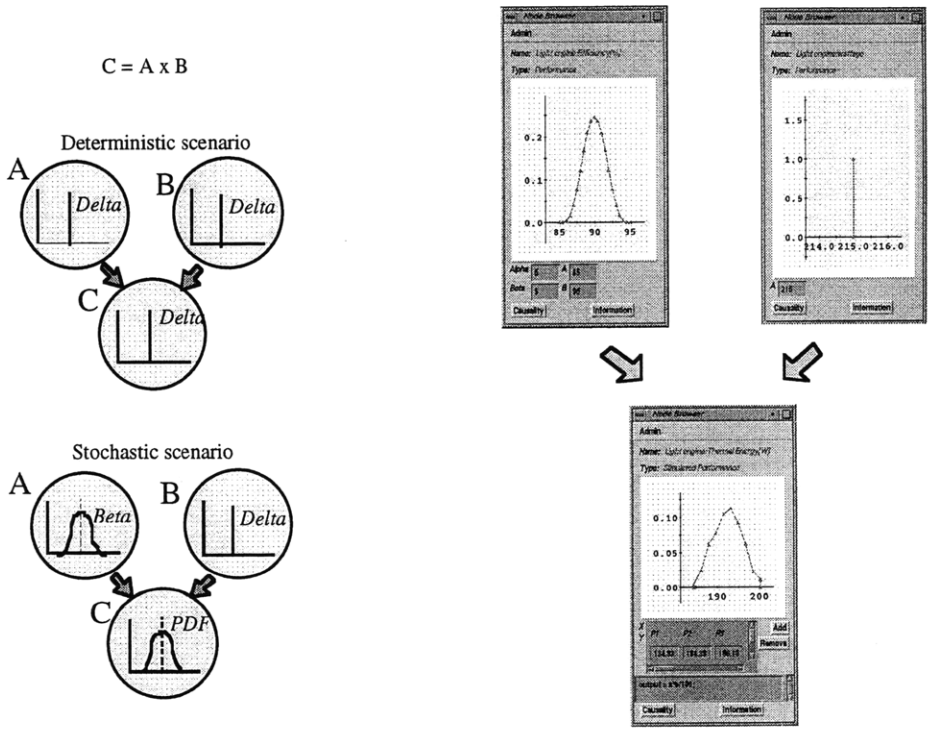


Figure 2-6 Illustration of variable generalization. The figures on the left illustrate the difference between the computation of the statement C = A + B. The upper case is the deterministic case, where the result C is a delta function because both A and B are delta functions. The lower case replaces A with a probabilistic value and shows the result C to be probabilistic as well. The images on the left are from a real model showing the input values and the resulting output.

The flexibility in the definition of variable types and creating and connecting services, provides an extensible environment where models can be evaluated at any time regardless of the **completeless** or **uncertainty** of information. Additionally, we can connect different modeling tools and maintain the integrated model performance. In this regard, the framework provides functionality associated with agent-based sysytem such as **flexibility** and **extensibility**.

The addition of a **gemoetric model** presents an opportunity to show how the two previously discussed features can be combined. The geometric model provides services for thermal models, cost and marketing models. The model is however using partically complete information. For example, the components are at different satges of completeness. Off the shelf items such as speakers and fans are complete. The power supply however, will be made to fit the available space, within certain constraints. It's form can therefore be computed by considering the power requiremnets and available dimensions. The results can then be used by the power supply manufacturers.

As can be seen from Figure 2-7, **thermal performance** of different components are monitored by comparing their rated operating temperature range to the calculated internal temperature of the case. This demonstrates a similar idea to constraint propagation. Users interested in a different aspect of the model, for example the fan cost or size, are able to experiment with alternatives and receive real-time feedback about the thermal performance impacts.

The model has been extended from a basic customer and cost based analysis, to include some of the interactions between different aspects of the design, from an engineering perspective. The network resulting from the various **relations** and the **update mechanism** make this integrated understanding possible. However, solving this network rapidly becomes intractable considering the number of independent parameters and the objective functions, which must be satisfied. We can automate the search, using a **search technique** based on a **genetic algorithm**, to identify promising areas of the solution space.

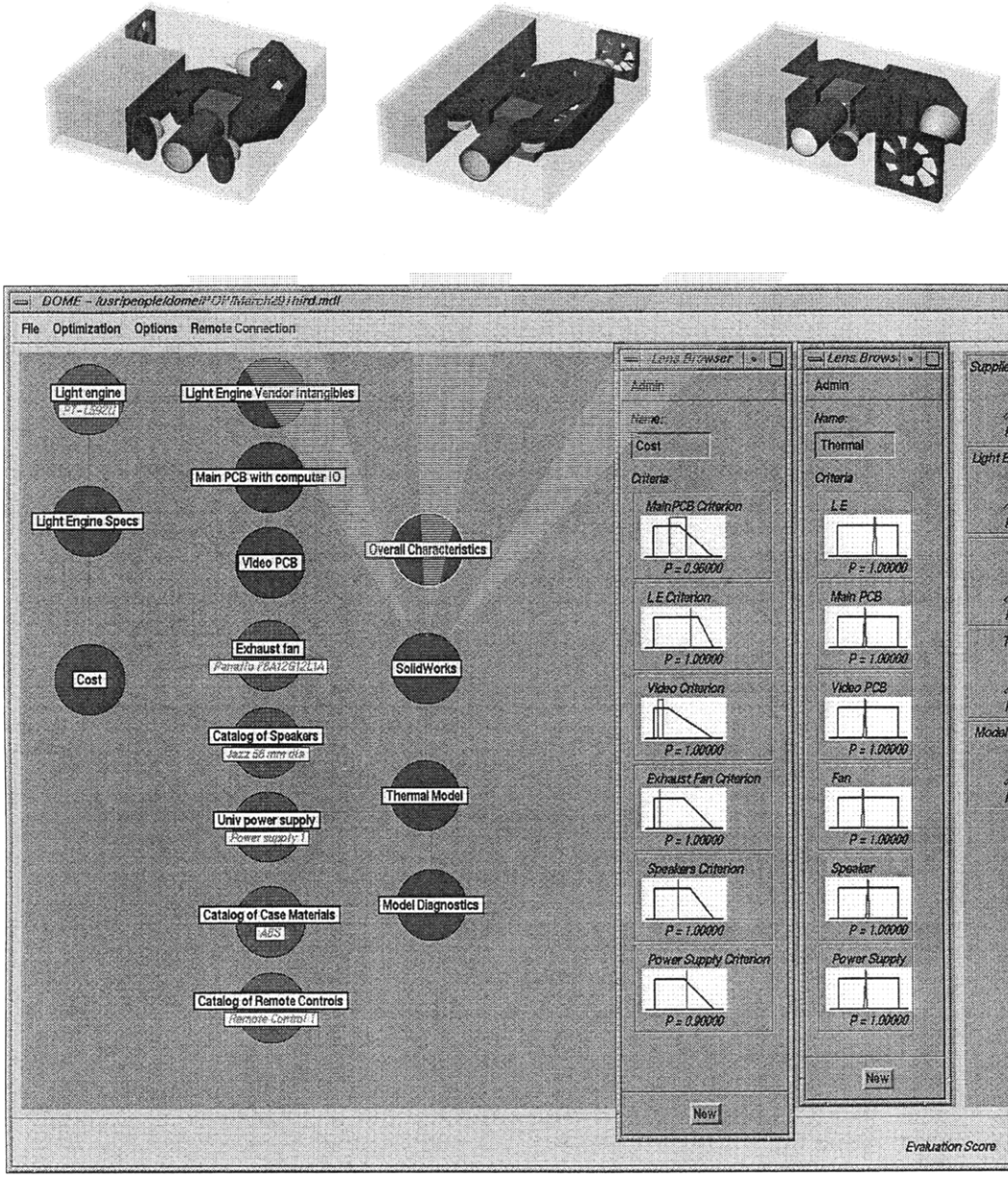


Figure 2-7 Addition of a geometric tool from a CAD tool. The addition of a geometric model is completed to provide services to a variety of other sub-models. Material usage and thermal calculations can be completed more accurately in response to changes to configurations and part dimensions. Further, abundant information is now available to the marketing by way of form and mass properties.

2.3.4 Solution space searches using genetic algorithms

The current implemented search technique is a genetic algorithm search (Senin 1996). The search manipulates **catalog choices** and **continuous independent** variables and evaluates resulting scores by receiving feedback from the decision modules. In this way it moves through the space identifying combinations of variables which yield higher scores. These may or may not be globally optimal, but will reflect combinations that produce solutions, which are relatively better than others.

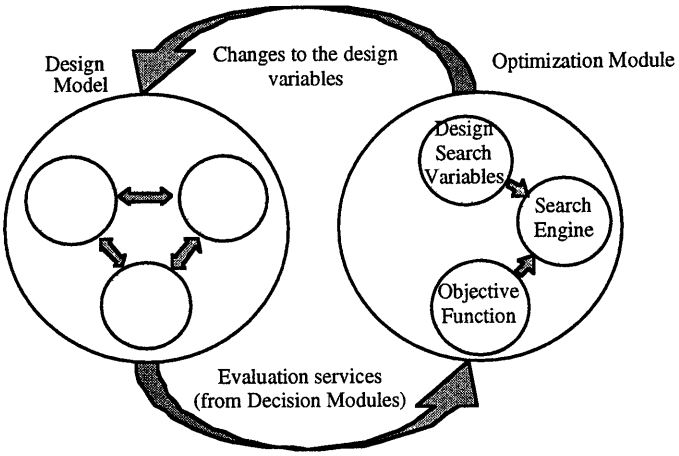


Figure 2-8 Diagram of genetic algorithm search process. The optimization module is able to make changes to a user- defined group of design variables based on user selections. The changes result in an update of the network, which in turn cause the evaluation modules to update. The result of the update is then used to further direct the search.

2.3.5 Addition of Voice of Customer and Environmental Models

The final evolution of the integrated model includes two more distributed connections; the first of which, is **TEAM**[5], an **environmental expert system**. This model provides life cycle analysis feedback to the designers using a variety of impact assessment schemes. This allows designers to receive detailed feedback on environmental issues, while also allowing different evaluation schemes to be substituted depending on the use location or particular **preference set** for the target market segment.

[5] TEAM is a product of Ecobilan

The second extension includes the connection to a remote marketing consultant, using utility metrics to assess customer responses to different designs. The connection can be run as either **synchronous** or **asynchronous**, allowing the consultant to evaluate only selected solutions and then make their assessments available to the designer. This functionality allows participants to select the mode in which they wish to participate.

2.3.6 Publishing and subscribing from different software tools

The connection of the **environmental** model provides an opportunity to revisit the manner in which services are published. Figure 2-9 shows a snapshot of an interface to a publishing program. The upper part of the interface allows users to define an interface to this model along with descriptions. The lower frame area, provides a means for setting up the model as a server, so that the services are published. The process for creating the wrapper is shown on the right although this is intended to be transparent to the user.

The other important aspect of the **environmental** model is the impact of the integrated modeling approach on environmentally conscious design (Borland 1998). As for other attributes of the design, much of the product's environmental impact is determined early on in the design process. The number of dimensions requiring consideration and expertise required for analyses make the task difficult and time consuming for designers. The integrated modeling approach provides the means to evaluate designs as they are defined and provide real-time feedback to designers, without requiring the designers to conduct the analysis.

In a similar manner, the **customer** model provides designers with real-time feedback about anticipated customer responses. The ideal scenario would allow full-time interaction with customers to determine potential responses to different product attributes. Integrated modeling provides similar functionality through the connection of online customer models. These models include data about competitor products as well as utility models for product attributes of different market segments. This enables designers to receive real-time feedback about different product attributes. The other important aspect of the customer model, is feedback to other parts of the model. The manufacturing cost effects final sale price, which will effect sales volume, which in turn effects the estimated manufacturing cost. These scenarios can now be considered in the design process as a result of integrated modeling.

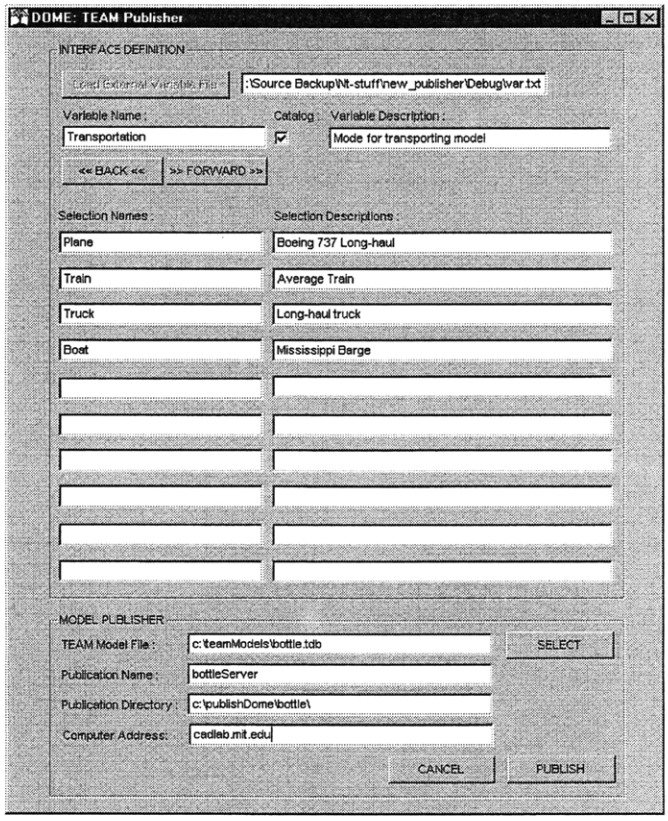


Figure 2-9 Publishing interface designed for the TEAM software. The diagram on the right shows the process whereby the isolated program is wrapped and made available to DOME. The objective is to minimize the overhead for making services available, therefore these helper programs are being written for different software applications.

A noticeable thread through this discussion has been the ability to flexibly evolve the model with any number of participants and software tools. While this is desirable it introduces model management issues in the form of activities such **tractability**, **debugging** and **maintenance** of the model structure. During the creation of this model, users routinely published their **interfaces** or **input/output** requirements. Although this was useful, it became clear that since there were specialists creating the sub-models, it was difficult for anyone to understand the intersection of the input/output requirements.

The limitation on *comprehending the integrated model structure*, made it difficult to predict what to do next or how best to allocate resource to create new pieces of the model. An analysis of the model structure was used to overcome this problem and became the seed for much of this thesis. This analysis will be explained in section 4.3.2. This need to understanding and evaluate *the emerging model structure*, leads to the discussion of **decomposition** in the following section.

3 DECOMPOSITION FOR INTEGRATED MODELING TOOLS

The previous section discussed the DOME framework for integrated modeling and demonstrated the need for methods to understand the structure of large distributed models as they are evolved dynamically. The creation of integrated models relies on two semi-distinct processes; *reducing a whole model into parts and connecting parts to form a whole model.* We refer to this activity as **decomposition**, a term which shares concepts with ideas such as chunking, perspectives, views, frames, morphs, granules or schemes from areas such as artificial intelligence and cognitive sciences.

In the following sections, different aspects of decomposition are discussed to clarify the concept and understand what is needed to support activities related to creation of integrated models. Examples are used to explore relationships between parts and wholes. This leads to the identification of structures known as **heterarchies**, which are variants of traditional **hierarchical** structures. **Object-based** approaches are introduced as a formalism to support the creation of these structures. Types of decomposition are identified and classified along the lines proposed by Bird and Kasper (1995). This leads to the identification of model properties that *can be used generally in integrated modeling for model structure comprehension and analysis.*

3.1.1 Relationships between the whole and parts

At the heart of decomposition are **relationships**. Relationships are important because it is the assumption of their absence that allows the subdivision of tasks or the breaking of systems into parts. At the same time, the need to connect different sub-models into a larger model, requires the creation of relationships between otherwise independent units. Two examples are used to identify different types of relationships: **sequential** and **functional**. The first is a function diagram for a robot "Builder" and the second is an excerpt from a Lego™ instruction kit. The examples show two very different applications, but clearly show the importance of relationships for understanding the result of the combination of parts.

The first diagram in

Figure 3-1 shows the instruction set for a "Builder"[6], illustrating three levels of capability that allow it to achieve different actions. The contrast between the parts on the left and the structure on the right emphasize how relationships facilitate understanding. The left to right organization introduces the first type of decomposition, which is a simple I/O (input/output) network. Each element performs a transformation from one state to another, such that in a sequence, the end-state of one step is the beginning state of another.

The second type of relationship is less clear, that is the grouping of sequences of transformations which form a "higher level" function, such as **FIND+GET+PUT = ADD**. The math for this is workable. Assuming a beginning and end state, the result of the application of **ADD** will have the same effect as the sequential transformations resulting from **FIND+GET+PUT**.

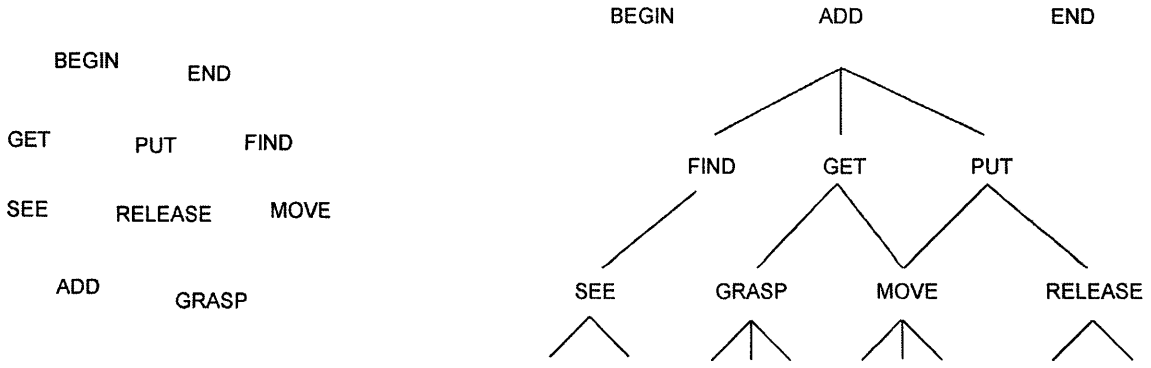


Figure 3-1 The role of structuring and relationships. The labels on the left yield little or no information about the whole, however the structure on the right uses functional and sequential decomposition to show how the actions work together.

[6] "Builder" is adapter from Society of Mind, M. Minsky 1986

The next example is intended to show how these relationships appear in the context of a simple example. The Lego components on the left in Figure 3-2 are "related" to produce the device on the right. We might ask a number of questions about the decomposition of this problem. Firstly, we can identify a number of possible serial relationships, such as the sequence of steps required to assemble the components, or a sequence of actions which relate the rotation of the motor to the motion of the arm as shown in Figure 3-3.

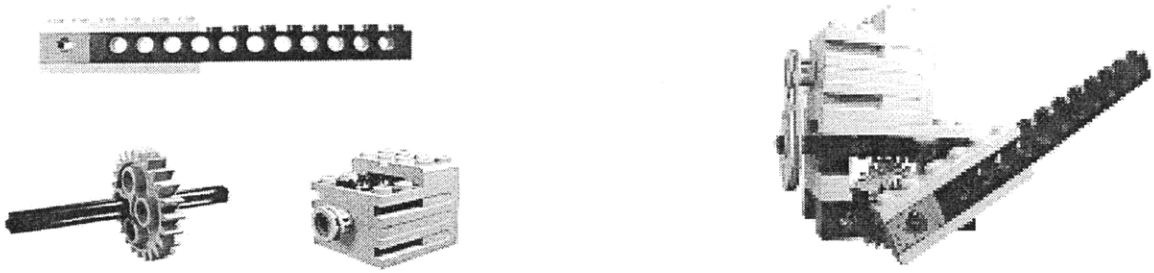


Figure 3-2 Relationships in a physical model. The components on the left can be associated with various functions. As unique components however, it is unlikely that anyone would identify the intended configuration on the right. The integrated view allows us to understand how these components combine to provide a larger function.

Functional relationships are less clear here. If we are mechanical engineers, we will recognize a **transmission** and a **transmission housing**. It is simple to show how elements of the transmission combine to produce a mapping of motor rotation to angular motion of the arm. It is not trivial to construct a similar diagram for the "housing". This demonstrates concepts defined in **Locus of Control** (Bechtel 1992), which define relationships allowing some parts to be easily identified or separated while others are more difficult as a result of the tight interaction.

The **Locus of Control** allows us to associate certain elements with behavior or function. For example, removing a gear from the drive-train will cause it to fail. Removal of parts of the housing are not guaranteed to have the same effect. We could break the housing down into artificial blocks and identify areas of different load bearing capacity as is the case in Finite Element Analysis (FEA). In the case of the transmission, the components are easily identifiable as independent physical parts. Not so in the case of the housing, the behavior is emergent. We have to uncover mechanisms at the material and geometric level, to understand it.

Figure 3-3 Inferring behavior as a result of understanding relationships. The identification of relationships between components provides a means to understand the system behavior. This allows us to predict the movement of the Lego crane if the motor is activated.

If we consider the creation of an integrated model for this Lego contraption, we might wish to relate the transmission and the housing. In particular, we might consider the reaction forces on the shafts of the transmission elements and relate that to the load bearing surfaces of the housing. How do we do this?

We will very likely have two distinct hierarchies; one for the transmission system and one for the housing (which includes material and geometric representations) and they are now connected by forces. Although the hierarchies remain, the new relationships that connect the branches of the hierarchies, result in a new structure called a **heterarchy**.

3.1.2 Heterarchies

A **heterarchy** is a representation structure in which some (but not all) of the parts can be organized into a **hierarchy**. The structure of this document can be derived based on the relationship between the different sections. If we consider each idea to be a node and the relationship between nodes to be arcs, we could construct a graph representing the input-output structure of concepts. Figure 3-4 shows an unstructured network and a heterarchy for the same set of nodes and relationships. The nodes in the network represent hyper-linked documents and the connections between them represent **hyper-links**.

If we consider the arrangement on the right in Figure 3-4, we can observe, what looks like a summary and then subsequent sections, where the hierarchy represents abstraction. Although there is feedback (arrows from higher detail back to more abstract levels), the hierarchy formalization remains visible. Hierarchies are useful for displaying structure, even if the hierarchy is only partially valid.

Another important aspect of these structures is the possible mixing of **sequential** and **functional** decompositions. If we imagine node ***a*** to be a summary which links to ***b,c,d*** and ***e***, we don't have a strict relationship that says ***b+c+d+e = a***, however there is some grouping with a summarizing relationship. This summary is in many ways similar to the idea of first order models. A complex representation can be reduced to a simpler one, by extracting the *critical* elements.

A further example of a heterarchy is a simple mechatronic system shown in Figure 3-5. This is a physical decomposition, where parts can be combined to form each higher level sub-assembly as in Figure 3-5 (a.). Figure 3-5 (b.) shows a heterarchy, which results when different perspectives are used to analyze the system. An electrical engineer, for example will consider the relationship between the **position sensor** and the engine's **electronic controller**. The designer of the accelerator sub-system, must consider the relationship between the **throttle cable** and **cam**.

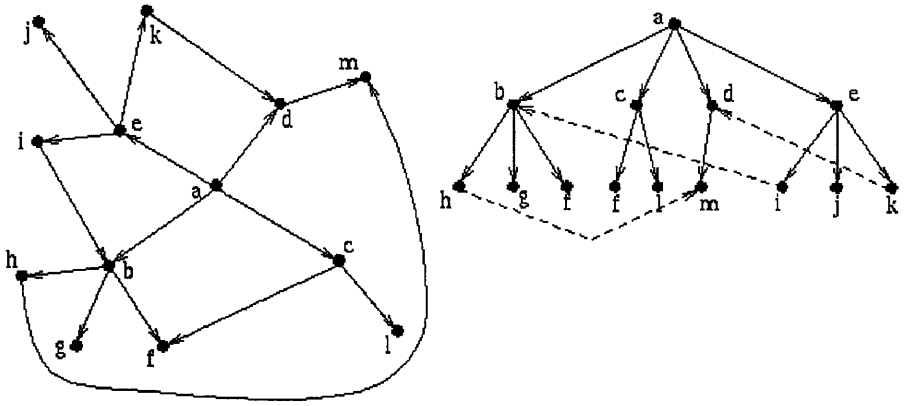
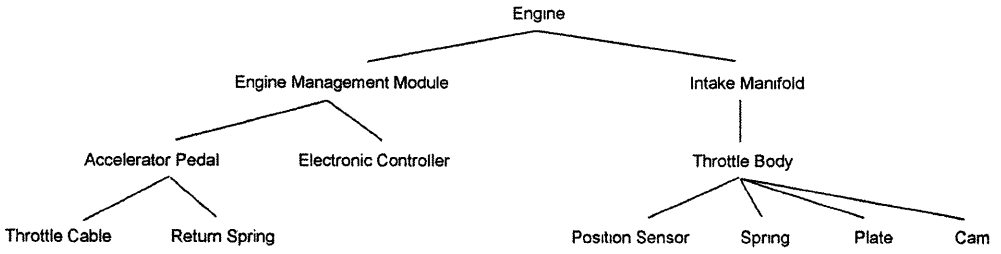
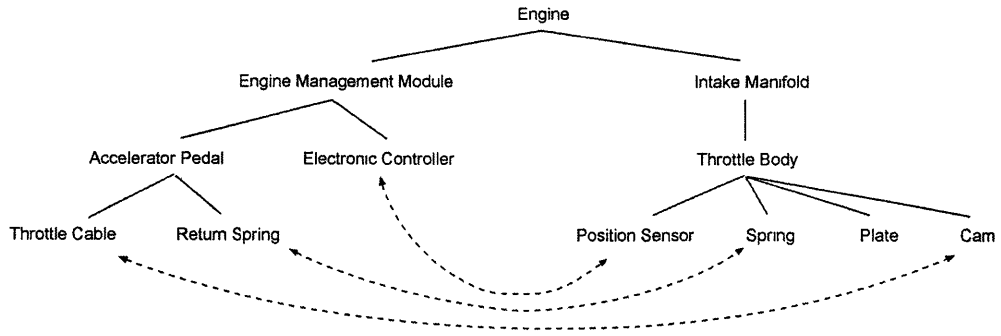


Figure 3-4 The partial structure imposed by a heterarchy. The nodes in the graph on the left are documents and the arcs are hyper-links. The figure on the right is a heterarchy, a structure that is for the most part a hierarchy, but for three exceptions.

When considering integrated modeling environments, similar heterarchical structures will emerge because of the manner in which people structure information. This can be thought of **intellectual** decomposition or **perspectives**.



(a.)



(b)

Figure 3-5 Engineering system heterarchy. (a.) The hierarchy shows a partial structure of an engine. The decomposition which is focused on assembly. (b.) Shows the interactions during design and operation, which result in a heterarchy.

3.1.3 Perspective and intellectual decomposition

Different types of decomposition are useful for different considerations. They can be thought of as filters, where certain information about a system is suppressed and other information in enhanced. The activity of modeling, is an example of such a filter. During modeling it becomes harder to define the nature of the relationships between elements since the particular expertise of the modeler will determine what is relevant and that which can be suppressed. The expert's knowledge will be used to construct or manage the relationships between the elements. This has been referred to as **Intellectual** decomposition by Kasper and Bird (Bird 1995).

Intellectual decomposition is the most difficult to generalize because the algorithms and representations represent the specialized knowledge for an individual or group. The intellectual domain will determine how a scope is defined (what information is suppressed). Further, the perspectives are somewhat unique to the domain. In fact it is argued that in problem solving, decomposition is achieved in very different ways for experts and novices, resulting in different level of success in solving the problem.

The throttle body example Figure 3-6 illustrates a mixture of relationships related to how different experts might choose to break down a problem. In the case of the overall hierarchy, it is not clear how this structure emerged. When an engineer with 20 years of experience in a manufacturing organization was asked to explain similar decomposition structures, he responded "It is historical!"[7].

[7] The comment was made by Shaun O'Reilly of the Advanced Manufacturing Technology Division at Ford Motor Corporation

The example in Figure 3-6 is used to demonstrate how a designer might decompose the system to reflect those elements that he or she is interested in. This is important if we consider the *myriad different decompositions, which will result in a collaborative project with multiple participants* in the modeling process. This example contains few elements. An existing model for the throttle body has more than 100 elements with the number of dependencies of the same order! This is not unlike the number of attributes and methods encountered in software programming. **Object-based** representations have been used to manage large-scale decompositions in software design, with some success (Stevens 1974).

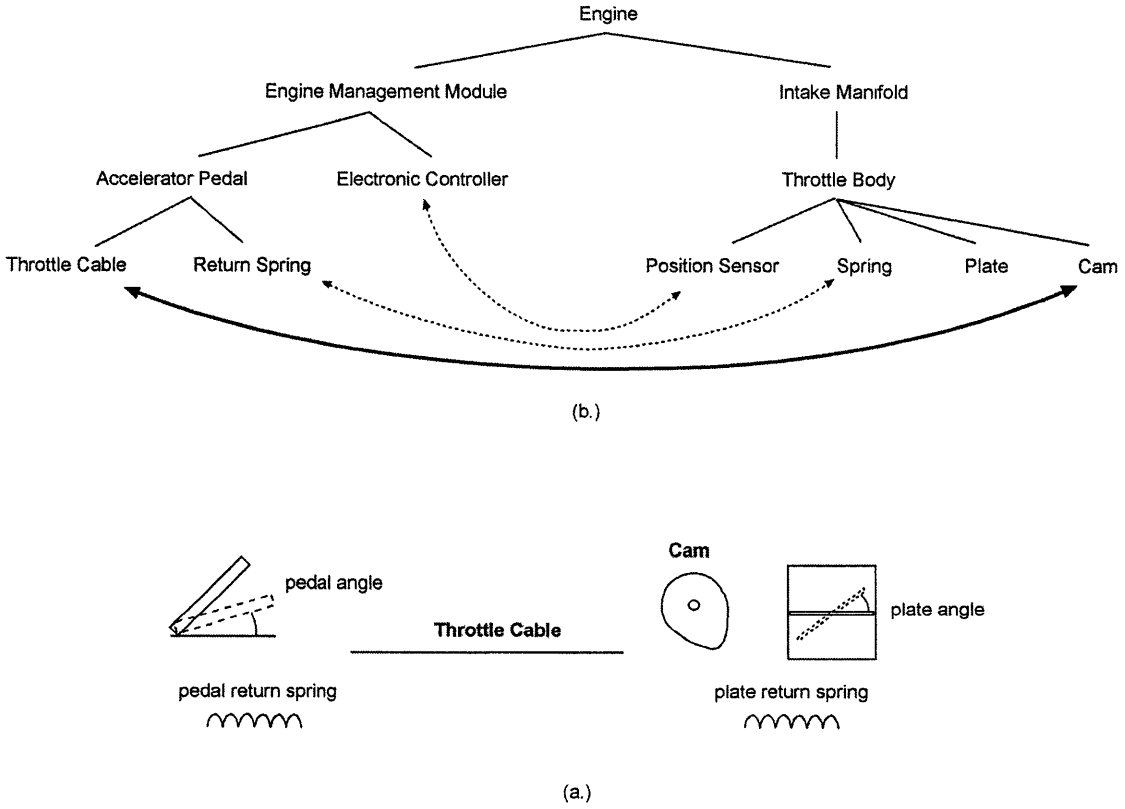


Figure 3-6 Intellectual decomposition of an throttle body subsystem. (a.) A heterarchy for the design of an engine system. (b.) Elements considered by the accelerator subsystem designer, which result in a relationship between the Cam and the Throttle Cable.

3.1.4 Object-based representations

Although much has been written about object-oriented approaches, some of the earliest work reveals most about the decomposition strategy (Stevens 1976). The focus was on **binding**, or the **relationships** that defined what should be placed in the same object structure. Initial work produced a list of binding or cohesiveness for software programs as shown in Figure 3-7.

Types of Cohesiveness	Brief Description
Coincidental	no identifiable relationship
Logical	Classification i.e. performing the same function
Temporal	Used in the same space of time - constructor methods
Communicational	Refer to similar data or have a section of code in common
Sequential	I/O relationship
Functional	A = f (B,C) therefore B and C are related by mutual effect in A

Figure 3-7 Binding types defined in early object-oriented work.

The methods and attributes, which the object implements, are grouped together. The relationship between objects is then limited to the manner in which they use methods and data from one another. Among other things, this structure has been shown to facilitate the following (Stevens 1976):

i. **Comprehension** of the program structure.
ii. **Maintenance** and **debugging** which depend on understanding, but also on arrangement of parts to create areas of high and low dependence.
iii. **Task allocation**, which is enabled through the definition of parts.
iv. **Code Reuse**, which becomes possible if objects are sufficiently abstract/general.

The object concept represents one of the best formal representations for decomposition. The advantages listed above are attributes of decomposition in general. While, it is not within the scope of this thesis, it is worth evaluating some of the formal aspects of object-oriented programming for the purpose of furthering understanding decomposition.

Some hierarchies in Object-oriented languages are intended to support aggregation of behavior, that is objects "inherit" abilities from one another (this can be thought of a functional relationship as defined previously). This is referred to as the class hierarchy and an example is shown in Figure 3-8 a. There are then a number of objects which are brought together to perform specific functions within a program. This is achieved by producing code that addresses increasingly specialized functionality, again representing a hierarchy as shown in Figure 3-8 b.

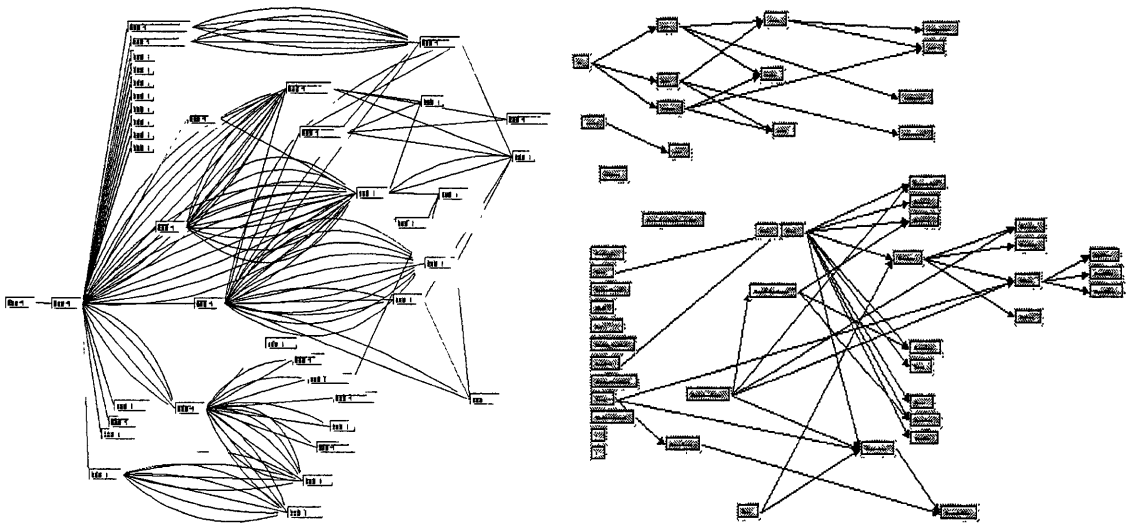


Figure 3-8 Object hierarchy and inter-class call structure[8]. The two images above are snapshots from object oriented code, where the rectangles represent different software objects. (a) On the left, the decomposition shows method calls between different classes. (b)The image on the right shows a class hierarchy, where arrows indicate inheritance of behavior.

[8] Structures obtained from analysis of a piece of source code used in DOME.

In object oriented software code, the structure used to guide the creation of objects may provide *little or no insight into the relationship between objects during the execution of the code*. In this case different views are required for different purposes although the same software code is considered, as demonstrated in Figure 3-8. Although the object structure provide a system for managing myriad perspectives and types of decomposition, there is no escaping the **heterarchies** which result from multiple different dependencies between elements in a given structure.

3.1.5 Summary of decomposition types

The definitions so far can be partially summarized in the following table, using the definition from Bird and Kasper. The four types of cohesion or relationships define types of decomposition: **Sequential, Functional, Informational** and **Intellectual.**

Type of Cohesion	Description
Sequential	Refers to the input/output type relationships as in the activities for "Builder", where the output/end of one step is the input/start of another
Functional	Characteristics of both "Builder" and crane examples where groups of functions or parts are combined to provide higher level functionality.
Informational	Third reflects the object oriented view of software, where code can be created using a procedural or non decomposed approach but is organized for a variety of management reasons
Intellectual	Defines the requisite body of information to solve a problem[9]

Figure 3-9 Classification of decomposition types. The table shows the types of decomposition as determined by the nature of information which is grouped together, beginning with explicit connection in Sequential to esoteric connections in Intellectual.

[9] This is defined as intellectual abilities required to perform a task, however in the context of the example it is adapted to mean the perspective used when deploying the relevant skills

Understanding the types of decomposition allows us to define mechanisms to enable them. The arrangement of decomposition types reflects the complexity of algorithms used to define the respective relationships. In the case of **sequential**, we can match input and outputs to synthesize structures. Synthesis based on these relationships has been achieved in a variety of ways for example PLANNER (Sacredoti 1977) or semantic parser used in Invention Machine Phenomenon (Invention Machine 1998). Functional relationships might also be amenable to analysis since in many cases they can be represented as groups of **sequential** transformations (Cagan 1998). The other decompositions become increasingly difficult to analyze.

Information and **intellectual** decomposition result in structures that are difficult to analyze because of the nature of the cohesion. The relationships do not lend themselves well to representation and when they do, the representations are likely to be esoteric. This may not present a problem for a single user, however when multiple participants are constructing pieces of the representation it becomes critical to find a mechanism for revealing some global structure, without requiring users to add more structure that they would need to represent the problem individually.

The **hyperlink** example demonstrates that although we cannot understand the reason for a structure, we can observe it through analysis of lower order structures such as I/O (hyperlinks) or **sequential** dependencies. This can help us recover some of the structure, which has not been formally represented. The underlying mechanisms in DOME produce sequential relationships, which can be analyzed to facilitate visualization and analysis of the structure of integrated models. This is the subject of the following section.

4 DECOMPOSITION IN THE DOME FRAMEWORK

This section describes the approach selected for supporting decomposition in the DOME framework. The first section outlines decomposition structures available to users for construction of models within the framework. The second describes the design and development of a prototype tool, which facilitates navigation of the decomposition structure through analysis of **sequential** dependencies.

4.1 Support for user defined decomposition in the DOME framework

The DOME framework is object-based; it provides structures, called **modules**, for grouping data and computation. The following section explains the module structure as well as the mechanisms for inter- and intra-module connections. These mechanisms allow each user to construct their desired representation and connect it to model elements defined by other participants.

4.1.1 Modules and mechanisms for creating dependencies

The basic construction unit in DOME is the module. For the purpose of this explanation we will focus on three types of modules, a **Real, Container** and **Relation**. The Real module is an example of the simplest type of module, which contains a Real number. As the name suggests, the Container may contain other modules such as the Real and Relation. The Relation allows modules to be related using mathematical statements.

The example in Figure 4-1a shows schematic representations of **Container** modules A and **B**. **C, D, E,F** and **G** are all **Real** modules and **A** also contains a relation module defining **E = D + C**. An additional property of modules is their ability to **alias** other modules. **Aliasing** allows a module (**G**) to become a clone of the *aliased* (**E**) module. This means that G will always maintain the same properties as E(thus creating a dependency), in this case since they are **Real** modules, this means a **Real** number.

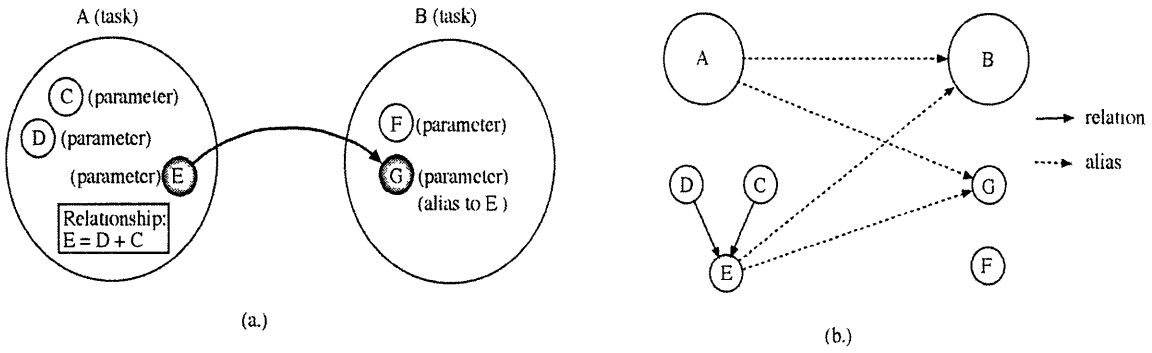


Figure 4-1 Creation of dependencies using aliases and relations. (a) shows container modules A and B which are connected by creating G in B which is an alias to E in A. The relation E = D + C is also added. (b) shows the directed graph structure of all the modules as a result of the alias and relation.

4.1.2 Current views of the model structure

Currently, the DOME modeling environment has a number of visualization tools, which facilitate *partial* understanding of decomposition. In much the same way as one might create a file-structure, different containers can be used to structure models. As a result the encapsulation structure can be viewed using a tree structure, where expansion of different levels allows movement through the parent-child relationships. In addition to this, each module provides a means to visualize it.

An example of such a module is the relationship module. At the lowest level, one can read expressions, which show relationships between different elements. This can also be viewed as a directed graph, showing dependencies that result from the creation of these relationships. Figure 4-2 shows a DOME model for components of an throttle body automotive sub-system. Although this provides insight into the model structure, it is not possible view the model through an alternative **dependency** structures, which make up the **heterarchical** structure.

4.1.3 The need for additional views of dependency structures

In section 3 different types of decomposition were discussed. At the one end of the spectrum are **sequential** relationships, as reflected by dependencies resulting from explicit connections such as mathematical expressions. At the other end of the spectrum are **intellectual** decompositions. This includes structures, which support specialized organization of data. In both cases, there *is no decomposition that is incorrect*, since at the lowest level of consideration, the model structure remains the same, explicit mappings and dependencies.

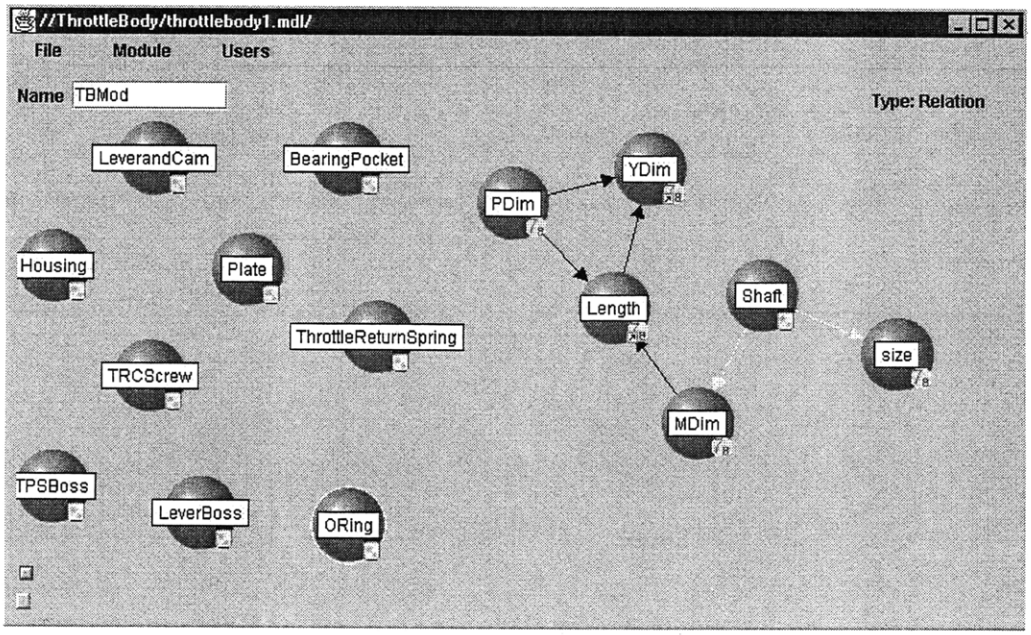


Figure 4-2 DOME graph dependency visualization for contained modules. Although the graph structure allows dependencies to easily followed, it works only within a given modules and only for a limited sized set of nodes and interconnections.

Although different intellectual decompositions are possible, the model structure or **sequential** relationships *remain consistent.* This phenomenon is discussed in "Discovering Complexity" (Bechtel 1992) who discuss decomposition in the context of approaches to research problems. Examples are cited where different decompositions were used to arrive at the same underlying model. This is important since the DOME environment is intended to support the collaborative construction of models. It is therefore necessary to identify structures that will be common between different participants, regardless of their individual choice of representation. This work attempts to use the **sequential** structure to support model visualization, as will be demonstrated in the following sections.

Viewing the structure of the integrated models is critical to understanding the relationships between elements. As demonstrated in Figure 4-1, the Object-based representation allows very flexible structuring of model elements. Current visualization in the DOME environment supports local views of the containment or **hierarchical** structure and the **relation** structures with input/output or **serial** connections. Because aliases are used to map between representations, the containment hierarchy does not completely reflect model structures, that result from interconnection of modules The result of aliasing are **heterarchies**, for which we need an alternative visualization method.

4.2 Visualization of decomposition structures

There are a number of ways to represent the interaction of elements in a network, the most conventional are the many flavors of graphs, such as *trees, directed or hypergraphs.* Although useful, graphs can quickly become unmanageable, particularly as the number of nodes N and the number of edges E increase as evidenced by earlier images such as Figure 3-8.

There are a variety of matrix-representations, which address this problem by allowing dense representations of dependencies. Although there are numerous variations in a variety of domains, the **Design Structure Matrix** is an example used in product development and will be used to illustrate how integrated model structures can be visualized. The limitations of current matrix visualizations are discussed and a visualization tool which addresses some of the shortcomings, will be introduced.

4.2.1 The Design Structure Matrix for Visualization

Alexander (Alexander 1964) proposed a method for creating diagrams representing the interaction of different elements of a design. This was in response to a recognition that the scope of modern systems are beyond the scope of the individual. These diagrams are intended to reflect the various components of design (and associated activities) and their interactions, as shown in Figure 4-3. Although Alexander discussed the method for obtaining the diagrams, after years of use, he commented that the power lay not in the methodology for creating the interactions, but in the study of the resulting network. Subsequently a number of visualization techniques have been proposed.

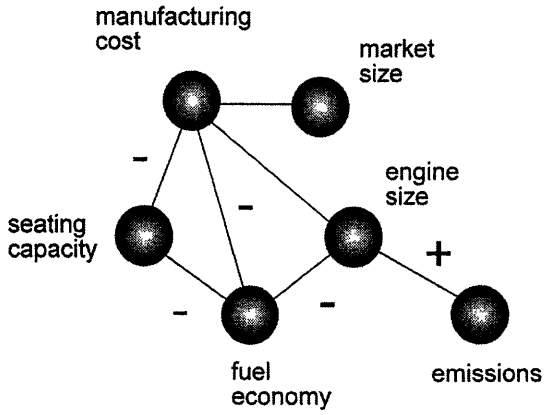


Figure 4-3 Alexander's model representation for an example automotive problem. The nodes represent elements under consideration, while the "-" and "+" signs indicate the types of interaction.

Following on Alexander's work, Steward introduced the Design Structure Matrix for information flow analysis. In its simplest form, the matrix can be compared to a directed graph, which is a traditional flow representation. The directed-graph, shows nodes and arrows representing flow or interaction between them. Similarly, the design structure matrix makes use of rows and columns, where (by convention) the columns represent source nodes and the rows, sinks. A comparison of the matrix representation and a directed graph is shown in Figure 4-4.

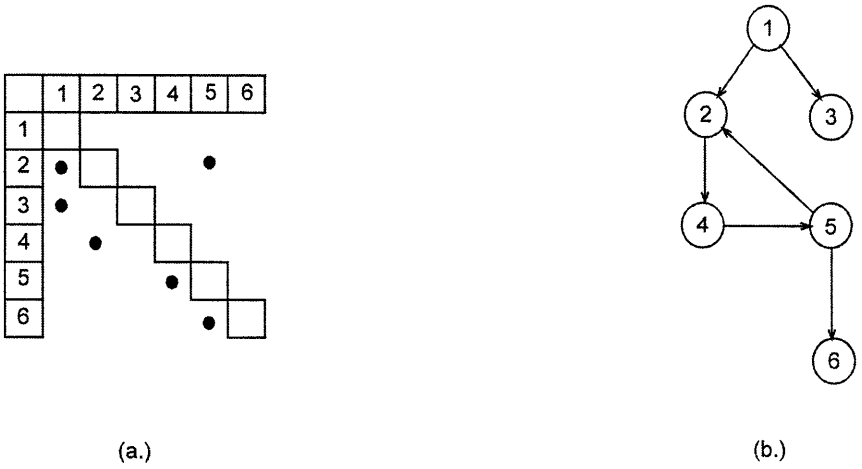


Figure 4-4 Comparison of a directed graph and a design structure matrix. The DSM allows compact representation. The symmetry allows feedback loops to be detected easily as in the case of nodes 5 and 2. Further, the ordering of rows can be used to illustrate possible node sequencing.

The DSM shown is the simplest matrix, known as **binary** matrix (Eppinger 1993-98) since dependencies are binary. The representation can be extended to support additional information for both the nodes and edges, allowing different interaction types to be considered. The design structure matrix approach has been used for **task** level descriptions as well as **parametric** descriptions. There are however a number of areas where current DSM visualization is limited, particularly for large models (100+ nodes), with a number of levels and large numbers of interconnections between modules.

4.2.2 Addressing Visualization Limitations of Design Structure Matrix

In an integrated model, the highest level view (abstraction) may consist of 10-100 containers and the lowest level (all detail) might be as many as 10000 or more. We would therefore like the ability to move seamlessly between these different levels of abstraction to understand emergent **heterarchical** structures. The problem can be framed in the context of the *dynamic range,* that is the level of detail, which can be adjusted by the viewer. We can estimate that we might wish to change the level of detail by 100-1000 times or more (change in order from top level to lowest level 10000/(10 or 100)).

In current representation schemes, the selected containment structure determines how the model is viewed. Although, this may appear irrelevant, it becomes important if one considers how the containment structures are determined. As mentioned in a previous example, the grouping of data or parts may be determined by historic manufacturing processes, that it everything is arranged by part structure. How important is this? That depends of the level of interaction between containers.

If we consider a component such as a throttle-body, it has been divided into 6 main parts. If we consider just basic geometric interactions, there are more than 20 interactions between these components. If we add other physical interactions such as thermal, the number rises again. This demonstrates the potential volume of information which moves between the defined structures and therefore the importance of making this information accessible.

An example is used to demonstrate how these types of interaction are handled in current Design Structure Matrix representations. Considering the alias, relation and containment mechanisms, we can examine how current DSM representations visualize interactions in models or processes. Figure 4-5 (a.) shows the hierarchical structure used previously in section 4.1. (b.) shows a "flattened" graph structure illustrating the types of dependencies between each of the module types. In this simple example, the multi-tiered DSM representation shown in (c.) and (d.), **50%** of the relationship information does not appear explicitly.

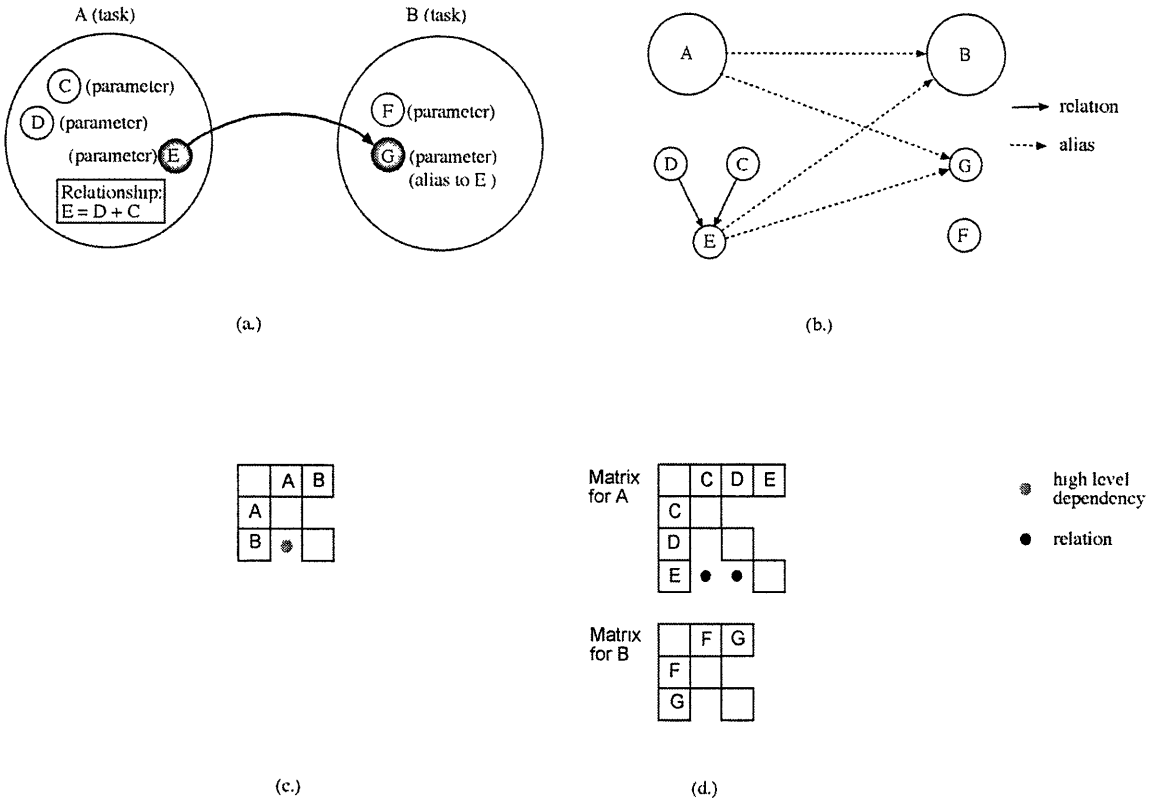


Figure 4-5 A design structure matrix view of the containers is presented in c and d. Although there are 6 types of direct and indirect relationships, a total of 3 are shown in the entire DSM representation. For example, there is no explicit connection between E and G indicated anywhere in the representation although we are made aware of some connection between A and B.

The same example shows how complete information can be represented, showing all types of relationships between modules. Figure 4-6 (c.) shows the most abstract view, demonstrating that there is an **abstract** dependency between A and B. This can then be expanded to view the source of these dependencies in A as shown in Figure 4-6(d.). Finally, a complete expansion allows us to view all the dependency types, providing access to the heterarchical structure.

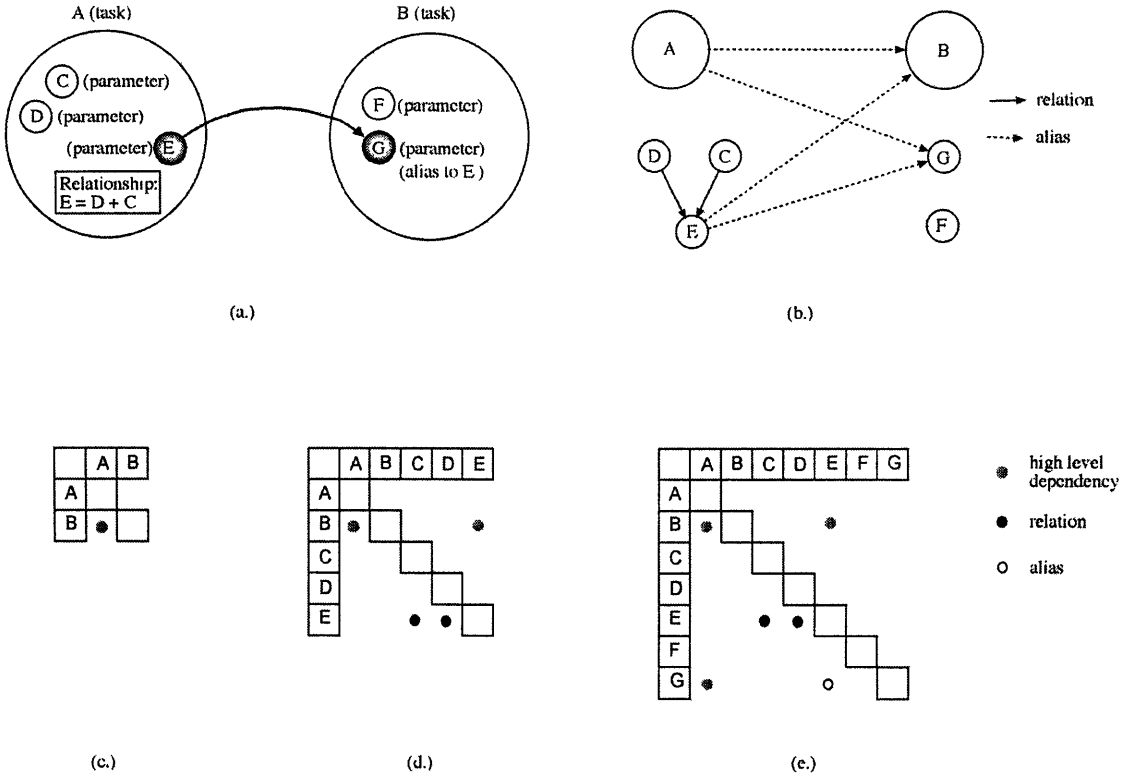


Figure 4-6 Matrix representation of containment, relations and aliases. The allows viewing of all types of dependencies, allowing movement between higher and lower levels of details. (a.) one representation of containment, relations and aliasing. (b.) "flattened" directed graph. (c.) a compact view of the relationship between A and B. (d.) a partially expanded view, with only A expanded and (e.) a fully expanded view, showing all direct and indirect or high level dependencies.

To summarize, a matrix-based tool to visualize heterarchical structures, should have a high dynamic range, allowing movement through multiple levels of abstraction as well as mixing of levels of abstraction. Further, distinctions should be made between different dependency types.

4.2.3 Description of the Visualization Tool

To address the requirements outlined in the previous section, a matrix viewer was developed. The objective of the viewer is to provide model views, which reveal all dependency types. To this end, there are currently two main views, an **encapsulated** and **ordered** view. The **encapsulated** view maintains the specified hierarchical structure, allowing navigation through the conventional tree interface. The **ordered** view "flattens" the hierarchical structure, interpreting containers as **hyper-nodes** as in Figure 4-6 (b.).

The viewer is divided into three main parts. An **overview** provides a complete view of the entire matrix using a 200x200 pixel display. The **main** view is closest to the conventional matrix views other than that it allows inline viewing of sub-matrices (i.e. matrices at different levels in the containment structure). The third and final view, is a **list** of all dependents for the currently selected element, providing the highest level of resolution.

The **overview** is intended to identify areas of interest within the scope of the highest level module. This allows identification of model features such as feedback connections or nodes, which provide or use high number of services. The **main** view allows exploration of features identified in the **overview**. In the **encapsulated** view, users can collapse or expand a tree-like view of the containment hierarchy, allowing movement between views such as (c) and (e) in Figure 4-6.

The **main** view is also the **main** navigation area, where particular types of dependency are identified by using different icons (presently abstract or alias). Users can then browse the areas by moving the currently viewed area to different positions in the **overview** matrix. The **overview** matrix provides an indication of what part of the overview is presently viewed in the main view. Users can find out more about elements of interest, by launching them from the containment structure.

The list view, maintains a list of all relevant nodes based on the current selected node. This is intended to provide fine-grained access to information about dependencies specific to the selected module. It will also indicate if there are elements which are beyond the scope of the matrix or unavailable because of limited access privileges. The following sequences demonstrate different capabilities, showing how the viewer is used in conjunction with other views and structures for creating DOME models.

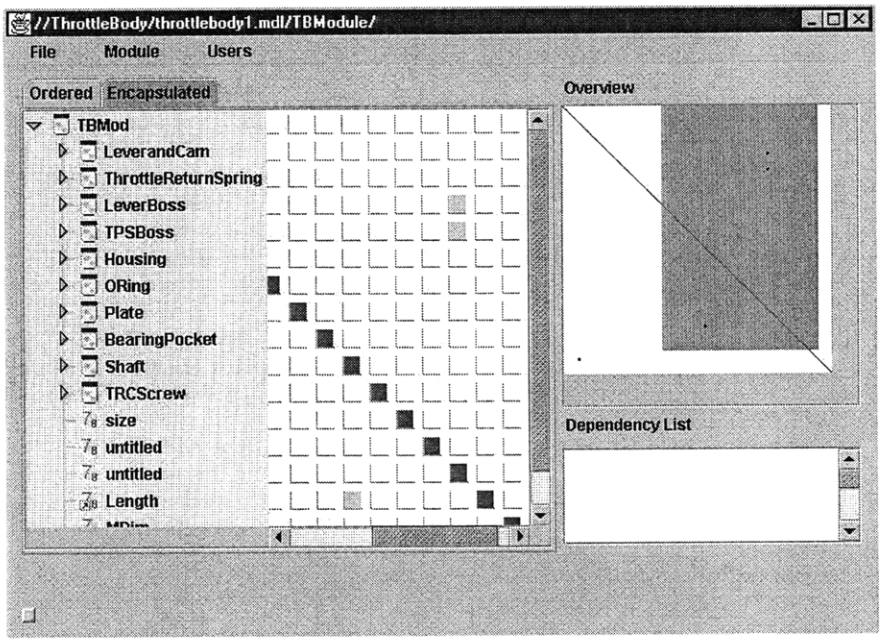


Figure 4-7 Image of visualization tool. On the left, jar icons are used to identify elements that contain other elements (which result in sub-matrices). The upper right view is the overview, which maintains the n square matrix and identifies possible areas of interest. Finally, the lower left shows an area used to display the names of dependencies for the current selected node.

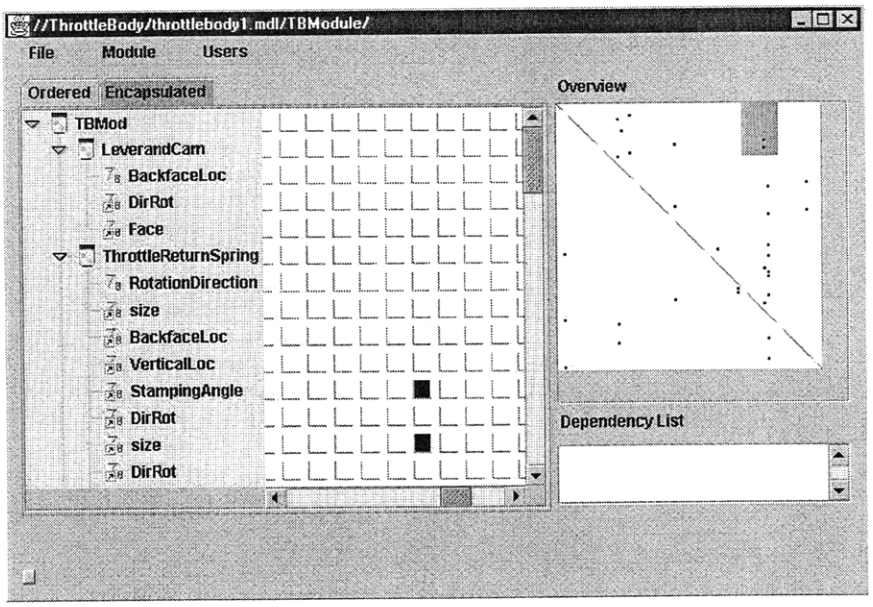


Figure 4-8 Expanded tree-view from Figure 4-7. Note the overall size of the model. The gray area represents the size of the current view in the main view.

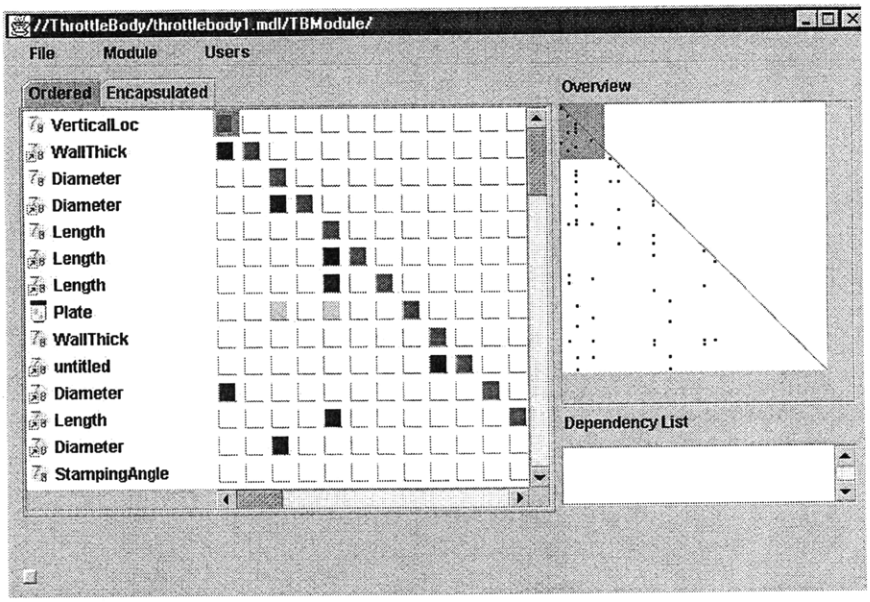


Figure 4-9 Ordered view of data from Figure 4-7 and Figure 4-8. The ordered view again shows the total model size in the overview areas as well as the lower triangular sort resulting from a graph analysis. The analysis will be discussed further in section 4.3.

The visualization capability is closely related to the analysis of DOME models. Although it is useful to view across multiple levels of abstraction, it is also a useful way to present results of analysis of the models. This analysis will be explained in the following section.

4.3 Decomposition through analysis of DOME models

Visualization is useful for understanding the dependency structures created in DOME, however analysis of the structure can provide additional feedback. Feedback may take the form of determining groups of highly interconnected nodes or finding optimal sequences to traverse a network. In the following sections, a brief overview of current approaches is presented. A graph theoretic approach is then introduced for the analysis of network structures, which have been created through dependency instantiations in the modeling environment.

4.3.1 Current approaches to decomposition

The objective of decomposition algorithms is to evaluate an **existing** graph structure in an effort to cluster or order the graph elements. In early product design work by Alexander, clustering was used to group nodes, which have *high interdependence, while minimizing interactions between the clusters*. The intention is to allow the resulting clusters to be optimized or completed somewhat independently. Thus the breaking of whole into parts ignores any existing structures such as **informational** or **intellectual** and only uses the structure to determine how best to organize the sub- units.

Examples of this type of analysis, which have been applied in product/process design include **genetic algorithm** or **rule based** systems such as those used in DEMAID[10] and DSM approaches (Eppinger 1993-1998) Tasks are represented as nodes with stated durations and edges reflect dependencies between tasks. A combination of rules and a genetic algorithm are used to search the space of possible sequences to determine shortest possible overall completion times. The model can be extended to include costs of various resources at nodes, adding yet another dimension to the selection of an appropriate ordering of node executions in the network.

[10] Information about DEMAID is available online at: http://www.hq.nasa.gov/hpcc/reports/annrpt97/accomps/cas_larc/WW96.html

The DEMAID tool was used as during the construction of the Polaroid model discussed in section 2.3. The image in Figure 4-10 shows a structure resulting from the analysis of the interfaces of various participants in the design process. The source file for the analysis is presented in Appendix A. The analysis revealed a high dependence on geometric data which was not yet being produced. The result was a shifting of resources to facilitate the creation of a geometric information to support the information needs.

###	Label
1	SPECS1
2	LE
3	VIDPCB
4	MAINPCB
5	VIDPCBPDN
6	SPEAKER
7	PWRSUP
8	PWRSUPPDN
9	SPEAKERPDN
10	MAINPCBPDN
11	EXFAN
12	CABLES
13	MASSPRP
14	CABLESPDN
15	EXFANPDN
16	LEPDN
17	ENVSCR
18	STYLE
19	FORM
20	ENVSPEC

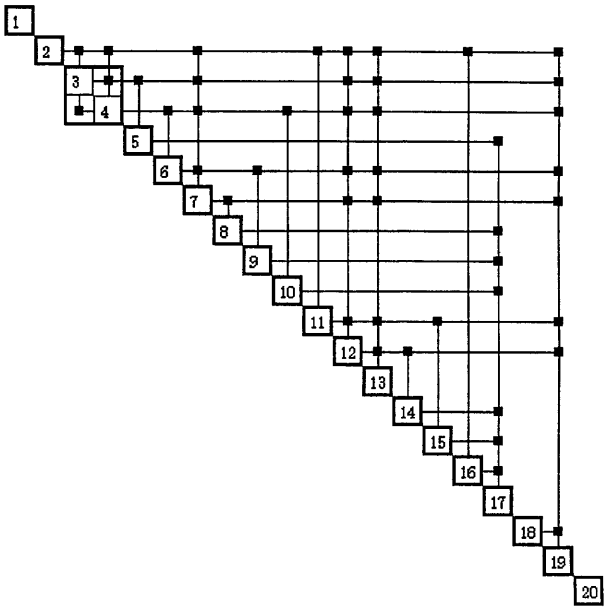


Figure 4-10 DEMAID Design Structure Matrix for the Polaroid project. The matrix above resulted form the analysis of the interdependence of interfaces published by the participants in the Polaroid project. The analysis revealed a need for geometric information which was not evident at the time.

4.3.2 Analysis algorithms for structural decomposition

As shown previously, DOME models can be viewed as directed graphs. These graphs are constructed by interpreting the DOME model. For example, **relations** determine dependencies through mathematical statements and **aliases** result in dependencies through explicitly mapping or referencing between modules. An important aspect of the analysis is **"flattening"**, first introduced in the visualization discussion.

Flattening occurs when the imposed hierarchy is reinterpreted as another type of dependency. Referring to the examples in Figure 4-6 (a.) and (b.), we can see that **A** and **B** contain other nodes. We can abstract **A** by giving it all the properties of it children, that is since one of it's children depends on another module, **G** in module **B**, **A** will assume this properties. Generally then, flattening refers to the process whereby **container** modules take on the **union** of all properties of their children.

Once the model is in the form of a directed graph, a number of well developed methods exist for graph analysis such as those discussed in the previous section. In the most basic form, algorithms make use of structures illustrating **nodes** and the **edges** or dependencies. In more complex analysis, properties will be associated with **nodes** and **edges**, allowing more complex analysis, such as weights.

One of the objectives of this work was to limit the information required to analyze the decomposition structure. The main reason was to allow users to create models to represent the design object, without any additional overhead. Overhead here is defined as the addition of information which does not relate directly to the creation of the design object. The current implemented algorithms do not consider any properties, only nodes and directed-edges, which result from the creation of models. Possible approaches are discussed further in "Future Work" in section 5.4.

The depth first search algorithm is used to build a graph by traversing a selected DOME model or model segment. The result of the depth first search is an ordered set which can then be re-searched. The results of the second search can be used to group nodes that are part of **circular dependencies**, effectively producing a hyper-graph, clustering nodes according to higher interdependence. Pseudo code for these algorithms is provided in Appendix B.

4.4 Overview of DSM tool architecture

The following section gives an overview of the architecture for the tool as implemented in the DOME environment. Figure 4-11 shows the system architecture, including details of the general DOME architecture and software implementation (a more comprehensive explanation is included in Appendix C). The implementation consists of the DSM plug-in and user interface, which make use of infrastructure made available by the DOME Engine and DOME Java-based Distributed Computing Components.

The DOME Engine contains a model as created by the users of the DOME environment. The DSM Module generates the desired directed graph structure by querying the selected model or model segment. It also contains the functions for conduction the clustering algorithms, explained in the previous section. The Distributed Computing Components provide infrastructure for messaging between the plug-in and the user interface. The plug-in and user interface make use this to ensure that displayed information reflects the current state of the model. Finally, the user interface was shown in Figure 4-7.

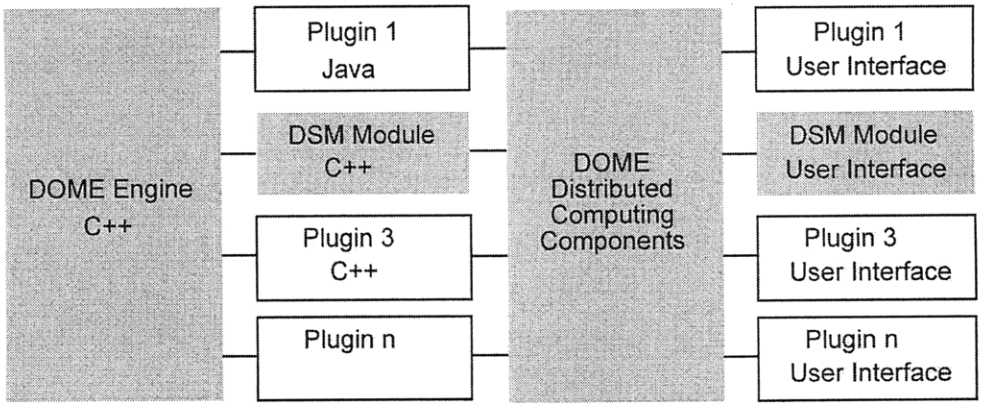


Figure 4-11 The DSM tool as part of the DOME architecture. The DSM tool is designed as a plug-in to the DOME Engine. The plug-in is used to build and analyze the directed graph structures. The DOME Distributed Computing Components provide capability necessary to interact with the DOME engine in distributed environments. The user interface allows visualization and interaction with the DSM Tool as well as interaction with other DOME components.

5 DECOMPOSITION TO SUPPORT RESOURCE ALLOCATION

Ultimately, the goal of providing decomposition tools, is to allow participants to better manage the creation of a product. The problem is one of **coordination and resource allocation**. Massimo Egidi (Egidi 1992) uses theories of problem solving to explore the relationship between coordination and resource allocation in organizations.

A conventional marketplace for buying and selling goods or services is one that coordinates tasks, where coordination is used in the economic sense to reflect the matching of buyers and sellers. Organizations perform this coordination, however they also perform the division of labor or more generally resource allocation to determine what will be done to satisfy the coordination requirements.

Resource allocation techniques are available from a number of fields from operations management to parallel processing, however the challenge is unique in the product development process (and therefore the creation of integrated models). The reasons for this are related to the difficulties in obtaining accurate representations of the design process and therefore an understanding trade-offs associated with fulfilling different coordination requests. Analysis and visualization of integrated models provides a method for obtaining this information.

We can treat the product development process as a marketplace. If we have a model of the current state of the object being designed as well as the various requests for resources, we can evaluate the model structure to determine request priority. A method is proposed for conducting this type of analysis using the DSM tool discussed in the previous section.

5.1.1 Current difficulties in constructing representations of the design process

Although there are candidate methods for evaluation of networks, two challenges have been identified for creating representations of the design process. The first is the sampling frequency or how well the method tracks the evolution of the network structure over time. The second is the level of detail, which is captured or conversely how much information is lost in the representation. Further, these methods require explicit actions by participants in the process being considered, introducing additional overhead as in the case of the evaluation of the Polaroid model.

Sampling frequency determines how often the current state of the network can be represented; for example new nodes or interactions are added to the network and we wish to consider their respective implications. The **Sampling Theorem** suggests a desirable sampling frequency, the **Nyquist** frequency, of at least twice the highest frequency of the signal which we wish to sample (Franklin 1994). If we consider changes to the network structure to be the signal that we are analyzing, we require a method, capable of measuring these changes. If, for example, we assume that the network might be modified on the order of hourly or daily we need to sample on the order of 30min to twice per day.

Resolution refers to the degree to which the representation captures the actual network; the level of detail which is represented. Level of detail expresses a difference between representations such as **task-based** and **parameter-based** descriptions. The difference is revealed in the degree to which each representation captures the richness of the process. The difficulties associated with data collection are varied, including overhead, identifying pertinent information, as well as defining an appropriate representation scheme.

Interviewing participants within a designated scope of a design process, is a common approach to data collection. The impact on sampling frequency is a function of the number of interviews and the rate at which these interviews can be conducted. Frequencies on the order of months or weeks are common (Dong 1998), therefore much of the dynamics of the process is lost as shown in Figure 5-1. In addition, there is a resolution issue, which is dependent on the level of detail of questioning. For example, problems are addressed at the level of parameter or tasks.

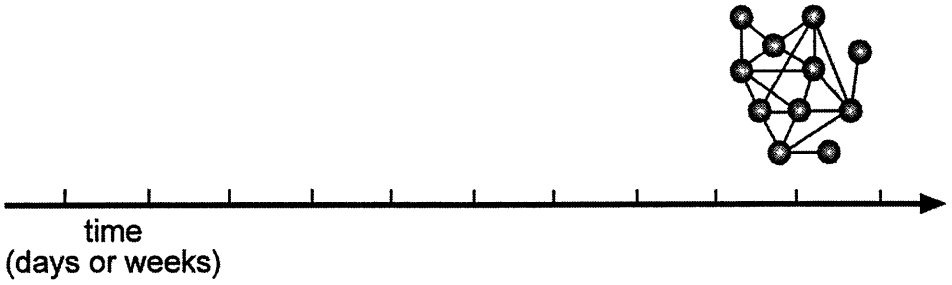
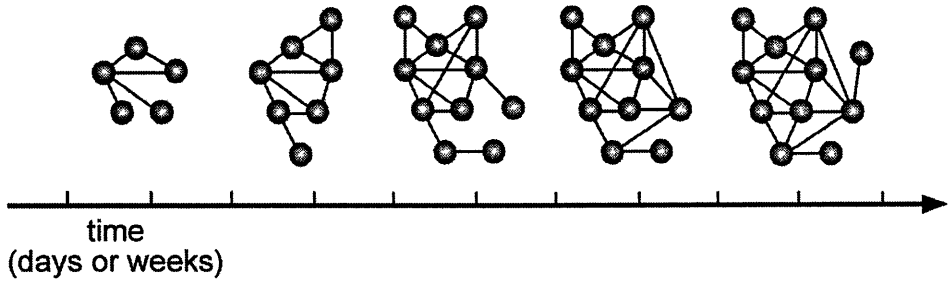


Figure 5-1 Comparative sampling frequencies of the design network. The image above and below show the information available for different sampling frequencies. The top view shows current methods, which allow sampling well *below* the Nyquist frequency. The proposed method allows sampling well above the Nyquist frequency. This ensures that the dynamics of the network formation and therefore the design process , can be captured.



At the task level information might include an entity such as "collect customer requirements", while a parameter focused version might include "mean time before failure", "weight", "cost" etc. The diagram below reflects the idea of resolution in design process representations. The model suffers loss of information in the process of creating a representation of the process as practiced. Further, translation by observers often causes further losses for example, interviewing. There are a number of dimensions to this problem, such as overhead or identifying appropriate information to represent.

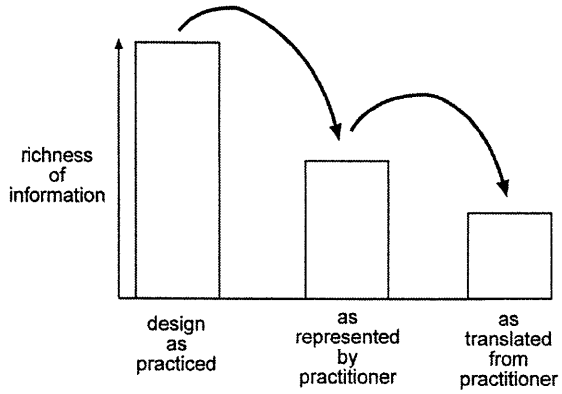


Figure 5-2 Reduction in resolution during collection of design process information. The actual design process is filtered as it is translated into a process model. Because the methods for creating process models rely on interviewing or translation, the resulting process model differs greatly from that practiced.

Currently, resolution issues are being addressed through the development of methods to speed the data collection process, thereby improving the potential sampling frequency. These methods use methods to allow participants to participate asynchronously on a common model to create a network representation. Although this may reduce frequency issues associated with interview method, participants are still required to explicitly update the model as they make changes, introducing additional overhead.

Current approaches require resources from the participants in the design process. These requirements increase in proportion to both the frequency and resolution. This means that process management will result in a trade-off between allocating resources to the design process versus allocating resources to provide information about the process. The focus of the proposed method is to effectively reduce these trade-offs by extracting process data from the activities of those participating in the design process.

5.1.2 Using decomposition for real-time resource allocation

The integrated model is created, by directing the growth of an evolving network of service exchanges. Evolution can be described as the addition of new interconnections or nodes using the mechanisms such as **aliasing** or **relations** described previously. This evolution is possible because of a **service marketplace**, where participants in the modeling environment can publish, connect and request services as described in the Polaroid case study.

In the example in Figure 5-3, each node has ***"Provided Services", "Currently Subscribed Services"*** and ***"Requested Services"***. The "Provided Services" reflect service sellers and the "Requested Service" are the buyers. The "Currently Subscribed Services" illustrate the existence of connections, forming an existing network. In some cases, coordination is possible by matching the buyers and sellers. In other cases however a decision must be made about how to deploy resources in order to fulfill service requests.

Consider the example of the transition between the state in Figure 5-3 (a.) and (b.). The **Marketing Consultant** has requested dimensions. The **Geometric Modeler** has a service called **Form**, which contains these. The service request can therefore be satisfied with the current resources. The Geometric Modeler has requested a **Component List**, that is not currently available in this scope. The participants must therefore decide how to get this information. This leads to a resource allocation problem as this information may exist or need to be generated.

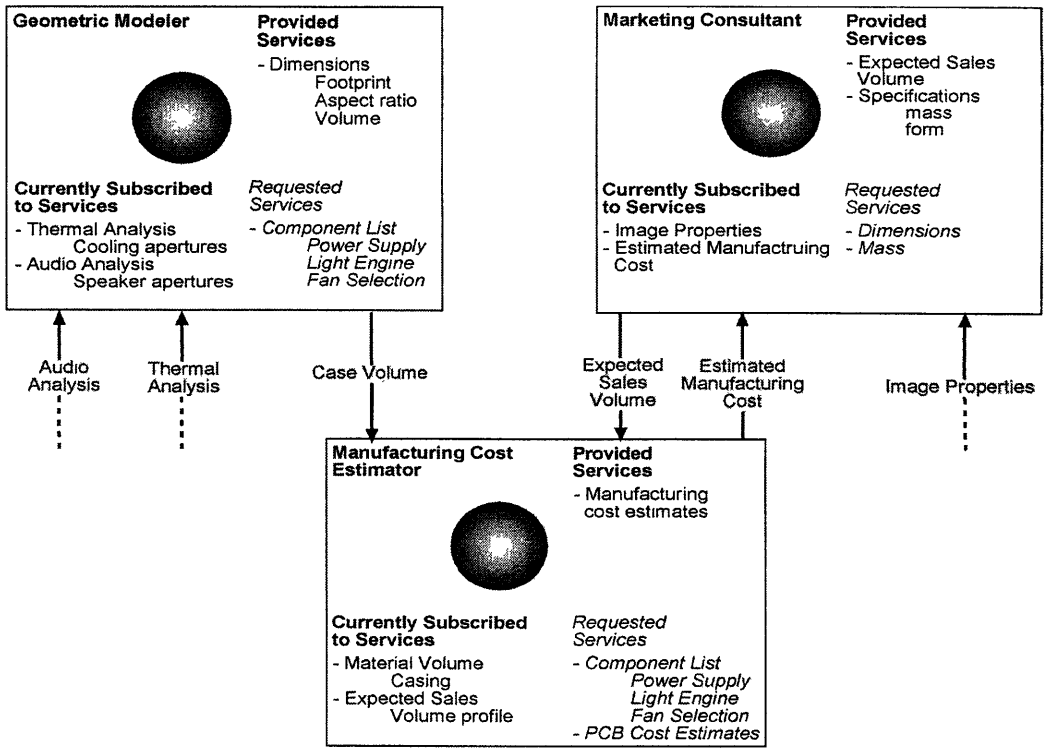
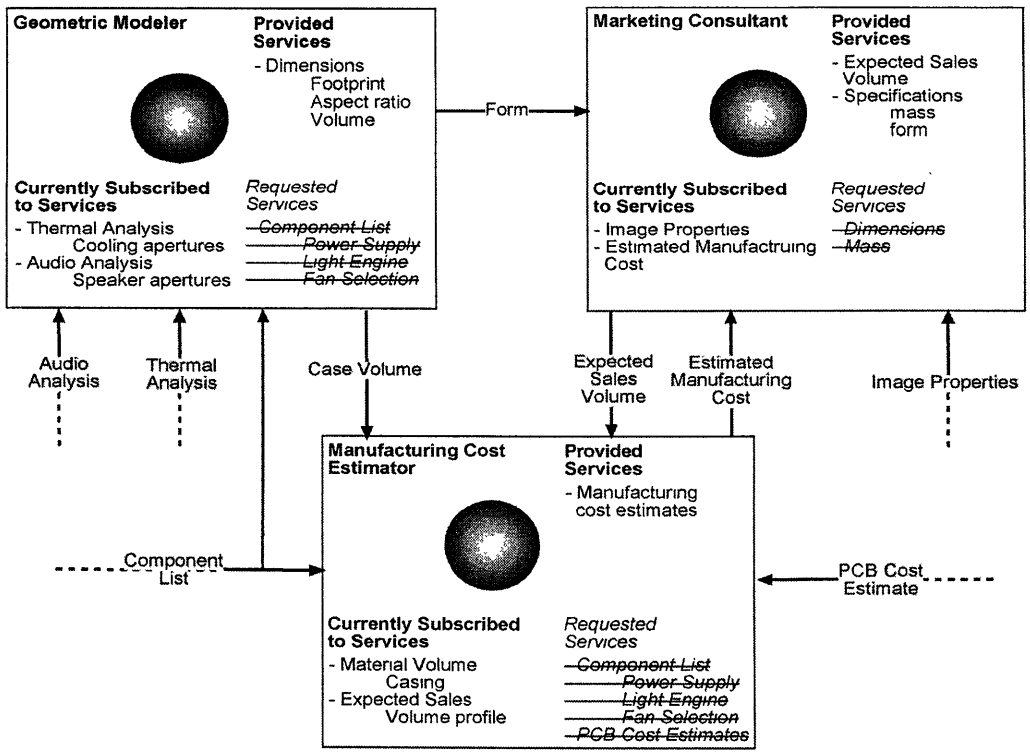


Figure 5-3 Evolution of the service marketplace. (a.) Shows the initial state of the service marketplace and (b.) shows the subsequent state after additional service connections have been made.



5.1.3 Resource allocation using solution space searches

The expansion of the network now requires the solution of a resource allocation problem. DOME provides a number of alternatives. When the model is at a point where it can be run, feedback will provide some insight into desirable directions. Ullman et al (Ullman 1997) explore three main questions posed by decision makers in the engineering design process:

4. *What is the best alternative?*
5. *Do we know enough to make a decision yet?*
6. *What do we need to do next to feel confident about our decision?*

The purpose of a "completed" DOME model is to address these questions, using decision theoretic tools. The completed model however is dependent on a complete structure. This feedback is made available by algorithms, which search the solution space and make suggestions about more attractive solution. Based on this information subsequent decisions can be made. Methodologies are needed when the model is under construction and cannot be evaluated.

5.1.4 Resource allocation by evaluating the network structure

The evaluation of networks, resulting from integrated model structures can assist in resource allocation decisions for the coordination of service requests from the service marketplace. This will provide additional mechanisms to support the evolution of the integrated modeling network.

An evolution step is portrayed in Figure 5-4. (1.) An integrated model is in a state of partial completeness, some service connections have been completed, while others remain unfulfilled or unused. (2.)The service marketplace facilitates matching of service **buyers** and **sellers**. (3.) Two requests remain outstanding from nodes **E** and **B**. We are now forced to make a decision about where to expend resource to fulfill these requests. In a one dimensional evaluation we can now consider the **structural** importance of the respective nodes. This is determined by analyzing the graph structure and ordering nodes according to their **role** in the network. (4.) The result is that **B** is deemed to be "more important" based on its position in the network.

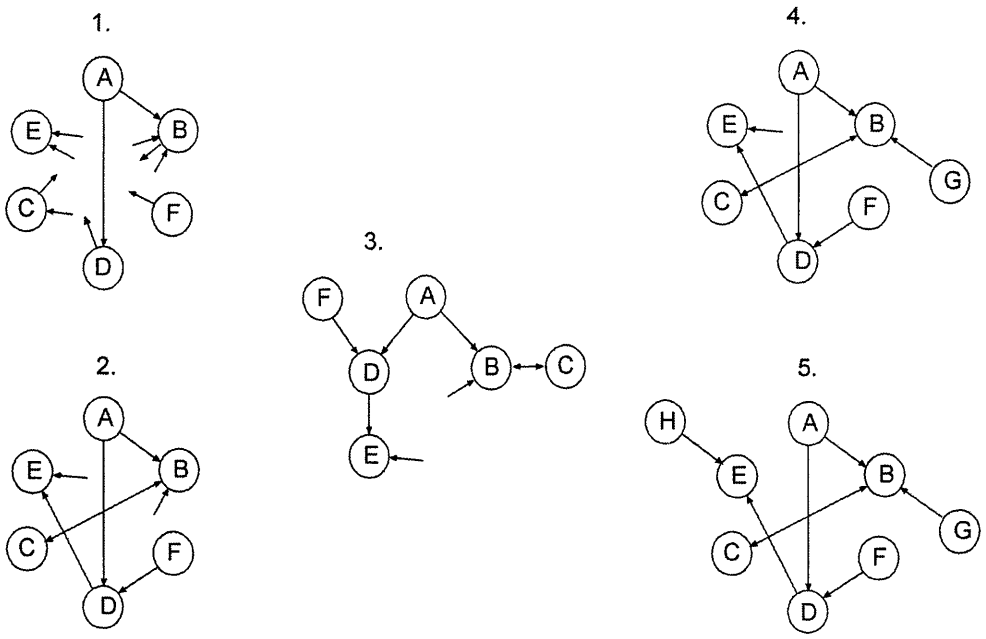


Figure 5-4 The process of resource allocation through network analysis. The sequence above illustrates the union of coordination and resource allocation. The first step shows groups of nodes requesting and providing services. The next shows partial connection of these services, leaving two requests unfulfilled. The third step uses graph analysis to prioritize nodes, illustrating the priority of nodes in the process. This priority is used to allocate resources to add node G in step 4. Finally node H is added as the requirements of E are deemed to be relatively important. Subsequent service requests will cause further iterations of this process.

The principal advantage of this approach is that the process information can be obtained directly by evaluating the integrated modeling. As explained earlier, currently, process analysis is often neither timely nor of a sufficient level of detail. The example of the Polaroid evaluation was possible only because a service marketplace was constructed and the network could be derived from the service interfaces. Although there was overhead associated with the evaluation of the network, however the utility of the approach was demonstrated.

CONCLUSIONS

5.2 Summary

The thesis has introduced decomposition in the context of integrated modeling for product design and development. General approaches to integrated modeling were introduced and the DOME framework was explained in the context of a case study. Ideas about representation and decomposition were introduced and classified. A distinction was made between user defined structures (informational and intellectual) and invariant structures (sequential) which reflected the underlying models.

Mechanisms supporting decomposition in the DOME environment were then introduced, leading to the introduction of the need for alternative visualization schemes. A prototype tool was described which provides support for the representation of the expected heterarchical structures. Analysis of model structures was introduced as a further means of obtaining feedback about integrated models through evaluation of the resulting network structure, independent of any intellectual or informational decomposition schemes. An example analysis from an industrial project was used to demonstrate the potential value of a dependency network analysis.

The challenges of design process management were then discussed, revealing limitations of current process evaluations; high overhead, low sampling frequency and low resolution. An approach was proposed to assist in coordination and allocation of design process resources. The approach makes use of the visualization and analysis tool to provide information about the relative "dependence" of elements of the integrated model. The process evaluation is possible without any additional overhead, based exclusively on information contained in the integrated model. The information in available in real-time, as the integrated-model evolves.

5.3 Limitations of the approach

The strength of the approach is also likely to be its greatest limitation. The approach depends on the creation of integrated models, which accurately reflect the design of the product under consideration. Accuracy here may be defined as the degree to which the model reflects the current considerations. This assumes a scope and timeliness not yet achieved in any design representation scheme. While DOME employs a variety of methods to facilitate "accuracy", the possible success of the approach is directly coupled to the availability of models created from a marketplace of objects exchanging services, as is the case for DOME.

Although the visualization and analysis tool is currently limited to a subset of dependencies (including aliases but not master/slave), extension should not prove difficult as the underlying mechanisms are in place. The dependency structure will be extended shortly after the completion of this thesis, providing a more complete representation. Although the implemented analysis algorithms provide solution to the problem of ordering, there is currently no support for clustering. That is, organization of model into sub-models with respect to multiple criteria. This is *critical* and will be discussed further in future work.

In addition to the limitations of the current implementation, there are also a number of additional infrastructure requirements to implement the approach. A **service marketplace** will be required to coordinate users of the system. Currently, there is support in the form of informal collaboration. However since the tool is intended for projects, large in scale and scope, the current mechanism will not be adequate. This is also discussed in future work.

Finally, although it is possible to evaluate the tools with respect to the stated objectives, it remains to be seen how this approach impacts the process of design. It is difficult to evaluate performance without data obtained from use on industrial problems. Initial use of the tool in industrial settings should provide valuable feedback about strengths and weaknesses.

5.4 Future Work

The previous section outlined a number of potential areas for future work. This section addresses these. The subsections are arranged from incremental extensions to completely new areas.

5.4.1 Decomposition to support model navigation

Currently the model scope or size is managed by encapsulation or intellectual or informational decomposition. While this is often effective, the selected groups become esoteric when participants have different backgrounds. While the DSM visualization addresses some of these issues there are a number of alternatives, which appear promising.

Work has been undertaken in several fields, including software or data visualization. Browsing hyper-linked documents is in many ways similar to browsing integrated models since a number of users will construct and manage groups of pages. A number of different visualization schemes have been explored to explicitly address issues such as maintenance, debugging and navigation of links.

Narcissus (Mukherjea 1994) is a generic program for visualizing large volumes of information. In this approach rules are used to analyze relationships which are then used to group modules. In this way, information can be hidden at different levels. The system is implemented in a three-dimensional scheme using web pages and analysis of their hyperlink structures. This is decomposition based on inter-relationships between objects/documents or structural decomposition as used in the DSM visualization.

Another visualization scheme, is galaxy, undertaken as part of the Multidimensional Visualization and Advanced Browsing (MVAB). In the galaxy approach documents are classified according to their "similarity" (which might be defined by any number of algorithms). When combined with temporal information such as dates of creation or editing, other views are possible. This is decomposition or classification is based on content. This is also promising since DOME increasingly includes meta-data such as users, creation dates, modification dates, types and documentation, all of which can be analyzed.

5.4.2 Analysis to suggest alternative model decompositions

There are a number of possible useful extensions. Currently, the criteria for decomposition are determined by the implemented algorithm; implicit in the algorithm is that structure is the only part of the decomposition which is evaluated. The objective is to develop a sequence of nodes, minimizing feedback. The decomposition should include multiple criteria such as measures of complexity, measuring different attributes of the resulting decomposition. For example, if an element is to be outsourced or a project involves a number of different groups, algorithms could be used to divide a model in such a way as to minimize the size of the interface in an attempt to reduce interaction.

Part of the problem remains to identify the desired criteria for evaluating decompositions, which remain uncertain. In general though it appears that reduction in coupling promises speed gains, while increased integration may yield possible quality improvements. Once the criteria are defined however, the DOME genetic algorithm or another optimization scheme such as simulated annealing might be used to evaluate alternative decomposition.

5.4.3 Formalization of the service marketplace

Currently, the service marketplace is not represented in any way and is not connected to the DOME environment. A complete service marketplace would provide a means to coordinate "buyers" and "sellers" of services. Possible approaches are as varied as the number of possible markets and associated mechanisms. Some ideas include integration of groupware. This might be used to notify participants in a project about new services or service requests. Mechanisms for searching for modules and services would also be useful. There is interesting work taking place in the areas of ontologies and agent interaction (UMBC 1999), which might prove useful for more abstract representations such as documentation. Further ideas such as collaborative filtering might be used in conjunction with user profiling to attempt to project what users might be interested in based on what they are attempting to achieve.

APPENDIX A

This appendix contains source files for the DEMAID evaluation of the Polaroid Project. DEMAID is managed by Jim Rogers who can be contacted at j.l.rogers@larc.nasa.gov. No version number was found for the release used. The system makes use of a system of rules to generate alternative DSM structures, which are then evaluated using a Genetic Algorithm.

Below is a source file, which contains a list of modules from the Polaroid example discussed in Section 2.3 with an associated list of dependencies. From left to right each line containes the module number and name. The O and 1 represent expected task duration and cost respectively and were not used in this evaluation. This list following "uk" contains dependencies (i.e. other modules which depend on this module).

```
//Tina (voice of customer)

//specifications
(module  1 SPECS1 0 1 SPECS uk)

//enclosure evaluation

(module  2 DIMS1 0 1 DIMS1 uk SPECS FORM)

//audio evaluation

(module  3 AUD1 0 1 AUD1 uk SPECS SPEAKER)

//ergo evaluation

(module  4 ERGO1 0 1 ERGO1 uk SPECS MASSPRP)

//picture evaluation

(module 5 PICT1 0 1 PICT1 uk SPECS LE)

//Tom (general modelling)

//following is a list of tasks which require the generation of catalog
//and then selection of individuals from the catalogs

(module  6 LE 0 1 LE uk)
(module  7 VIDPCB 0 1 VIDPCB uk LE MAINPCB)

main PCB is assumed to include the remote sense PCB

(module  8 MAINPCB 0 1 MAINPCB uk LE VIDPCB)
```

```
 (module  9 PWRSUP 0 1 PWRSUP uk LE MAINPCB VIDPCB SPEAKER )
 (module  10 EXFAN 0 1 EXFAN uk LE )
 (module  11 SPEAKER 0 1 SPEAKER uk MAINPCB)
 (module  12 CABLES 0 1 CABLES uk LE VIDPCB MAINPCB PWRSUP EXFAN
SPEAKER)

 //Shaun (geometric modeling)

 //following is a list of characteristics resulting from the above
//selections which are fed back to evaluations in
 /the first few modules

 (module 13 STYLE 0 1 STYLE uk)
 (module 14 FORM 0 1 FORM uk LE VIDPCB MAINPCB PWRSUP EXFAN SPEAKER
CABLES STYLE)
 (module 15 MASSPRP 0 1 MASSPRP uk LE VIDPCB MAINPCB PWRSUP EXFAN
SPEAKER CABLES)


 //Ines (LCA analysis)
 //the following is a list of components from which various parameters
//are required such as material mass. etc
 //the parameters will remain the same however the selection will
//change depending on catalog item used


 (module 16 LEPDN 0 1 LEDPN uk LE)
 (module 17 VIDPCBPDN 0 1 VIDPCBPDN uk VIDPCB)
 (module 18 MAINPCBPDN 0 1 MAINPCBPDN uk MAINPCB)
 (module 19 PWRSUPPDN 0 1 PWRSUPPDN uk PWRSUP)
 (module 20 EXFANPDN 0 1 EXFANPDN uk EXFAN)
 (module 21 SPEAKERPDN 0 1 SPEAKERPDN uk SPEAKER)
 (module 22 CABLESPDN 0 1 CABLESPDN uk CABLES)

 //these are all fed to an evalution which will use a variety of specs
//to evaluate environmental impact

 environmental score

 (module 23 ENVSCR 0 1 ENVSCR uk LEDPN VIDPCBPDN MAINPCBPDN PWRSUPPDN
EXFANPDN SPEAKERPDN CABLESPDN)

 //enviornmental specs

 (module 24 ENVSPEC 0 1 ENVSPEC uk)

 //environmental evaluation

 (module 25 ENV1 0 1 ENV1 uk ENVSPEC ENVSCR)
```

APPENDIX B

This appendix contains information about the implementation of the various analysis algorithms. The search algorithm is based on a traditional depth first search, which is explained in the following pieces of Pseudo code which can be found in Chapter 23 of ref.

DFS(G)

```
for each vertex of  u ∈ V[G]
        do color[u] ← WHITE
        all data  ← 0
counter ← 0
for each vertex u ∈ V[G]
        do if color[u] = WHITE
        Visit(u)
```

Visit(u)

```
color[u] ← GRAY
counter ++
find[u] = counter
for each v ∈ Adj[v]  //look through list of dependencies for the current node
        do if color[u] = WHITE
        Visit(v)
color[u] ← BLACK
counter++
finish[u] = counter
```

The complete algorithm uses the Depth First Algorithm as well as a sorting algorithm to generate a hypergraph. The pseudo code for the entire algorithm is :

i. DFS(G)
ii. Sort by finish time
iii. Compute G^T
iv. DFS(G^T)
v. Sort by finish time

APPENDIX C

This appendix provides an overview of the classes used to implement the DOME DSM. Further detail is contained in documentation within the code. The code segments are divided into 5 sections as shown below:

1. C++ plug-in to DOME library.
2. Server Side Including JNI
3. Client Side Including DOME distributed computing components
4. Client Adapters
5. Client GUI components

C++ Plugin

The plug-in contains the implementations of a variety of algorithms and associated utility methods. The code also includes standard implementations for the creation of plug-in modules. The following sequence outlines the execution of the code:

i. Initialize data
ii. Generate dependency structure from DOME model
iii. Apply depth first search algorithm
iv. Reapply depth first search on ordered set of modules

Files: DSMModule.cpp

Server Side C++ and Java

The server side provides a number of basic functions including mapping between Java and C++ through JNI. A number of methods are used to send and receive events to initiate the C++ algorithms and send data to the client.

Files: DSMModuleServer.cpp; DSMModule.java

Client Side Java

The ClientModule duplicates the functionality of the server, receiving and generating events with the server. Instead of communcating back to C++, other interaction takes place with a variety of adapters explained in the next section.

Files: DSMClientModule.java; BeginSortEvent.java, ReadArrayEvent.java, WriteArrayEvent,java,

Client Adapters Java

This includes all elements required to send and receive events from the GUI components as well as similar interaction with the DSMClientModule.java. The two main adapters are for the ListTable and TreeTable components which are used to produce the visualization structures. Presently, the implementation performs a variety of queries to convert dynamically produce structures in response to requests from GUI components and changes propagated up from the Server via the DSMClientModule. In addition to data adapters there are also state, selection and action adapters.

Files: DSMListener.java; DSMActionEvent.java; DSMTreeAttributeAdapter.java; DSMTreeExpansionAdapter.java; DSMTreeStructureAdapter.java; DSMTreeTableAdapter.java; DSMListDataAdapter.java;DSMListModelAdapter.java; DSMZoomAdapter.java;DSMDepListModel.java

Client GUI Components

The main GUI component is the PaneRenderer, which defines the GUI which is lauched by the DOME environment. The TreeTable and ListTable are custom components created by combining Abstract version of the JList, JTree and JTable in swing. The CellRenderers and dynamically generated Icons provide capabilities ot customize information displays, particularly for the table components.

Files: DSMPaneRenderer.java; DSMTreeTale.java, DSMListTable.java; DSMZoomPanel.java; DSMCellRendere.java;DSMListCellRender.java; PlayIcon.java; ReferenceIcon.java

6 REFERENCES

Akama, ten Hagen, Tomiyama. (1994/95) "Desirable Functionalities of Intelligent CAD Systems". Intelligent Systems in Design and Manufacturing. 119-138.

Alexander (1964) "Notes on the Synthesis of Form". Harvard University Press, Cambridge Massachusetts and London England.

Bird. G. M, Kasper, S. D.. (1995). "Problem Formalization techniques for Collaborative Systems." IEEE Transactions on Systems, Man abd Cybernetica **25**(2): 231-242.

Bechtel, Richardson (1992) " Discovering Complexity: Decomposition and Localization as Strategies in Scientific Research " Princeton University Press, Princeton NJ 1992.

Borland, N., (1998), "Integrating Environmental Impact Assessment into Product Design: a Collaborative Modeling Approach," 3rd Design for Manufacturing Conference: ASME DETC98, September 13-16, 1998, Atlanta, GA.

Branki, C. N. (1995). "The Acts of Cooperative Design." Concurrent Engineering: Research and Applications **3**(3): 237-245.

Cukosky, Olsen, Tenenbaum, Gruber,(1994) R. "Collaborative Engineering Based on Knowledge Sharing Agreements" Procedings of the 1994 Database Symposium.

Cutkosky, M. R., R. Engelmore, et al. (1996). "PACT: An Experiment in Integrating Concurrent Engineering Systems." IEEE Computer (January, special issue on Computer Support for Concurrent Engineering): 28-37.

De Pauw, Kimelman, Vlissides (1997) "Visualizing Object-Oriented Software Execution". MIT Press, Cambridge, MA, London, England. Editor John Stakso et al.

DOME framework (1998) cadlab.mit.edu.

Dong, Q (1998) Personal communication.

Egidi, M (1992) "Organizational Learning, Problem Solving and the Division of Labour" Economics, Bounded Rationality and the Cognitive Revolution, Edward Elgar Publishing, Vermont USA.

Eppinger S, K. R. M. a. S. (1993). "Managing the Integration Problem in Concurrent Engineering".

Eppinger S, T. U. P. a. S. D. (1994). "Integration Analysis of Product Decompositions." DTM **68**. Personal Communication.

Eppinger S, R. P. S. a. S. (1997). "Identifying Controlling Features of Engineering Design Iteration." Management Science **43**(3).

Eppinger, N. S. a. S. (1998). "Product Development Process Capture and Display Using Web-Based Technologies". Personal Communication.

Eppinger S, D. E. W., Robert P Smith and David A Gebala (1994). "A Model-Based Method for Organizing Tasks in Product Development." Research in Engineering Design **6**: 1-13.

Eppinger S, M. V. N. a. D. E. W. (1997). "Generalized Models of Design Iteration Uding Signal Flow Graphs." Research in Engineering Design **9**: 112-123.

Franklin, Powell, Emami-Naeini (1994) "Feedback Control of Dynamic Systems". Addison-Wesley Series in Electrical and Computer Engineering.

Invention Machine Software (1998) www.invention-machine.com.

Kim, J. B. and D. R. Wallace (1997). A Goal-oriented Design Evaluation Model. ASME Design Engineering Technical Conference, DETC97/DTM-3878, ASME.

Klir, G. J. (1985). Architecture of Systems Problem Solving. New York and London, Plenum Press.

Kusiak, A. and N. Larson (1995). "Decomposition and representation methods in mechanical design." ASME Journal of Mechanical Design **117**(June).

Kusiak A, Wang J (1993) "Decomposition of the Design Process" Journal of Mechanical Engineering, Vol. 115(December).

Minsky, M.(1986). "The Society of Mind". Simon and Schuster.

Michelena, Papalambros (1995) "Optimal Model-Based Decomposition of Powertrain System Design". Transactions of the ASME Vol. 117.499-504.

Molina, A., A. H. Al-Ashaab, et al. (1995). "A review of computer aided simultaneous engineering systems." Reseach in Engineering Design **7**(1): 38-63.

Pahng, K. F., N. Senin, et al. (1997). Modeling and evaluation of product design problems in a distributed design environment. ASME DETC'97, Sacramento, California, ASME.

Papatheodorou Christos, M. V., Kiountouzis Vangelis (1993). ìProblem Decomposition in Distributed Problem-Solving Systems.î Journal of Applied Intelligence **3**(4): 300-315.

Parnas, D. L. (1972). "On the Criteria to be Used in Decomposing Systems into Modules." Communications of the ACM **15**(12): 1053-1058.

Paranuk, Van Dyke (1998)."Practical and Industrial Applications of Agent-Based Systems". Industrial Technology Institute. http://www.cs.umbc.edu/agents/.

Sacerdoti, E. D. (1977). A Structure for Behaviour and Plans, Elsevier Scientific Publishing Company.

Senin, N., D. R. Wallace, et al. (1996). Mixed continuous variable and catalog search using genetic algorithms. ASME Design Engineering Technical Conferences, Irvine, CA, ASME.

Sougata Mukherjea, j. D. F., Scott E Hudson (1994). "Interactive Clustering for Navigation in Hypermedia Systems." ACM.

Stevens WP, G. M. a. L. C. (1974). ìStructured Design.î IBM Systems Journal **13**(2): 115-139.

Suh, N. P. The Principles of Design. Oxford University Press, Cambridge England.

Sun Java Tutorial 1999). http://java.sun.com/docs/books/tutorial/index.html.

Sussman, G. J. (1975). A Computer Model of Skill Acquisition, Elsevier Scientific Publishing Company.

Tomiya, Yoshikawa, Kiriyama, Umeda (1994). "An Integrated Modelling Environment Using the MetaModel". Annals of the CIRP. Vol. 43.

Ulrich, Eppinger.(1995) "Product Design and Development". New York: McGraw Hill.

Ullman, Herling, Ambrosio (1997) "What to do next: Using Problem Status to Determine the Course of Action" Research in Engineering Design, 9:214-227.

UMBC Agent Web Site.(1999) http://www.cs.umbc.edu/agents/.