Modeling and Evaluation of Design Problems in a Network-Centric Environment

by

Kontong Francisco Pahng

Bachelor of Science, Mechanical Engineering,
University of Iowa, Iowa City, May 1993

Master of Science, Mechanical Engineering
Massachusetts Institute of Technology, June 1995

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

Doctor of Philosophy in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1998

Signature of Author_____
Department of Mechanical Engineering
February 24, 1998

Certified by_____
Professor David R. Wallace
Thesis Supervisor

Accepted by_____
Professor Ain A Sonin
Chairman, Department Committee

I dedicate this dissertation, the accomplishment of my long journey in the States, to my parents and the Lord.

Modeling and Evaluation of Design Problems in a Network-Centric Environment

by

Kontong Francisco Pahng

Submitted to the Department of Mechanical Engineering
on February 24, 1998 in Partial Fulfillment of the Requirements for the
Requirements for the Degree of Doctoral of Philosophy in
Mechanical Engineering

Abstract

Accelerating time to market with improved product quality is one of the driving forces encouraging companies to streamline product development processes these days. Numerous high quality specialized design software tools have played significant roles in various stages of product design processes. However, many of these applications are aimed at expediting or automating specific design tasks rather than providing an integrated product design perspective. In product design it is important to meet potentially competing design objectives and evaluate design solution alternatives from various perspectives. Product design can be viewed as a multi-objective/multi-disciplinary problem, and frequently requires interaction between many designers and design resources.

This research will introduce a computer-aided design method for integrating various aspects of product design problems and help designers interact effectively in a computer network-centric design environment. By providing designers with a framework to formalize and model unstructured product design problems, they can obtain a better understanding of product design problems and effectively share design resources to develop satisfactory design solutions in a cost-effective and time-efficient way. The proposed framework for this environment is called *Distributed Object-based Modeling and Evaluation (DOME).*

The vision for DOME is to provide product designers with an integrated environment for modeling and evaluating product design problems. In this environment, a designer can rapidly build a model of a complex multi-objective/multi-disciplinary design problem, evaluate design solution alternatives from various perspectives and automate the search for feasible and satisfactory solutions using an optimization algorithm.

Product design problems are formalized using objects or modules which reflect both physical and non-physical perspectives of the design. Modules are inter-connected together to form integrated models of the product design problem. When modules interact with other modules through computer networks to form an integrated model of a product design problem, a new environment for product design can be created. In this design environment, designers can aggregate modules available in computer networks to form integrated large-scale models of complex multi-objective and multi-disciplinary design problems. These design models would then enable geographically dispersed designers to build, exchange and trade design models, information and knowledge.

Thesis Supervisor: David R. Wallace
Title: Professor of Mechanical Engineering

Acknowledgments

The work presented in this thesis is not, by all means, the result of one person's struggle. There are many people who have guided me through difficult moments and helped me to finally reach where I am today.

Fist of all, I would like to take this opportunity to thank professor David Wallace, my thesis advisor, for his guidance and support. As a wonderful teacher and a good friend, he was always there for me when I was struggling through my life at MIT. Thanks should also go to Grace, Dave's lovely wife, who is a great cook. I'll always remember her fabulous dishes at the Christmas diner. Especially, Mahn-doo!

I also want to thank professors Woodie Flowers and Steven Eppinger, members of my doctoral committee. Woodie was the very first professor I met in MIT when I came to MIT during the Summer of 1993. I still remember when he told me how he has done his doctoral study not only to get a profound knowledge of his area, but also to widen his vision of the world. Steve, in particular, helped me to realize mistakes that I had in my thesis. His keen advice has enabled me to go a step further.

The MIT CADlab has been a home-away-from-home during my life at MIT. It was not only a wonderful place to work, but also a special place for friendship. I'd like to remember and thank many CADlab alumni; Haeseong-ee-hyoung, Minho-hyoung, Li Shu, Kazu, Jay, Ashley, Nicola, Krish and Paul. I owe million thanks to Seokhoon-ee-hyoung for his work in CADlab, which helped me to complete my thesis. Even though many people have left CADlab, the spirit has never faded away. I'd also like to thank the CADlabbers; especially, Jinpyoung-ee-hyoung, Junho-hyoung, Junbeom-ee-hyoung, Matt, Nick, Tom, Shaun, and Manuel.

My life at MIT has been especially memorable thanks to the many friends in the Korean Graduate Student Association. I would like to give special thanks to brothers/sisters Insoo, Wonjun, Rena, Booho, Shinseok, Seokyoung, Dongsik, Youngchul, Jungmok, Cheolmin, Taejung, Yoonkyu, Donghyun, Jeongyun, Sukwoo, Jaehyun, Hyuksang, Sangjun, Steve, Hoyoung, Seungsun, Taewoong, Jungsin, Jinwoo, Sungwhan, Songyee, and many more.

I must also thank those who have guided and supported me during my study in many ways: Dr. Kwan Lim, Prof. K.K. Choi, Prof. L-D Chen, Prof. Kunwoo Lee, Dr. Harry Yae.

It's been eight years and eight months since I left home to study in the States. It could have been a very lonely place if I didn't meet these great friends. During the wonderful days at the University of Iowa, I met Seongwhan, Junseok, Yoonjung, Junwon, Lisa, Inkyu, Jungyun, and Minseok. My other friends in Boston, who have encouraged me to escape from my window-less office, were Jungsun and Sooyeon. And, I give my very special thanks and love to Jeonghye for her friendship and love, and wish the best for the days to come.

My very last but the most precious thanks must go to my family. Mom and Dad, I have finally done it! Hyoung, look over our family until the day when we reunite in Heaven. Noona and Joel, thank you for your everlasting love and support. Hyoung-soo-nim, I'm coming home! Sungho-ya, I wish all the best for your bright future and happiness. I'll be always on your side.

Table of Contents

List of Figures

1

Introduction

1.1 Motivation

Accelerating time to market with improved product quality is one of the driving forces encouraging companies to streamline product development processes these days. Numerous high quality specialized design software tools have played significant roles in various stages of product design processes. Many of these applications are aimed at expediting or automating specific design tasks rather than providing an integrated product design perspective.

In product design it is important to meet potentially competing design objectives and evaluate design solution alternatives from various perspectives. Product design can be viewed as a multi-objective/multi-disciplinary problem, and frequently requires interaction between many designers and design resources.

This thesis will introduce a computer-aided design method for integrating various aspects of product design problems and help designers interact effectively in a computer network-centric design environment. By providing designers with a framework to formalize and model unstructured product design problems, they can obtain a better understanding of product design problems and effectively share design resources to develop satisfactory design solutions in a cost-effective and time-efficient way.

1.2 Vision

The vision for the proposed framework is to provide product designers with an integrated environment for modeling and evaluating product design problems. In this environment, a designer can rapidly build a model

of a complex multi-objective/multi-disciplinary design problem, evaluate design solution alternatives from various perspectives and automate the search for feasible and satisfactory solutions using an optimization algorithm.

Product design problems are formalized using objects or modules which reflect both physical and non-physical perspectives of the design. Modules are inter-connected together to form integrated models of the product design problem. When modules interact with other modules through computer networks to form an integrated model of a product design problem, a new environment for product design can be created. In this design environment, designers can aggregate modules available in computer networks to form integrated large-scale models of complex multi-objective and multi-disciplinary design problems. These design models would then enable designers located at different places throughout the world to build, exchange and trade design models, information and knowledge. The proposed framework for this environment is called *Distributed Object-based Modeling and Evaluation (DOME).*

Figure 1.1 shows such a design scenario. Designers involved in a hand-held drill design construct a model by aggregating modules from other companies (i.e., motor, battery, and gearbox) with modules created in house (i.e., housing, human factors, and drill performance). As the figure implies, designers working on specific parts of the design problem, the mathematical models and data are not located at the same place. Although the drill and gearbox divisions of an electronic tool company and other OEM's like a gear manufacturer are distributed around the world, the modules of their design problems can be inter-connected through computer networks (i.e., the Internet) and the system behaves as a single integrated model of the design problem.

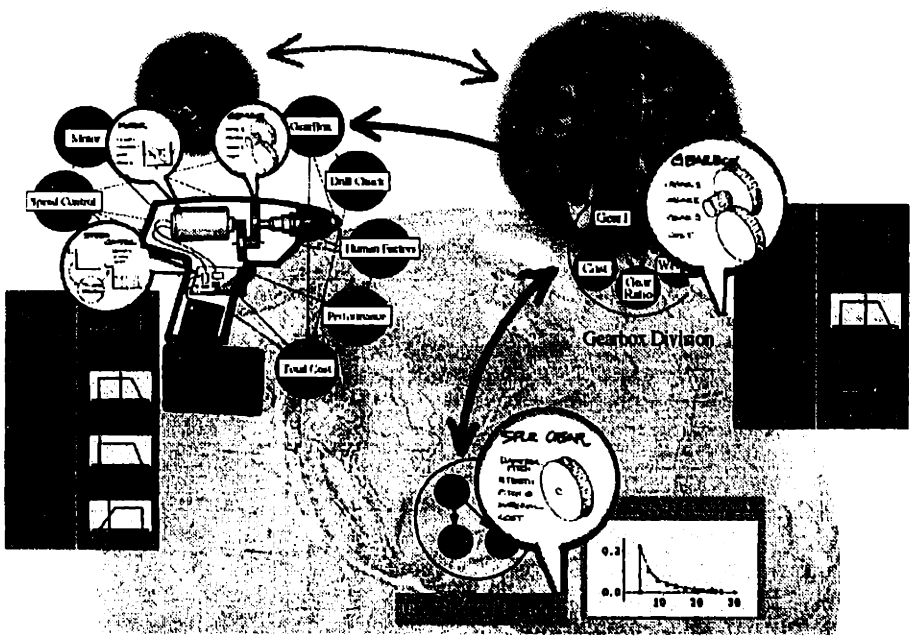


Figure 1.1 Integrated product design modeling and evaluation

At the drill design problem level, a variety of design perspectives can be considered. For example, the module of customers' needs provided by the marketing department can be added so that the drill designers can consider the customer's needs (e.g., size and weight preferences for a targeted market) as they develop a new drill product. Perhaps, the entire drill model can be provided as a module to a new product program manager for strategic planning in his/her division.

1.3 Problem statement

In this research, an object-oriented modeling and evaluation framework (Senin, et al., 1996) is extended to the distributed design environment. The following problems are addressed and resolved.

1.3.1 Inadequate understanding of the paradigm

A thorough understanding of the distributed environment is needed and the systems design requirements for such an environment must be carefully specified. Although a system based upon the object-oriented programming methodology generally has inherent characteristics that are favorable for extending the system to a distributed computing environment (CORNAFION, 1985), the nature of a distributed environment affects, to a great extent, the design and implementation of such a system.

The use modality (e.g., proprietary and public) of the system in the context of product design is also an important consideration because the interactions between modules of a large scale model often reflect the actual relationships between designers and, thus, require careful coordination.

1.3.2 Assessment of theoretical limitations

In the DOME framework, the model of a product design problem constructed by a designer defines the topology of a distributed computing system and information flow within the system. If care is not taken during the modeling, the designer may create an undesirable distributed computing system that is infeasible, unnecessarily complex, or which scales poorly.

1.4 Goal and deliverable of this thesis

The goal of this thesis is to develop the architecture for and demonstrate the DOME framework while resolving the problems identified above. There are three major components in thesis. First, the principles of the distributed design environment and design activities in such an environment are identified. Second, in order to quantitatively address the theoretical issues associated with constructing a design problem model or distributed computing system based on the DOME framework, a computational model is investigated and a complexity measure of a DOME model is presented. Using the measure, system characteristics (e.g., complexity, scalibility and performance issues) can be addressed. Also, the potential limitations of the current implementation (e.g., circular dependency) are identified and several solutions are proposed.

Third, a DOME prototype is implemented and a number of product design problems are presented to validate the framework. The preliminary DOME prototype is then further enhanced by adopting new software technology (i.e., Java, CORBA, etc.) to provide a versatile design environment on the World Wide Web.

1.4.1 Characterization of the distributed design environment and design activities

Principles of the distributed design environment

The thesis enumerates the characteristics of the distributed design environment, where design participants interact through computer networks in order to explore and evaluate design solution alternatives. The computational environment of the proposed framework is a distributed computing environment. It has an inherent characteristic of parallelism or concurrency. Many studies of systems design in a distributed environment have been done (e.g., distributed object-oriented programming, distributed operating system design, etc.). The understanding of a distributed environment in such disciplines provides useful insights, if they are carefully interpreted in terms of the modeling and evaluation of design problems.

Design activities amongst design participants and their coordination

In the context of the modeling and evaluation of design problems, possible design activities amongst design participants distributed around networks are identified. Different types of relations or dependencies between distributed design participants require different aggregation or coordination to form an integrated model. In the DOME framework, the coordination of design activities is provided through different levels of modeling privileges in the environment.

1.4.2 Formal studies of the distributed modeling and evaluation system

To formally represent a design problem model in DOME, a notation formalism is presented. This provides designers a means to describe a DOME model in detail and a basis for the formal studies of the DOME framework. The computational model of a DOME model is obtained by associating the formal notation and computational characteristics of the model. The computation time of evaluating a design solution alternative is used to assess the complexity of a model. This complexity measure can assist designers in building a more efficient and effective product design problem model. The improved model will help the designers to quickly evaluate and search design solution alternatives using DOME framework.

1.4.3 Implementation of the DOME framework and its validation

Based on the knowledge and techniques learned in the previous steps, an appropriate architecture of a computer-assisted design system based on DOME framework is proposed and its prototype is implemented

to explore the merits of the proposed framework. The framework and its implementation will be also validated with an engineering design problem.

1.5 Organization of thesis

In chapter 2, the background for the thesis is presented. It first provides a scenario of the product design process that the proposed DOME framework is intended to provide. The scenario clearly illustrates how the framework can help designers to expedite the design process in a collaborative environment. Next, the general concept of the problem modeling in DOME and the fundamental modeling and evaluation framework called Object-oriented Modeling and Evaluation (OME) is presented. Last, background on the interprocess/cross-platform communication protocol, CORBA, is presented

Chapter 3 reviews past and on-going research work relevant to the work presented in this thesis. First, it provides the general background on the term *modeling* in the context of product design to help the readers clarify what kind of modeling the proposed framework deals with. Then, research on the area of computer-assisted design systems in the computer network-centric environment is presented.

In chapter 4 the issues in the development of computer network-centric design systems is discussed and followed by the general concept of the proposed DOME framework. Then, the system architecture that provides the necessary capability for supporting collaborative design problem modeling is discussed.

Chapter 5 presents the formal studies of the proposed DOME systems. It provides a formal notation for a DOME model that is later used as the computational model of a DOME system. The system characteristics such as the computation time is discussed and the problem of circular dependencies in a model is addressed.

Chapter 6 provides the implementation details of the DOME framework. The first half provides the capability of the initial DOME prototype and its system elements. A simple example illustrates how a model is created. Then, the on-going implementation of a Web-based DOME system is presented.

In chapter 7, examples of design problem modeling using DOME are provided to show the capabilities of the preliminary DOME system implementation. These include the implementation of a hand-held power drill design problem presented in Section 1.2 and a bottle design problem that shows the integration of DOME with existing software applications (i.e., ProEngineer®, Excel®, and TEAM®).

Two other examples including a central district heating plant design problem model are presented to show the Web-based DOME implementation[1].

---

[1] ProEngineer, Excel, and TEAM are the registered trademarks of Parametric Technology Inc., Microsoft, and Ecobilan, respectively.

Chapter 8 summarizes the work presented in this thesis with conclusions and provides the direction for future studies.

2

Background

The vision of the proposed framework is to provide product designers with an integrated environment for modeling and evaluating product design problems in a computer network-centric distributed design environment. This vision for DOME is perhaps best illustrated through a scenario. In this chapter, a scenario of the product development environment for the drill design illustrated in the previous chapter is first presented. Then the issues associated with design problem modeling, decomposition of a complex design problem, and distribution of a problem are discussed and followed by a brief description of the object-oriented product design problem modeling approach used in this thesis. This chapter draws upon a paper titled "Distributed Modeling and Evaluation of Product Design Problems" (Pahng, et al., 1998).

2.1 A Design Scenario

Considering the example of a battery-powered cordless drill presented in Figure 1.1, many components (motors, gears, speed control, etc.) may be designed and manufactured by part suppliers, while other elements, such as the housing, might be designed in-house by the OEM. The designers need to carefully integrate available standard or semi-custom parts with custom created components in order to achieve a number of performance goals. This is challenging as it requires an understanding of the tradeoffs amongst numerous and potentially conflicting design objectives.

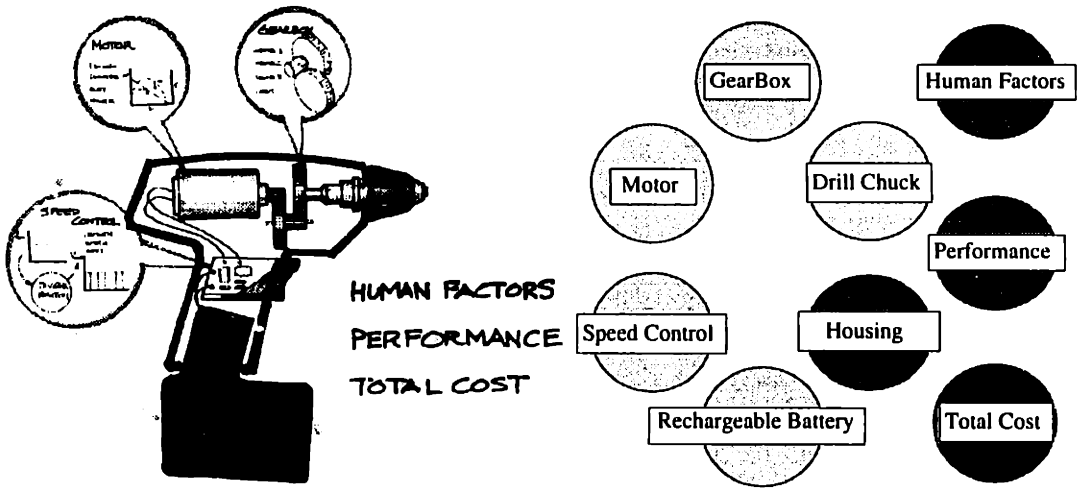


Figure 2.2 Product topology of a hand-held power drill and its decomposition

The topology of the problem is defined in Figure 2.2(a), identifying the main components and their principal functional roles. Some design issues are also identified (human factors, performance, cost). Figure 2.2(b) shows how this design problem might be decomposed into *modules*. This decomposition reflects both the physical subdivision of the product into components or subassemblies and the evaluation perspectives or analysis capabilities.

In a design tool based on the DOME framework, these modules can be plugged together to form an integrated model of the design problem. They contain data and mathematical models. They can provide information about themselves or perform analyses given inputs from other modules. They provide services which evaluate their status. For example, the motor module will provide its geometric information and performance characteristics given operating conditions.

Clearly, a single designer does not have sufficient time, knowledge, or information to create all the individual modules needed for the integrated design problem model. This implies that design knowledge and modeling responsibilities are distributed amongst many designers. A possible distribution is shown in Figure 2.3. Several companies and divisions within companies are involved in the power drill design. As shown in Figure 2.3, these distributed design participants are connected through a computer network.

Given this context, scenarios from different viewpoints will be used to illustrate the functionalities to be provided by the DOME framework.

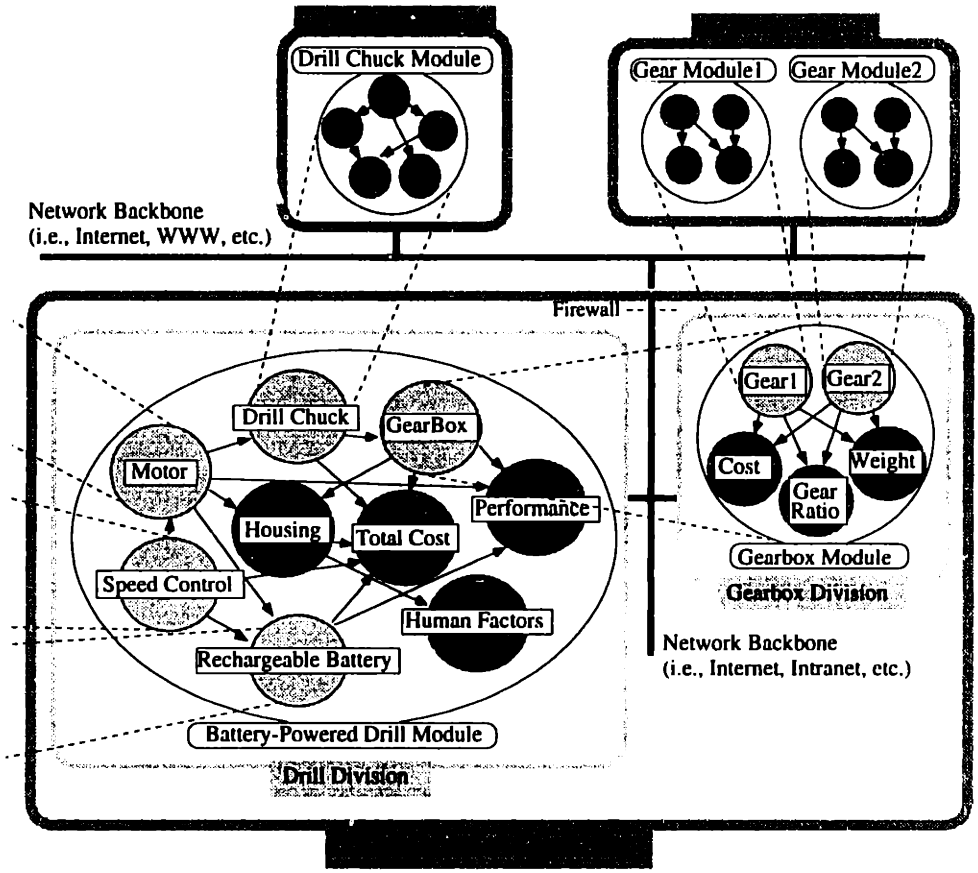


Figure 2.3 Distribution of the modeling resources for the drill scenario

2.1.1 Gear Manufacturer: a part supplier/module provider

The Gear Manufacturer is a part supplier for other companies. They also participate in the distributed design problem modeling environment. Using their in-house mathematical modeling capabilities and software applications, they have created modules representing the characteristics of their gears.

Since they produce different families of gears, they have a number of DOME modules, each representing a particular group of gears. These modules are organized into catalogs, ready to be selected by customers.

2.1.2 Gearbox Division: a subassembly supplier/module provider

The Gearbox Division is a division of the company that makes power tools. It offers design services and provides gearboxes for the parent drill manufacturer[2]. The Gearbox Division buys gears from the Gear Manufacturer. From their viewpoint they must solve a complete design problem, selecting suitable gears

[2] Note that this subdivision is product-centered. When referring to divisions, we imply all the people who work on a particular product. The division itself might be geographically distributed.

and determining several design parameters in order to meet requirements for use as a subassembly in the drill.

To be part of the DOME environment, the Gearbox Division develops modules describing its gearbox designs. Since the physical gearbox will make use of gears provided by the Gear Manufacturer, gear modules provided by the Gear Manufacturer are also embedded into the gearbox problem model. The gearbox design problem model is used by the Gearbox Division to explore design solution alternatives. Many design decisions, including which gears to buy from the Gear Manufacturer, may be explored. These designs will be made into DOME modules. Since the Gearbox Division is incorporating modules from the Gear Manufacturer, an agreement on intellectual property and usage modalities will be reached before this process is completed[3]. If the Gearbox Division has generated many alternative solutions, the corresponding modules may be organized into catalogs for the Drill division to select from.

2.1.3 Battery-Powered Drill Division: original equipment manufacturer/module provider

The Drill Division creates the complete drill design. It relies upon several part suppliers/module providers. It uses gearbox modules from the Gearbox Division and other module providers for the battery, the chuck and the electric motor. The gearbox design, a complete product design problem from the Gearbox Division's viewpoint, is seen as a single module from the drill design problem perspective. Providers prepare modules that allow the Drill Division to test or use the capabilities of their product.

Thus far, the module providers are all part suppliers that manufacture physical components once the customer is satisfied with the performance characteristics revealed by corresponding DOME modules. However, a DOME module provider may not necessarily also be a part supplier.

2.1.4 Information Broker: a pure module provider

Some companies, that we call Information Brokers, may exist only to locate module providers and to connect them to potential customers. In a DOME environment, such companies would probably become module providers themselves, taking modules from part suppliers and restructuring them into catalogs. They may provide services to their customers for searching their catalog-based databases in order to select, test or buy modules. Eventually, they may buy the components directly from the suppliers. In addition to Information Brokers, many other pure providers, such as software companies or engineering consultants might offer their services through modules.

---

[3] Note that incorporating a module does not necessarily mean copying it into the current model. Incorporation may also consist of the establishment of a permanent remote communication with the module.

In the scenario, the Drill Division has contacted an Information Broker to locate a drill chuck. The Information Broker may provide a wide choice of modules to test in the drill design problem before selecting the most appropriate chuck supplier.

2.1.5 Scenario Summary

Modular approach: both distributed and local

Designers are likely to use pre-existing modules to model standard parts or subproblems, derive new modules from pre-existing ones for semi-custom parts and create others from scratch for custom parts or previously unmodeled problems. Designers build an integrated problem model by combining modules. When modules are connected, they form a model that can be used to predict characteristics of the design and evaluate them against requirements. Furthermore, a complete design problem model (e.g., the gearbox) can become a new module, available for others to use as a subproblem in a larger integrated design problem (e.g., the drill). This is illustrated in Figure 2.4.

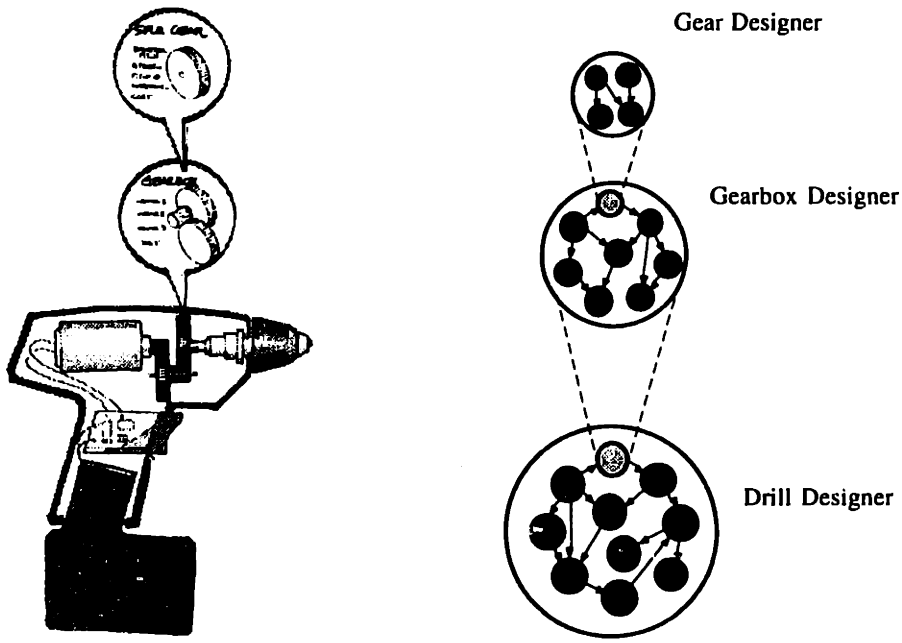


Figure 2.4 Example of how design problems use modules and can become modules for other design problems

In the distributed design environment, each designer is potentially both a module customer and a module provider. Distributed modules communicate with each other through a standard network protocol.

Modules and parts

A module provider can be also a part supplier. Typically, this means that once the customer has tested his design model using modules that represent physical parts, he will purchase the components. Modules may

refer to standard off-the-shelf or semi-custom produced items. For example, the Gear Manufacturer may provide the Gearbox Division with modules that allow them to customize different classes of gears. Likewise, the Drill Division may be provided with gearbox modules that allow semi-customization.

Part suppliers may often choose to provide modules for free in anticipation of increased product sales. Other companies, such as Information Brokers or firms which provide analysis services, may sell modules as services to designers.

Information hiding and reusability

A module interacts with other modules independently from its internal implementation. In other words, a module might hide an embedded model based on a CAD system, a FEA package, or any other engineering software package. For example, the gearbox module would probably include a geometric representation of the product and several computational capabilities (e.g., input-output relationships and failure prediction). Modules can be built with embedded models strongly linked with preexisting applications. These modules do not need to be copied to the local problem model as their functionalities can be accessed remotely through requests for their services.

2.2 Product Design Modeling

In a product design process, many interrelated decisions must be made. These range from initial decisions about the topology of a design to variations of design parameters over continuous intervals (e.g., the length of a shaft) and choices among discrete sets of alternatives (e.g., a material, a motor type, etc.). A set of such decisions constitute what we call a *product design problem.*

2.2.1 Design problem modeling

Design decisions are made to meet potentially competing objectives that may span many disciplines, such as manufacturing, in-use performance, cost, or life-cycle and environmental considerations. However, the effects of design decisions propagate and interact with each other in ways that often make it difficult to fully understand the integrated problem. Modeling of product design problems is a way to help the designer predict effect of decisions or choices.

Product modeling is used to capture properties and the geometric representation of a product. We use *design problem modeling* to relate product models and product data to design objectives/ decision criteria. A design problem model can be used to predict product properties and *evaluate* how well solution alternatives meet stated design objectives. The proposed DOME framework is intended for *design problem modeling*.

2.2.2 Decomposition to manage complexity

Modeling and evaluating many aspects of a large problem involves specialized knowledge in many fields that cannot be mastered by a single designer. Problem decomposition is a key technique to manage complexity. Different aspects of a product design problem can be modeled and solved separately by different people who have appropriate core competencies. However, this typically makes it hard to understand the tradeoffs between goals related to different subproblems. Time consuming iterations and perhaps inferior decisions can result. However, we suggest that, if the decomposition is appropriately structured, large problems can be modeled by aggregating subproblems so that the interaction between subproblems can be captured. This is the purpose of the Object-oriented Modeling and Evaluation (OME) framework. In addition, well structured and generic models of subproblems are also potentially reusable in other problems (e.g., a model representing a particular manufacturing process should be reusable whenever that particular manufacturing process is involved in a design problem). The goal of reusability is to significantly reduce the time needed to build design problem models.

2.2.3 Distribution of decomposition

Decomposition structures and separates the knowledge needed to model a problem. Once the modeling tasks have been distributed, there is the problem of putting the pieces back together. This is critical in design problem modeling for making optimal tradeoffs between objectives. Integration also requires agreement on communication protocol and terminology. It requires an awareness and understanding of the needs and capabilities of each subproblem model. For example, a representation called the design structure matrix helps designers visualize what information is needed and provided by different design subproblems (Eppinger, et al., 1994). An object-oriented approach such as the one adopted in OME can also help to resolve such issues. Object-oriented modeling enforces decomposition of problems into units with compatibility and reusability (Graham, 1994). Representing a subset or an aspect of the problem, each object acts as a standalone model managing data and the services that it can provide. The integrated design problem model is accomplished by different objects in simultaneous execution.

In Distributed OME, objects exchange information by means of a standard communication protocol. Services are invoked by requests from clients over communication networks. We refer to the objects in the design problem modeling domain as modules. A module is distributed if it can be executed as a standalone unit and is capable of providing its services to other modules, objects or applications through a standard network communication protocol. Thus, an integrated model of a complete product design problem can be built using modules that are physically distributed. Communication, for example, can occur through the Internet or the World Wide Web (WWW or Web).

The proposed Distributed Object-based Modeling and Evaluation (DOME) framework uses these principles to provide designers with a powerful and flexible environment to model and evaluate design problems using

modules and distributed modules. The goal is to provide designers with the ability to rapidly construct integrated design models that reflect interactions between subproblems in the effort to reduce development time and improve overall product quality.

2.3 Object-Oriented Modeling and Evaluation (OME) framework

The notion of an object-oriented product design problem modeling approach was illustrated in the design scenario. The object-oriented modeling and evaluation framework (OME) is the backbone of DOME. OME deals with the definition of the basic entities that are used to model design problems. Preliminary work on object-oriented modeling may be found in (Senin, et al., 1997; Senin, et al., 1996). Here, the key elements to be addressed by the OME framework are described.

2.3.1 Quantities and relations

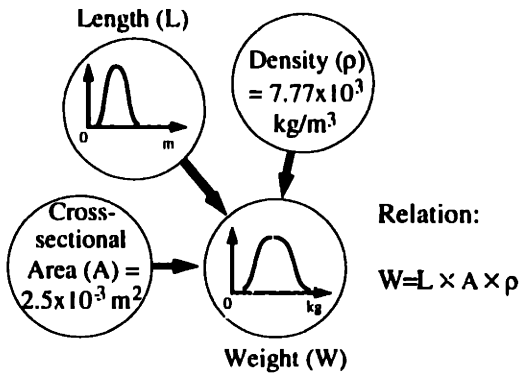


Figure 2.1 A model: variables and relations

Many aspects of design involve mathematical models of design problems that usually deal with quantities related to geometric properties (e.g. length), physical properties (e.g., weight), or other aspects of the problem (e.g., cost). These quantities may be deterministic (certain) or probabilistic (uncertain). Typically, both deterministic and probabilistic quantities may exist in a given problem. The OME framework models such quantities and, in the future, must model any standardized product model data structures such as those based upon STEP (Owen, 1993).

The OME framework provides the same capabilities of a programming language in terms of defining any kind of mathematical relations among variables and seamlessly handles both probabilistic and deterministic data. Variables and relations together form a *model* as illustrated in Figure 2.1.

2.3.2 Modules

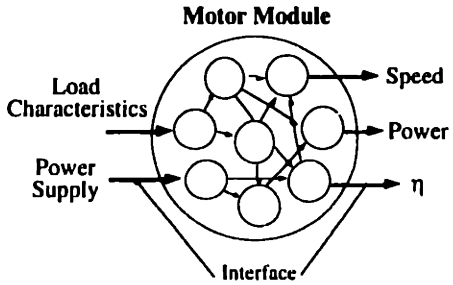


Figure 2.2 Simplified motor module: interface and embedded model

The basic building block of the OME framework is the *module*. A module is an object representing a particular aspect or physical component of a product design problem. Modules were referred to extensively in the design scenario. A module is capable of providing services to the designer. Figure 2.2 illustrates a motor module.

A module contains variables and their relations (i.e., the *embedded model* of the module). Services operate using the embedded model. For example, a motor module should be able to provide its speed, power and efficiency given a power supply and load characteristics as inputs.

Modules allow the designers to group together knowledge related to a particular issue in a product design problem. If the module is general, it may be potentially reusable in other contexts where the same services are required, (e.g., other design problems involving motors may make use of the same motor module).

2.3.3 Services and interfaces

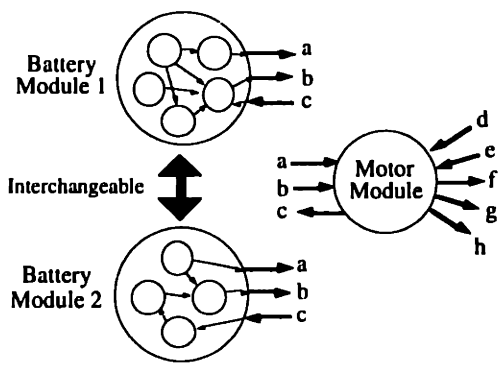


Figure 2.3 Interchangeable battery modules with respect to the motor module

In OME, the set of services that a module can provide is its *interface*. Modules can interact with each other if the interfaces are compatible. That is, the modules can mutually satisfy services for requested inputs and outputs.

In OME, a module sees other modules only in terms of their interface. This means that two different modules capable of providing the same services are indistinguishable from the viewpoint of another module asking for those services. For example, any battery module capable of providing the motor module with the services requested can be used as (shown in Figure 2.3).

These battery modules can have very different embedded models. They might represent different battery technologies; they might come from different vendors and have been modeled differently; or they may differ in the quantities in the embedded data structure (e.g., different Ah cells, different cell costs, etc.).

2.3.4 Wrappers

Since a module can be accessed only through its services, anything can be put inside the module as the embedded model. The term *wrapper* is used to indicate a module that is compatible with the OME framework, but whose embedded model is a preexisting software program (e.g., a CAD system, an FEA package, a spreadsheet, a database manager, etc.). Wrappers can be used to allow preexisting software programs to communicate with other modules in the OME framework. The communication mechanism between the embedded application and the wrapper is provided by the user. The wrapper itself only provides the standard OME communication mechanism.

2.3.5 Building design problem models

Modules can be used as building blocks to create an integrated model for a given design problem. The integrated model of a product design problem is a collection of modules embedded into each other and/or interacting at the same level through service call chains. Macroscopically they provide an integrated system behavior.

The OME framework should be able to support both bottom-up and top-down design approaches. In the bottom-up scenario, modules are first constructed in detail and then either joined together or embedded into one another. In the top down scenario, the integrated model of the problem is first created as a single partially defined module and then progressively all the services needed are categorized and organized into modules embedded in the integrated module.

Predefined modules can be used *"as is"*, or preexisting modules could be adapted to specific needs. The latter mechanism can be achieved by using inheritance mechanisms, a common property of object oriented approaches. Finally, completely custom modules can be built from the ground up.

2.3.6 Interchangeable modules

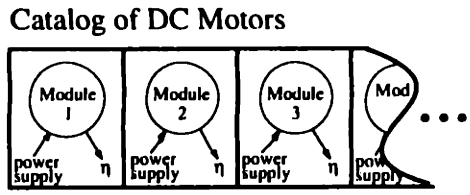


Figure 2.4 Example catalog of DC motors

Since a module is accessible only through its interface, modules that have a common set of services can be interchanged with each other within the context of a problem model. In the drill scenario, different gear modules can be interchanged in the gearbox problem provided that each module is capable of performing a set of services as required by other modules. Modules that are interchangeable in a given problem model are called *replaceable modules*. Replaceable modules can be used for design problems involving catalog selection. A set of replaceable modules representing DC motors can be organized into a catalog of motors as shown in Figure 2.4. Drill designers could then select any motor module from the catalog and connect it to the integrated drill design model.

2.3.7 Interchangeable modules and modeling accuracy

Another use of replaceable modules relates to modeling accuracy. Several modules representing the same subproblem at different levels of detail can be defined. Provided the services are compatible, they can provide a seamless transition from approximate to detailed models. Modules with complex embedded models may be used to give more accurate responses with higher computational expense, while modules embedding simplified models may provide less accurate responses at lower computational expense. This approach offers advantages in both the performance of the integrated problem model (providing a rapid way to optimize between speed and accuracy) and in modeling flexibility.

Interchangeable modules and abstraction

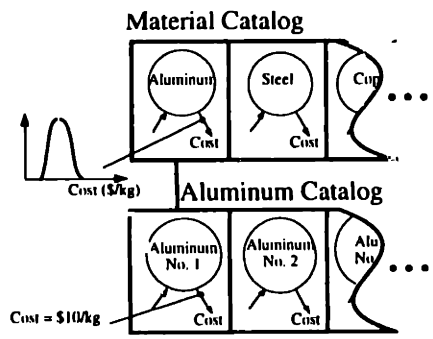


Figure 2.5 General Aluminum module and specific Aluminum module in a material catalog hierarchy

Replaceable modules might also be used to represent the same subproblem at different levels of abstraction or generality. For example, a module might be defined to represent a family of aluminum in a material catalog as shown in Figure 2.5. Another could represent a specific aluminum in the same material family. The designer is then free to evaluate a design using the general family of materials (represented by intervals or probability distributions) or, through a module replacement, evaluate the performance of a specific aluminum. Abstraction can be useful to organize catalogs into a hierarchy as depicted in Figure 2.5.

2.3.8 Interface and physical compatibility

Interchangeability of modules is based on the compatibility of interfaces (i.e., the services). This information-based compatibility should not be confused with physical and geometrical compatibility. Product components must physically fit together in addition to functionally interacting in a meaningful way. OME focuses on information modeling issues. Compatibility is in the information model. Still, geometrical and physical issues can be represented in mathematical terms, and thus, several aspects of physical compatibility should be addressable in the OME framework. The drill scenario will be used in the implementation section to illustrate that problems related to geometry can be represented in the framework.

2.3.9 Evaluation

Given an OME model for a product design problem, *solutions* are sets of states of the key parameters (design variables) defined within the model. In the drill example, a solution can be identified as a set of values of the independent continuous variables (e.g., the width of a rib in the drill housing) and choices for replaceable modules (e.g., the chuck or motor). The evaluation of solutions against requirements or needs (such as cost, performance and ergonomics) is necessary to compare alternatives. Typically, many competing perspectives must be balanced when generating an objective function to evaluate a solution. It is often difficult to find the relative importance of each factor with respect to the others and to organize them all into a meaningful metric. Various decision theory approaches address the issue of constructing evaluation models (for example, (Keeney and Raiffa, 1993; Saaty, 1980)).

In OME, the evaluation model can be embedded in a module. Thus a module can also provide a designer with *evaluation services*. Any module may have its own evaluation perspective (set of design requirements). For example, the gearbox module may have specifications/requirements on its gears plus overall specifications provided by the drill company. The evaluation service will describe how well these requirements are satisfied. The embedded mathematical model for such an evaluation might be quite complicated, requiring expertise and special knowledge to be formulated, as much as the design model itself.

In OME, each evaluation service is usually referred to as a *Lens* (a metaphor to an evaluation viewpoint or perspective). The designer is free to define the behavior (evaluation model) of the lens service. The current

implementation of OME also embeds a facility based on a probabilistic acceptability model (Wallace, et al., 1996). The acceptability model provides the designer with an intuitive way to evaluate design performances.

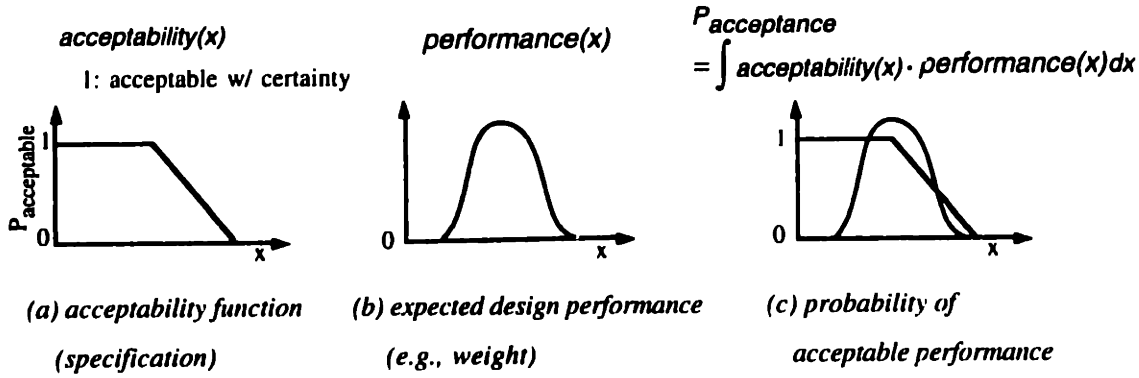


Figure 2.6 Probabilistic acceptability model

A specification-like acceptability function (Figure 2.6(a)) is used to indicate desired performance levels and is compared to the expected design performance (Figure 2.6(b)). A single quantitative output that expresses the probability that the design's performance will be deemed as acceptable is computed (Figure 2.6(c)). This evaluation model resembles traditional design specifications and is implemented in OME. Our implementation example will use this approach. However, other multiple attribute evaluation services could be implemented as specialized modules.

Interchangeable modules and module organization services

Given that one of the main objectives of OME is to allow the reuse of modules and interchangeability of compatible modules, it makes sense to store interchangeable modules into structures to facilitate the search and selection mechanism. OME provides a catalog architecture to organize modules into structures (Senin, et al., 1996). Ultimately, different catalog structures can be created from the same set of modules, each organizing the modules in a different way according to some specific perspective (e.g. vendor based, model based, accuracy based etc.). Furthermore, since many relevant engineering data are already stored in preexisting databases, wrappers must be provided to generate catalogs of modules linked to preexisting data representations.

2.3.10 Summary of capabilities of OME

OME is a framework to support object-oriented modeling and evaluation of product design problems. A problem created in OME is decomposed into modules that interact with each other by means of compatible services. OME supports the reusability of modules through standardization of interfaces and the structured organization of modules into catalogs. The framework should also provide modeling power and flexibility by enabling basic object-oriented mechanisms such as inheritance and encapsulation. This allows for

modeling abstraction and specialization. Evaluation models are also embedded in the modules to provide evaluation service capabilities. These services may be used to observe design quality from different viewpoints, compare different alternatives and form objective functions for use by search/optimization techniques.

2.4 Distributed Computing Environment

The work presented in this thesis addresses the modeling of design problems in a computer network-centric distributed design environment. This environment is inherently based upon the client/server architecture and distributed computing environment, which enables a number of autonomous software applications to communicate with one another and complete given tasks.

The proposed DOME framework is in particular based on the standard communication protocol for network-oriented computing, which is called Common Object Request Broker Architecture (CORBA)[4]. Since the underlying implementation detail of CORBA in DOME is hidden form the users of the system, it provides a seamless integration of software components or modules to form a model of design problem.

2.4.1 CORBA

The CORBA, which was introduced in 1991 and driven by Object Management Group, is a specification for distributed object technology (Orfali, et al., 1996). It is aimed for realizing the interoperability among heterogeneous software applications. CORBA allows software applications to communicate with one another regardless of the location or hardware where the applications are running by defining standard communication protocol based upon its Interface Definition Language (IDL) (Deshpande, 1994; OMG, 1998; Siegel, 1996).

The functionality of the CORBA standard is realized by the implementation of an Object Request Broker (ORB), which is a middleware that establishes the client-server relationships between distributed objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In doing so, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

---

[4] DCOM/ActiveX, which is the rival distributed object technology from Microsoft, will not be adopted in this work as its distributed computing environment is by large limited to Microsoft's Windows/WindowsNT environment (Brockschmidt, 1995; Chappel, 1996).

3

Related Work

The purpose of this chapter is to review past and on-going research work relevant to the work presented in this thesis. It first provides the general background on the term *modeling* in the context of product design to help clarify what kind of modeling the proposed framework addresses. Then, research on the area of computer-assisted design systems in the computer network-centric environment is presented with different perspectives of design. Last, several conceptual models for representing a system that can be characterized by a collection of simpler subsystems or components are presented classifying their features.

3.1 Modeling in the paradigm of design

The proposed framework presented in this thesis is a computer-aided modeling environment where geographically dispersed designers create an integrated model of a design problem to examine different design solution alternatives and choose a set of good design solutions to satisfy design goals.

In the community of design research, the term "modeling" is used in several contexts. These different contexts are briefly presented based upon the intended use of the model. Although the research areas in the following categories are often not mutually exclusive, it will provide a broader view of product and process modeling in the context of design and a better understanding of the product design problem model created in the DOME framework. For more extensive taxonomy in design research and modeling formalisms, refer to (Bliznakov, 1996; Finger and Dixon, 1989).

3.1.1 Design Process

One primary issue in the study of design is to better understand "how designers design" and to subsequently develop tools that can support or improve the design process and product. This is so called *descriptive modeling* of the design process (Finger and Dixon, 1989).

Product development process modeling

Process modeling of product development addresses issues involved in a large design project or complex manufacturing process where many tasks and/or teams are interrelated. The models of design process (e.g., PERT chart, Design Structure Matrix, etc.) are created to provide insights into the complex and, sometimes, unwieldy activities in order to control and improve the process (Gebala and Eppinger, 1991; Steward, 1981; Wiest and Levy, 1969).

Information modeling

Information modeling in the context of the design process generally refers to the representation of Design Information Systems (DIS) and the data flow within them. The information model provides a framework to capture and retrieve the engineering design information. Many modeling frameworks are intended for supporting collaborative design, change propagation, constraint maintenance, design re-use, and design process management, control, and analysis (e.g., (Bliznakov, 1996)). Other researchers have also studied the representation of information flow between design processes. For instance, the Structured Analysis and Design Technique (SADT) is one well-known notation for the systematic representation of information flow in the form of Data Flow Diagrams (Marca and McGowan, 1986).

Problem modeling

Many studies have been conducted with the perspective of the design process as problem solving. A design is modeled as a set of equations or engineering formulae and design constraints. Various techniques are applied to solve the problem quickly or to find the optimal solution that satisfies all the given constraints. In the course of finding the best solution or set of solutions, a variety of optimization methods (e.g., linear programming, combinatorial search techniques, genetic algorithms, simulated annealing, etc.) are adopted (Arora, 1989; Goldberg, 1989).

Decision-making and conflict resolution

Modeling in this area addresses issues in selecting the best design solution alternatives with the existence of conflicting design goals. Since decision-making, like the activity of designing, is not confined by physical laws, there is an abundance of decision models such as Axiomatic design methodology, Pugh charts, House

of Quality, Probabilistic Specification-based Model, and so forth (Hauser and Clausing, 1988; Pugh, 1990; Suh, 1990; Wallace, 1994).

3.1.2 Artifact

In the course of product design and design research, many different aspects of artifacts need to be modeled in order to understand their underlying characteristics and to improve unsatisfactory characteristics. Some areas that are intensively studied are functionality, geometry, features, assemblibility, tolerance and so forth. Since the computer has become an indispensable design tool in various fields, many studies have been done addressing how to represent and exchange product information in computer media. In this respect, a framework for the standard representation of product data called STEP has been on intense research topic in recent years (Owen, 1993).

The Object-oriented Modeling and Evaluation (OME) framework used in DOME can be positioned in the problem modeling and decision making process in product design. It provides the generalized computer-based design tool that supports the modeling of probabilistic and deterministic design problems within a unified decision making environment. Since OME is based upon an object-oriented design methodology, the framework provides a way of explicitly describing the information exchanged between different subproblems. And, as these subproblems or modules managed by different designers are geographically dispersed and distributed in computer networks, the DOME framework can provides a means to visualize the interaction between design team members.

3.2 Computer-assisted design systems in the network-centric environment

Recent advances in the open computer network and information technology have greatly influenced the types of information available to engineers and the general public (Deitz, 1996; Dertouzous, 1991). The World Wide Web (WWW), which is based upon HTTP (Hyper-Text Transfer Protocol) client-server communication, is becoming the standard for providing information on the Internet. This network-centric technology has catalyzed the emergence of numerous pragmatic applications for a wide range of areas, such as CAD/CAM, concurrent engineering, collaborative engineering and manufacturing services.

Several researchers have been working on enabling technologies or infrastructure that can assist product designers in the computer network-centric design environment. Some of these are intended to help product designers to collaborate in or coordinate design projects by sharing product information and manufacturing services through formal or informal interactions. Others are intended to provide designers with a formalized framework that manages conflicts between design constraints and to assist designers in making decisions.

3.2.1 Design information system for collaborative design

The SHARE project by Cutkosky *et al.* is aimed at supporting design engineers or teams by allowing them to gather, organize, re-access and communicate design information over computer networks to establish a "shared understanding" of the design and development process (Toye, et al., 1993). While SHARE is primarily directed towards interaction among design engineers or teams through integrated multimedia communication and groupware tools, the NEXT-LINK project incorporates agents to address the problem of coordination among design decisions affected by specifications and constraints (Petrie, et al., 1994). Another network-centric design system using interacting agents to integrate various manufacturing services available over a network is also under development (Frost and Cutkosky, 1996). The motivation and vision presented in this paper has similar themes but emphasizes mathematical modeling, decision-making and search/optimization.

Maintaining the look and feel of an engineering document, the *Electronic Design Notebook* (EDN) is an interactive electronic document which provides an integrated user interface to computer programs, design studies, planning documents, and databases (Lewis and Singh, 1995). In the EDN, the manufacturing tools and services are encapsulated in hypertext documents and distributed through servers using HTTP (Erkes, et al., 1996; Sobolewski and Erkes, 1995).

A design information system proposed by Bliznakov *et al.* incorporates a hybrid model for the representation of design information at several levels of formalization and granularity. It is intended to allow designers in a large virtual organization to indicate the status of tasks assigned to each designer or team so that other designers can follow their progress (Bliznakov and Shah, 1996; Bliznakov, et al., 1995). The functional requirements of the proposed system suggest that a central database manages pointers and access methods for product and process information in a distributed environment.

Hardwick *et al.* proposed an information infrastructure architecture that enhances collaboration between companies in the design and manufacture of new products (Hardwick and Spooner, 1995). This architecture integrates the World Wide Web (WWW) for information sharing on the Internet with the STEP standard for product modeling. It utilizes the Common Object Request Broker Architecture (CORBA) for interoperability between software applications in the virtual enterprise.

The n-dim is a computer-based collaborative design environment that captures, organizes and shares data and information of complex corporate activities (Westerberg, et al., 1995). The system is viewed as a base on which applications can be added for the purpose of history maintenance, access control and revision management. The primary focus of this research relates to how to model information. The system provides various ways of defining the type of information which determines the relations between data or models.

There are also increasing national-level collaborative efforts between universities and industries to bring a variety of engineering services that designers can access and utilize during product design and analysis using the Internet (MADEFast, 1998; NIIIP, 1998; RaDEO, 1998).

The RaDEO program is concerned with comprehensive information modeling and the design tools needed to support the rapid design of electro-mechanical systems. It is aimed at supporting the engineer by improving his/her ability to explore, generate, track, store, and analyze design alternatives. The National Industrial Information Infrastructure Protocols (NIIIP) Consortium is concerned with developing open industry software protocols that will make it possible for manufacturers and their suppliers to effectively interoperate as if they were part of the same enterprise. The NIIIP is aimed at enabling suppliers throughout America's industrial base to take advantage of recent advances in object information technology (CORBA); product data definition (STEP); and communications networks (INTERNET) to assemble "Virtual Enterprises".

The CONSENS project (Concurrent & Simultaneous Engineering System) is a European project that is exclusively devoted to concurrent engineering. It is largely focused on the information infrastructure where process management, product data management and project data management are supported by several engineering software applications integrated with the SIFRAME framework (Singh, 1995).

3.2.2 Design rationale and conflict resolution

Case and Lu's Discourse Model is intended to provide software support for collaborative engineering design by treating interactions between designers as a process of discourse (Case and Lu, 1996). The model treats design commitments made by designers as opinions, subject to review and revision by other designers. It also utilizes agents to identify conflicts between designers and to negotiate the resolution of conflicts.

A computer-based design system developed by Sriram *et al.* provides a shared workspace where multiple designers work in separate engineering disciplines (Sriram and Logcher, 1993). In their DICE (Distributed and Integrated Environment for Computer-aided Engineering) program, an object-oriented database management system with a global control mechanism is utilized to resolve coordination and communication problems. Design rationale provided during the product design process is also used for resolving design conflicts (Pena-Mora, et al., 1996; Pena-Mora, et al., 1995).

3.3 Formal models for network-oriented systems

There are several modeling methods/conceptual models for representing and analyzing a system that can be considered of as a collection of simpler subsystems or components. For example, Directed Graph, Petri Nets, Input/output automaton model, PERT chart, Design Structure Matrix (DSM), and Structural Analysis and Design Technique (SADT) are some of the widely used modeling techniques. What distinguishes these methods are the types of the components and the rules for aggregating these components to create system functionality. Depending on the their characteristics, the conceptual models can be generally categorized as

(1) state-oriented; (2) activity-oriented; (3) structure-oriented; (4) data-oriented; or (5) heterogeneous (Gajski, et al., 1994).

A *state-oriented model* is one that represents the system as a set of states and a set of transitions between them, which is triggered by external events. Signal Flow, which is based on the Directed Graph, and Petri Nets are good examples of state-oriented models that are widely used (Desrochers and Al-Jaar, 1995; Eppinger, et al., 1996; Mason and Zimmermann, 1960). An *activity-oriented model* (e.g., PERT chart, SADT, Design Structure Matrix, dataflow graph, Input/output automaton model, etc.) is one that describes a system as a set of activities related by data or execution dependencies (Lynch, 1996; Marca and McGowan, 1986; Steward, 1989; Wiest and Levy, 1969). Unlike state-oriented and activity-oriented models, which primarily reflect a system's functionalities, the *structure-oriented model* focuses on the system's physical composition. Using a structure-oriented model, such as a block diagram, a system's physical modules and interconnections between them can be described. A *data-oriented model* is used to represent a system as a collection of data related by their attributes, class membership, etc. When a variety of different views of a system need to be represented, a *heterogeneous model* can be used.

When a designer creates a DOME model, such as the drill design problem model shown in Figure 3.1, the DOME model reflects the structure of the design problem. For a product design problem model, modules in the model would typically represent different components (both physical and conceptual) and/or design considerations. In this respect, DOME models are structure-oriented. The structure of a model can be the result of the convenient organization or systematic breakdown of the problem based upon rigorous decomposition methodology (Eppinger, et al., 1994).

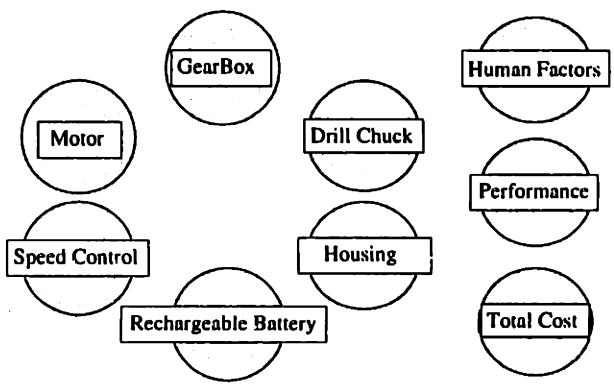


Figure 3.1 DOME model of drill design problem

When a designer connects the inputs and outputs of these modules, dependency relations between modules are created. These dependency relations defined by the designer will dominate the way that DOME performs the computation for evaluating design solution alternatives, since "computing" implies following up the dependency relations to obtain all the necessary information. Therefore, the computational model needs to be activity-oriented.

4

Proposed DOME Framework

The distributed modeling and evaluation of product design problems are realized by extending the concept of the Object-oriented Modeling and Evaluation (OME) framework to the computer network-oriented distributed design environment (DOME)[5].

The main focus of the DOME framework is to enable geographically dispersed designers to model and evaluate integrated design problem models regardless of the location of the designers and models. By providing this ability, it becomes possible to support model-based collaborative design activities between multiple designers and realize a distributed, but integrated, concurrent product development environment. In such an environment each designer focuses on the area of his/her design expertise while the integrated model of the overall design problem can be obtained by aggregating objects or modules that each represent a specific aspect or component of the design (e.g., subproblems, design tasks, different disciplines, etc.).

In this chapter, the requirements of the network-centric distributed design environment for modeling and evaluating product design problems are first discussed, which is followed by the general concept of the DOME framework in terms of two different viewpoints (the designer and implementation viewpoints).

---

[5] The reason for using the term "object-based " as opposed to "object-oriented" is that the DOME framework does not currently address the issue of inheritance and polymorphism, which are fundamental characteristics of the object-oriented methodology. Instead of incorporating all of the characteristics of object-oriented methodology DOME is focused on componentized problem models and interactions between them in the context of computer network-centric modeling and evaluation of product design problems. However, the core of DOME (i.e., Object-oriented Modeling and Evaluation or OME) is based upon the general object-oriented methodology that does address the issues of inheritance and polymorphism as well as encapsulation.

Then, the means of interactions between multiple designers is discussed. The system architecture of DOME and its main components are also presented, which is later used to describe different use modality of DOME. The architecture of DOME is also compared to other architecture and technical approaches.

4.1 Characteristics for computer network-oriented design tools

A computer network-oriented distributed design environment is intended to facilitate interactions between many designers and experts to participating in a product development effort. Participants may be in different locations and utilize different design resources and tools. In this section, what are believed to be key characteristics of the distributed design environment and the requirements of a software system for supporting design activities in this environment are discussed.

4.1.1 Open & interoperable environment

In this environment, it is almost impossible to expect a homogenous environment. Each expert may have their own preferred computer environment and specialized software tools. Therefore, it is crucial for a network-centric design system to support an open and interoperable computing environment. Distributed object technology, such as CORBA and DCOM can be used to address this issue (Brockschmidt, 1995; Chappel, 1996; Siegel, 1996).

A computer language-independent interface definition across different platforms allows software applications to communicate with each other provided a neutral interface has been agreed upon. For instance, the World Wide Web has gained its popularity and momentum through the platform-independent protocol (i.e., HTTP) and a language-independent scheme (i.e., HTML) for presenting information that is easy to understand and use.

4.1.2 Distribution and decentralization

The computer systems for product development are generally stand-alone applications. However, design activities may involve many participants from different domains. Thus, there is also a need to support and coordinate highly distributed and decentralized activities. Distributed design systems might have two distinct forms: distributed designers with access to centralized resources, or distributed designers with distributed resources (e.g., engineering models, databases, software applications, etc.). This work will focus on the later architecture.

The term *decentralized* implies the support for coordination between design participants and models is not centrally modeled or controlled, like the WWW, rather than enforcing strict control in the interaction and participation of distributed resources. This is important because a centralized control over the aggregation of distributed resources to create a integrated model may hinder the scalibility of the model. In the case of the

WWW, if there was a centralized control over linking the hyper documents, it would have been difficult to achieve the complex network of information and rapid growth we have observed.

4.1.3 Knowledge encapsulation & object-oriented approach

Although participants provide services so that an integrated product model can be constructed, it is not likely that each participant will disclose the full details or structure of their proprietary models and data. Providing the means of encapsulating designer's expert knowledge or know-how from other participators is especially of significant when designers from different divisions or organizations collaborate in product design. Thus, it is necessary to allow abstraction in terms of how a participant may interact with others. An object-oriented approach provides a framework for such knowledge encapsulation (Graham, 1994).

4.1.4 Concurrency

Concurrent engineering (or simultaneous engineering) emphasizes the early consideration of downstream requirements (Evbuomwan, et al., 1994). Critical to concurrent engineering is the maintenance of information consistency between participants and the rapid handling, exchange and propagation of design information or events. It is imperative that an integrated design system addresses these issues. Further, in order to reduce training costs and support a broad spectrum of designers, the integrated system needs to provide a consistent system image, which gives the users the impression that they are accessing a single application through a consistent and easy to use interface (Salzberg and Watkins, 1990). In this respect, a common graphical user interface using a Web-browser on the Internet has many advantages. It presents the unified image of a user interface and eliminates the different system images between various computer operating environments.

4.1.5 Collaboration

The design process is inherently a collaborative activity as designers from different disciplines need to communicate and interact with one another. We assume that problems are decomposed into subproblems and the integrated model reflects the detailed and explicit information-flow between the participants or designers in the design process. Some of the key roles in a collaborative process would include a project leader who coordinates design problem modeling with his/her design team members. Each of the team members focuses a particular portion of the subproblem or may utilize models provided by other designers and build his/her own model on top of it.

With regard to the types of design participants and their activities, three different roles are addressed in DOME. The main designer, who manages a specific aspect of a design problem, coordinates the design problem modeling with his/her design team members. Each of the team members focuses on a particular portion of the subproblem while all the members contribute to the overall design problem modeling and

solving in the same scope of the problem. Another type of participant categorizes designers who utilize the models provided by others and build their own models on top of them. This classification of design roles is reflected in DOME as different levels of access privilege which controls and coordinates the modeling activities of design participants. The issue of access privilege in DOME is discussed in section 4.4.

4.1.6 Tradeoff analysis and design decision making

In any design activity, it is important to provide designers with a means to compare design solution alternatives and support the decision making process. In the distributed design environment, since there are multiple number of designers collaborating on a design problem, there is a need to bring the consensus amongst designers over a certain design solution. The conflicts in the design process can be resolved by enforcing designers to accommodate his/her design decisions to resolve the conflicts. Some design systems try to resolve the conflicts by notifying the designers who have made the conflicts and let them determine the neutral solution. In DOME, the system provides a unified design metric which a designer can use to evaluate design solution alternatives from the perspectives of other designers.

4.2 Design activities amongst design participants and their coordination in a distributed design environment

When a number of designers create a model of a design problem by aggregating modules representing various aspects of the problem, the integrated model inherently reflects the detailed and explicit information-flow between the participants or designers in the design process. Therefore, the decomposition of the model describes the interdependencies between tasks as well as how the design tasks are arranged.

This implies that a computer tool created to assist designers needs to be capable of addressing and supporting different types of design procedure and activities. In terms of the coordination of design process, the information-flow in design tasks can be categorized into three types; series, parallel, and coupled. From the computational viewpoint, creating a distributed model involving series and parallel information-flow is rather straightforward. In these cases, a model with many remote modules, whose embedded models are distributed around the network, will be well suited for the actual design procedure.

When there exist complex interdependencies in the design, the development of a design tool that can support the corresponding design activity can be challenging. The system must be able to identify the coupled modules (i.e., in the form of a circular dependency) and to resolve the computational instability while producing reasonable design solutions. This can be done by examining the dependencies defined in the model and apply an appropriate computation strategy. However, if designers in charge of design tasks are geographically distributed and possess proprietary models, it would be more difficult to handle the circular dependency because it is impossible to obtain the global picture of the model.

4.3 DOME framework concept

In DOME, objects (modules) that constitute the model of a design problem communicate with one another through service calls as specified by the OME framework (see Chapter 2). Since the underlying architecture and implementation of DOME relies upon a standard communication protocol that enables interprocess/cross-platform communication, modules on remote machines can be easily incorporated into the local design problem model as long as the host machines of the modules are connected with computer networks[6].

One important requirement of the proposed DOME framework is to maintain the transparency of the modeling and evaluation activities for a designer even if modules provided from remote machines are incorporated into the designer's local problem model. In the following subsections two different viewpoints of a design problem model are discussed. By separating the designer's viewpoint of the model from its actual implementation, the framework allows the designer to utilize remotely created modules in a way that is consistent with locally defined modules.

4.3.1 Designer's viewpoint

An OME module consists of two parts: a module declaration where all the available and required services of the module are described, and an implementation corresponding to the embodiment of the services (the embedded model). In DOME there is the possibility of declaring modules (that is listing the services that each module can provide) without subsequently implementing the embedded models. Once modules are declared with their services defined, they can be interconnected to form a design problem model. However, to evaluate a design solution, all the embedded models must be implemented.

The modules with embedded models implemented in the designer's problem scope are *local modules*. Other modules, whose embedded models are not implemented in the local scope, must be associated with a compatible module (or set of modules) defined elsewhere. These modules are seen as *remote modules* from the local design model viewpoint. In the design problem model the locally declared but unimplemented modules must map to remote modules whose services provide the implementation of their embedded models.

In the drill-level scenario shown in Figure 2.1, the cost, performance, ergonomics and housing are local modules. Their implementations are provided in the local problem scope. The motor, battery, gearbox and chuck modules are only declared through service exchanges with other remote modules. The designer must

---

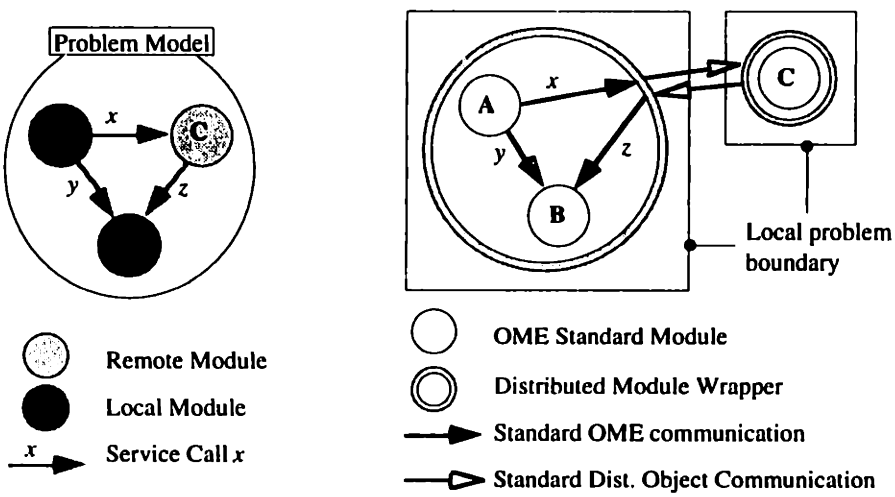
[6] The commercial implementation of the Common Object Request Broker Architecture (CORBA) standard is currently utilized to support the distributed computing capability. The brief background for CORBA is provided in Chapter 2.

specify the information needed to locate the remote modules to obtain a fully functional design problem model.

Finding a compatible remote module may not be an easy task, especially when required services are very problem specific. Standardized service sets specified in a network-based interface definition repository could help in the location of suitable components, but matches may not always be found. Typically, the author envisions that designers developing problem models would consider the interface definitions of remote modules already available and make acceptable tradeoffs between specific modeling needs and module availability. The issue of locating compatible modules can be resolved by adopting the concept of a model repository. The model repository of problem models is discussed in section 4.5.

4.3.2 Implementation viewpoint

The user-visible layer or *modeling layer* is the designer's viewpoint. At this level the designer defines the problem in terms of modules and interactions among modules as shown in Figure 4.1(a). In this example, the implementation or embedded model is provided only for the local modules A and B, while module C is declared as remote. Location constraints are provided by the designer for compatible modules that may be used as C.



(a) Modeling layer (seen by designer/user)

(b) Implementation layer (transparent to designer/user)

Figure 4.1 Modeling and implementation layers in DOME

The unseen *implementation layer* in Figure 4.1(b) is created to provide the functionality described in the modeling layer. This layer locates remote modules. The remote modules must be distributed objects capable of communicating via a standard communication protocol. A distributed interface is wrapped around the group of standard OME modules (A and B in Figure 4.1) to allow the local and distributed modules to

communicate with each other. This distributed module's external interface now offers service calls to and from the remote module. A design problem model sees the distributed module as a separate application that is capable of providing services upon request.

Importantly, encapsulation principles can be applied to hide the details of distributed communication. In the drill scenario in Chapter 2, the Gearbox Manufacturer makes use of distributed modules representing gear components. The gears are distributed modules that establish a bi-directional communication with the gearbox model. The gearbox company defines the complete gearbox design problem, including the gears, as a distributed module so that other designers (e.g., the drill designers) are able to use it as a component. The gearbox module is encapsulated inside another distributed object wrapper and this hides the internal distributed communication between the gear and gearbox. Therefore, the communication with the gear module can be invisible from the viewpoint of the drill design problem.

4.4 Interaction in DOME

The designer's viewpoint discussed in the previous section is further generalized by providing an intuitive way of making one's model available in the design environment and utilizing modules created by other designers. The interaction between different designers through distributed modules is generalized by using the concepts of publishing and subscribing a model

The term "publish" refers to making one's local model visible to other designers. When a model is published, other designers can utilize or "subscribe" the services that the model provides in their local problem scope.

4.4.1 Publish

Before a designer publishes a model to the distributed design environment, it is necessary to assign access privilege for the services that the model provides. In the DOME framework, there will be three types of access privileges; *Owner*, *Builder*, and *User*. *Owner* is the original creator of the model and has access to all the services defined in the model. *Builder* describes a participant in the design problem modeling. Builders can have access to the local problem scope of a model owner and create modules in it. *User* indicates a designer who only has access to the services of a model that has been published.

4.4.2 Subscribe

When a designer subscribes a model available in the distributed design environment, the designer can use the services that the model provides and connect them with the modules in his/her local problem scope. Since the services of a model can be associated with a certain access privilege, the designer can see only the services that are intended to be visible to the designer.

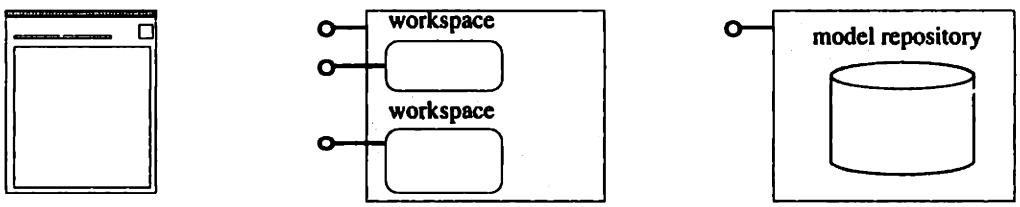
4.5 System Architecture

In order to realize the distributed modeling and evaluation environment as specified in Section 4.1, a client/server-oriented architecture based on the distributed object technology was adopted for DOME[7]. The general configuration of the DOME architecture can be referred as the three-tiered client/server architecture shown in Figure 4.2 (Hines, 1998).

Figure 4.2 Three-tiered client/server architecture

The DOME architecture is designed such that various design activities be feasible in the context of modeling and evaluation of a design problem model. Using DOME a number of designers can participate in the modeling of a design problem model. Also, a designer can reuse design models created by other designers who are dispersed around computer networks.

The main system components of the proposed DOME architecture are the *Graphical User Interface (GUI)*, *modeling and evaluation server*, and *Model repository server* as shown in Figure 4.3.



(a) Graphical user interface (b) Modeling and evaluation server (c) Model repository server

Figure 4.3 Main system components for the architecture of DOME

By arranging the configuration of these components depending on the characteristics of a design problem, designers can create a flexible design environment for both collaborative and cooperative design modeling and evaluation. Since these components interact with one another using a standard communication protocol based upon the CORBA standards, it is not required to maintain all of them on a single machine.

In the following discussions, graphical symbols shown in Figure 4.4 are used to clarify the description of the connection and interaction between system components.

---

[7] The distributed object technology refers to an interoperable computing environment for inter-process and cross-platform integration of software components (Orfali, et al., 1996).

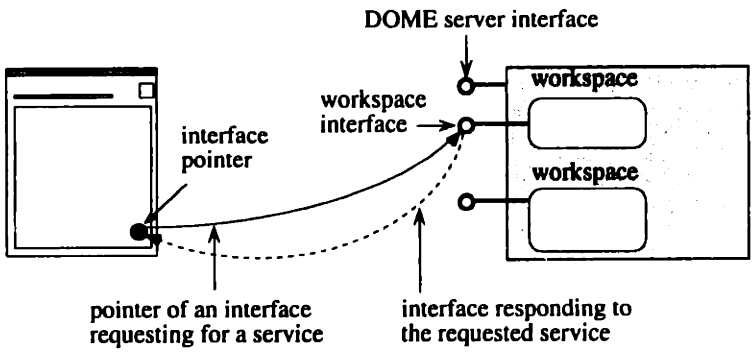


Figure 4.4 Interface and pointer

In Figure 4.4, open circles connected to a system component boundary is called an *Interface*. As a gateway for providing services, the interface of a system component invokes the necessary actions to provide the requested services. To request a service, a system component must have a pointer, or *interface pointer*, to the desired interface (illustrated using solid circles). A solid arrow is used to illustrate a service request between and interface and interface pointer. A dashed arrow illustrates a corresponding response to answer the service request.

4.5.1 DOME Graphical User Interface

The main functionality of the DOME GUI is to enable designers to create, manipulate, and store the model of a design problem. The DOME GUI also provides methods to evaluate design solution alternatives of a design problem and explore a large space of potential design solutions using optimization techniques.

As a system component, a DOME GUI is a pure client of a DOME server. It handles all the events of modeling and evaluation of design problem models by delegating them to an associated DOME server. A DOME GUI may interact with one or more DOME server at the same time.

4.5.2 Modeling and evaluation server (DOME server)

Although a DOME GUI interacts with designers, it delegates the user events and requests to a DOME server, which provides all of the back-end implementation for modeling and evaluating design problems. The core of the DOME server is the *modeling and evaluation kernel*.

A DOME server manages the modeling and evaluation session for each design problem model in a *Workspace*. A server maintains several workspaces at the same time. In addition to the modeling and evaluation of a design problem model, a workspace manages administrative aspects of a model (e.g., ownership, access privilege, links to other workspaces in different DOME servers, etc.). A DOME server itself is a CORBA-compliant distributed object and is able to communicate with other DOME servers using a standard communication protocol.

4.5.3 Model Repository Server

The model repository server (henceforth referred to as *Repository*) maintains the persistent storage for design problem models created in DOME servers. The repository stores a design problem model in the form of a Model Definition File (MDF). MDF is comprised of two parts; meta definition and model definition. The meta definition contains the general information of a model such as model ID, ownership, access privilege information; and the public and private services of the model, etc. The model definition, which is based upon a DOME Model Definition Language (MDL), contains the actual model description that is used to construct a model.

Figure 4.5 illustrates a possible system configuration of the design problem shown in Figure 4.1. In this example, there are three designers who interact with one another using the model of an integrated design problem. As shown in the figure, there are three modules in the model.

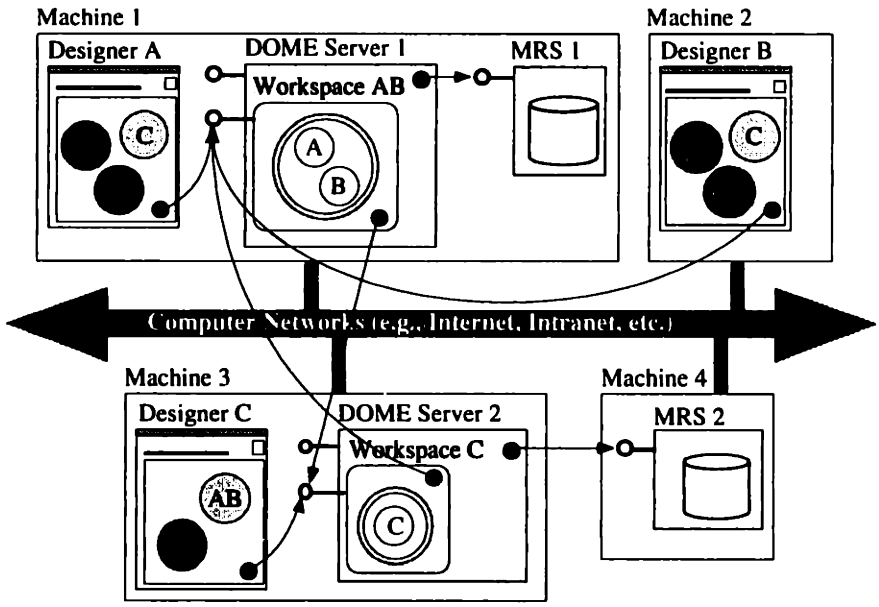


Figure 4.5 System configuration of the example design problem model

Designer A (on Machine 1) is the creator and owner of Workspace AB, which contains a design problem model comprised of module **A** and **B**. In this workspace, Designer A created module **A** and provided the embedded model of the module in DOME server 1. While Module A is created by Designer A, another designer (Designer B) entered to Workspace AB and created Module B. The DOME server keeps track of the information about the creator of a module.

While these two designers create a design problem model in the same workspace, another designer (Designer C on Machine 3) connected to the local DOME server 2 and created Workspace C. In this workspace Designer C declares a remote module named **AB**. Since it is a remote module in this problem scope, it is

not necessary for the designer to provide the embedded model of the module. Instead, being informed that a distributed module with two embedded **A** and **B** is available in Workspace AB of DOME Server 1 on Machine 1, Designer C can contact the workspace and utilize the distributed module as the embedded model of remote module **AB** in his local problem scope.

In the same manner, Designer A can also subscribe a remote module named **C** in his/her local problem scope. By connecting to the distributed module of Workspace C in DOME server 2, the designer can utilize the distributed module as the actual embedded model of **C**.

4.6 Comparison with architectures of other approaches

A distinct characteristic of the DOME architecture is that when designers connect modules available or published over the network it creates an integrated model in the form of a concurrent service exchange network as shown in Figure 4.6, which is referred as *module network*.

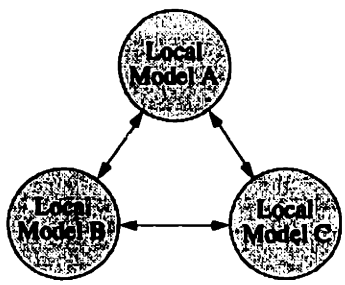


Figure 4.6 Service exchange network

In this configuration, when a designer evaluates a solution alternative of a design problem, the local model asks for services of remote subscribed models to complete the evaluation process. If the subscribed models themselves need services from other models in order to provide the requested services, they will again request those services from their remote models. Therefore, the service request are propagated through the network of connected models. This is distinct from other integrated modeling architectures.

4.6.1 Centralized multi-user system

One of the common system architectures for the network-centric collaborative design system is the *centralized multi-user system* architecture as shown in Figure 4.7.

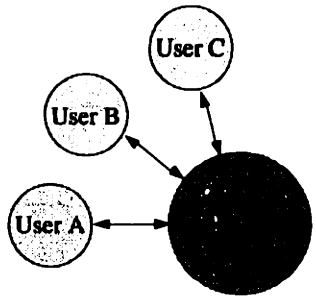


Figure 4.7 Architecture of the centralized multi-user system

In this architecture, multiple users have access to the centralized main system, which stores and manages information such as product design models, design information, design history, etc. The simplest form of this application would be one that allows users to modify the same information one at a time. Therefore, while a user is modifying a certain data, other users cannot make any change.

The more sophisticated applications generally provides a means to manage the version of the information stored in the system. When multiple users make changes in the data or model stored in the main system, the system checks for conflicts between data modifications and mitigates the conflicts or notifies the users in order to resolve the conflicts. In this system architecture, the system may assign levels of modification privilege for different parts of the data or model. Some systems utilize a common workspace called a *blackboard*, where different users can put up data or messages, which is often managed and organized by the main system. The DICE system and DIS proposed by Bliznakov *et al.* are considered to be in this architecture.

When groups of people with their own local computing resources need to participate in an integrated design problem, maintaining a centralized system often has the disadvantage of scaling up the system to integrate various types of models or existing engineering applications. Furthermore, when resources are centralized, it is often difficult to maintain the proprietary information of each participant hidden from one another.

4.6.2 Data and model exchange system

As the globalized open network, such as the Internet becomes a cost-effective infrastructure for the exchange of information, design information systems that allow a large number of users to exchange engineering data or standard product description files has become feasible. The SHARE project by Cutkosky *et al.*, EDN, MADEfast, RaDEO, etc. address this issue of design and manufacturing information exchange. This type of system is referred to here as a *data and model exchange system* and is shown in Figure 4.8.

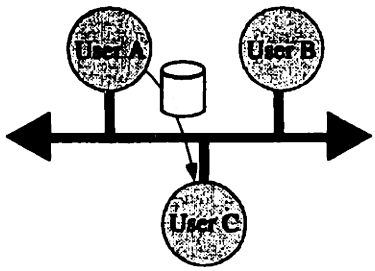


Figure 4.8 Architecture of the data and model exchange system

This system architecture is generally aimed at the rapid exchange of design information. A designer, who worked on a certain aspect of design, passes down the result of the work in a neutral data format (e.g., STEP) to other designers so that they can work on different aspects of the problem and make modifications.

This architecture, however, reflects an "over-the-wall" type of interaction between designers. When a designer receives a model or data from another designer, he/she works on the design and sends the result of design modifications to others. Therefore, it could potentially lead to a locally optimized solution at each user location, which may not be the proper design solution from other designers' perspectives.

4.6.3 Multi agent-based distributed system

Agent-based distributed system architecture in general provides a more abstract level of system interaction and interoperability. An agent is generally referred to a program that behaves at its own preference or logic and performs tasks without direct human supervision (Chen, et al., 1996). There are also several different types of architectures for agent-based systems: agent network, federation, and distributed blackboard architecture (Lander, 1997). From the systems architectural viewpoint, the distinction between DOME and other agent-based systems is the use of distributed-object versus agent[8]. DOME maintains generalized communication objects (i.e., modules) and utilizes explicitly defined messages or services that contains design information. This is referred to as the *communicating objects paradigm* (Bic, et al., 1995). In the *autonomous objects paradigm*, a message has its own identity and behavior as it decides at run time what tasks to perform. Therefore, agents are more appropriate for loosely coupled environments where mutual interactions between objects are not well defined. In case of product design problem modeling in DOME, since the interactions between subproblems are explicitly defined through interfaces of a module, the distributed object is more appropriate for the functionality that the DOME framework specifies. However, for future enhancement of DOME, agents can be useful if a designer is not certain about which module to

[8] It is true that many consider the term *agents* and *distributed object* interchangeably. However, the agent can be considered as a subset of distributed objects with abstract messaging and independent decision-making capabilities.

use and wants to look for an appropriate module. In this case, an agent can be sent out to look for different model repository servers and find a matching module that is requested by the designer.

4.7 Use modality

The general architecture of DOME provides designers with several different ways for participating in design activities. The usage modality of DOME can be categorized into four modes: individual workspace mode, shared workspace mode, service provider mode (for reusing published modules), and service provider/user mode for collaboration. In the following subsections, each of these usage modes is discussed referring back to the simple design problem presented in Figure 4.1.

4.7.1 Individual workspace

As to using DOME, the individual workspace mode shown in Figure 4.9 is the most typical system configuration for modeling and evaluation of a simple design problem model. In this mode, a single designer creates a design problem model in a workspace provided by the connected DOME server.

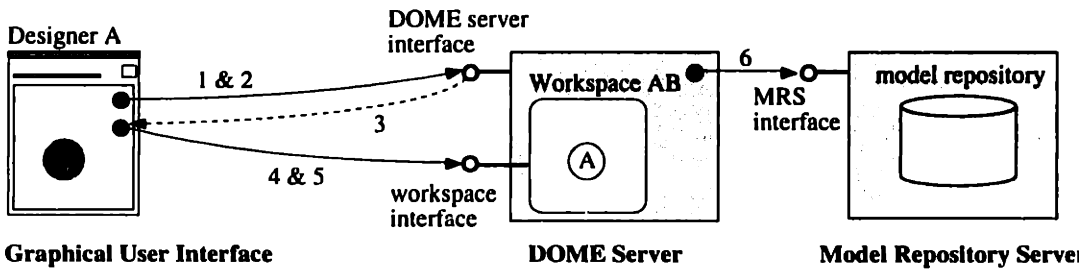


Figure 4.9 Typical configuration of a DOME system for the individual workspace

The general procedure for creating a design problem model in the individual workspace mode is as follows.

(1) Designer A contact to the DOME server in order to obtain the handle of the server interface using DOME GUI.

(2) Designer requests a new workspace, in which a new design problem model can be created and modified. If the designer has previously created and stored a design problem model, the designer can request the DOME server to load the existing problem model. In this case, the DOME server contacts the associated Modeling Repository Server (MRS) to retrieve the model.

(3) After instantiating a new workspace, the DOME Server returns the interface handle of the new workspace to the designer.

(4) The designer creates the model of a design problem through the workspace interface and can evaluate design solutions.

(5) When the model is completed, the designer requests the workspace to store the model.

(6) The request for storing the model is delegated to the DOME server by the workspace and DOME server requests MRS to create a Model Definition File (MDF) and store the model.

4.7.2 Shared workspace

In the shared workspace mode a designer can connect to a workspace created by another designer given the permission to access the workspace. Figure 4.10 illustrates an DOME system configuration that corresponds to the shared workspace mode. In this example, two designers are connected to the same workspace to create a design problem model with each designer responsible for a certain part of the design problem model.

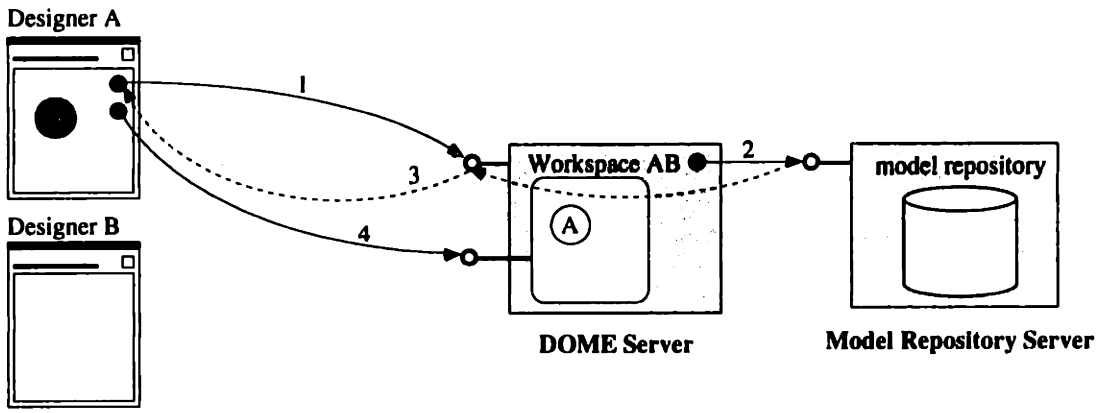


Figure 4.10 A DOME system configuration for the shared workspace mode: part 1

The procedure for opening a design problem model in the local multi-user mode is as follows.

(1) Having obtained the interface handle of the DOME server, Designer A requests the DOME server to load the previously defined design problem model, which is comprised of a single module A.

(2) DOME server delegates the designer's request to MRS to obtain the stored persistent model.

(3) MRS provides the model definition file to DOME Server so that it creates a workspace for the model and pass the handle of the workspace interface to Designer A.

(4) Designer A can modify A or evaluate designer solution alternatives for the design problem model with single module, A.

At this point, the owner of the model (Designer A) must declares the workspace as a shared workspace and specify the list of designers who are allowed to access the workspace.

In this mode, it is important to maintain the ownership of each module. Since each module is primarily owned by one designer, other designers, who create modules for the design problem model, are allowed to utilize the output services that the module provides without being able to modify the module itself. If a

module requires input services to provide output services, only the owner is supposed to select the output services of other modules and make the proper connection with the input services.

With the workspace created by Designer A, the procedure for another designer (Designer B) to access the workspace to collaborate in modeling of the design problem model is as follows.

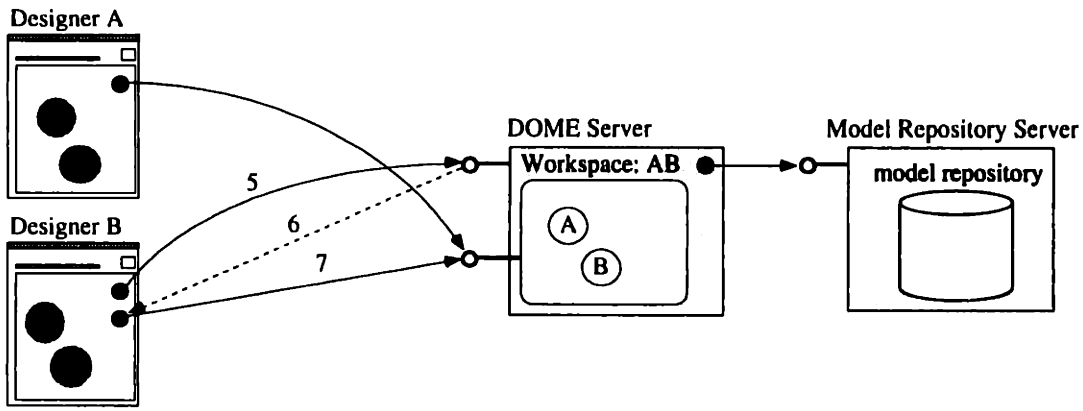


Figure 4.11 A DOME system configuration for the shared workspace mode: part 2

(5) Having obtained the handle of DOME server interface Designer B requests for the interface of workspace AB, where Designer A is currently working. To do so, Designers A and B need to make an arrangement in advance so that proper privilege settings are specified.

(6) DOME server returns the interface handle of workspace AB to Designer B.

*(7) Designer B creates a module, **B**, in the workspace. At this point, Designer B can utilize the output services of module **A** as inputs for module **B**.*

4.7.3 Service provider

In DOME a design problem model can be declared as a distributed module. When a model is declared as a distributed module, it is required to specify the output services that the distributed module can provide and input services required as shown in Figure 4.12.

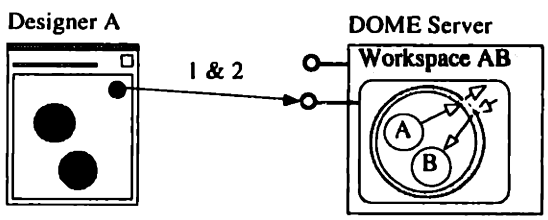


Figure 4.12 Declaring a design problem model as a distributed module.

The general procedure for creating a distributed module is as follows.

(1) Designer A opens a workspace in the DOME server of their local machine and loads up a design problem model comprised of modules A and B. If several designers participated in the modeling of a design problem model as discussed in subsection ***Error! Reference source not found.****, it would be necessary to form an agreement between Designers A and B with regard to the publication of the module.*

(2) The design problem model is declared as a distributed module. When a design problem model is declared as a distributed module, it is required to specify the output services that the module can provides to and inputs required from its users. The line with a hollow arrowhead indicates that the interfaces of the distributed module.

When a distributed module is defined, it is possible to assign different level of access privilege for the services that the module provides. For example, the owner of the distributed module can make the module entirely visible so that any designer can see the embedded models or the module hierarchy of the distributed module. Also, it is possible to protect a certain set of internal modules while keeping another set of modules visible to any designer.

Once a model is declared as a distributed module, another designer can use it as the embedded model of a remote module declared in his/her problem model. Figure 4.13 illustrates that another designer (Designer C) attempts to use the distributed module created by Designer A, who operates on a different platform.

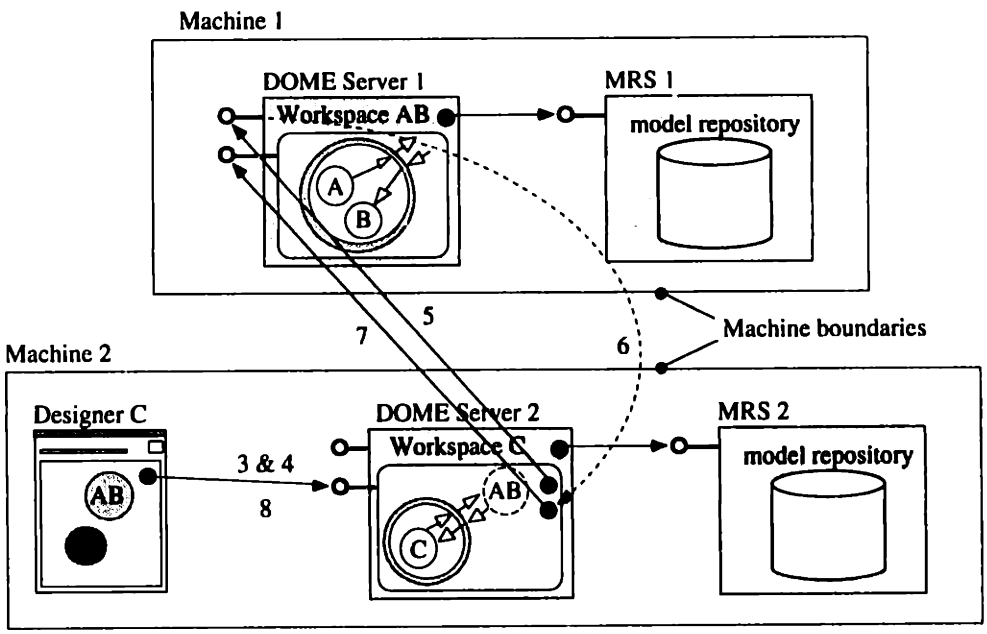


Figure 4.13 Service provider mode for using distributed modules

The following procedure describes how another designer located on a remote host can utilize a distributed module on another host for creating a design problem model.

*(3) Designer C opens a design problem model of module **C** and declares a remote module **AB** in his/her design problem scope.*

(4) After locating a distributed module to use as the embedded model of the remote module, Designer C request for the connection with the distributed module.

(5) Workspace C requests for the interface handle of the workspace which contains the desired model.

*(6) DOME server 1 returns the interface handle of the workspace containing the design problem model **AB**.*

(7) Now connected with the distributed module in Machine 1, workspace C is able to get the services from the distributed module in order to evaluate the design solution alternatives.

When all the remote modules declared in the problem model are associated with distributed modules, the designer can evaluate design solution alternatives of the model. The design problem model with remote modules can be also declared as a distributed module so that other designers can utilized it as a remote module in their problem scope.

*(8) Designer C declares the model with local module **C** and remote module **AB** as a distributed module so that other designers can utilize the model.*

4.7.4 Service provider/user

In the service provider/user mode, several designers can participate in design activities using distributed modules. When a design problem model is declared as a distributed module, other designers can utilize it by requesting the services that the distributed module provides. To declare a design problem model as a distributed module, it is required to define the output services that the module can provide and any input service it requires.

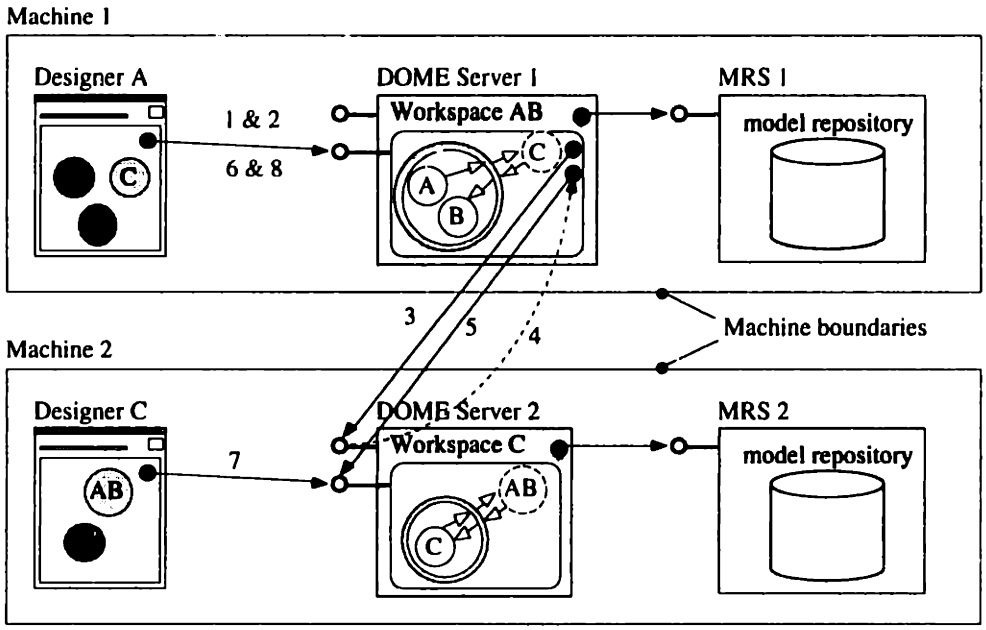


Figure 4.14 System configuration for the service provider/user mode

(1) Designer A connects to the workspace where his design problem model is currently loaded. This workspace is created since another designer (Designer C) is utilizing the distributed module.

(2) In the model, Designer A declares a remote module, C, and request for the connection with the distributed module available on Machine 2.

(3) Workspace AB requests for the interface handle of the workspace that contains the desired model.

(4) DOME server 2 returns the interface handle of the Workspace C to Workspace AB.

(5) Now connected to the workspace of the distribute module C, modules A and B can request services from the remote module. This service request are in fact delegated to the connected distributed module in Machine 2.

(6) Designer A can freely modify the design problem model by changing the values of design variables or selecting a certain catalog items defined in his model.

(7) When Designer C evaluates the design solution alternative, which is influenced by the design solution with respect to the module C, the designer will see the effect of design change made in the distributed module by Designer A.

(8) When Designer A evaluate his design solution, the effect of design decisions made in the model corresponding to the distributed module C will be visible.

5

Formal Studies of DOME Systems

5.1 Motivation of the investigation

In the previous chapter the general concept of the proposed DOME framework and its architecture were discussed. In DOME, a design problem is modeled by aggregating modules and defining how they are related. Each module represents a particular aspect of the design problem and may contain not only engineering models, but also independent computing resources such as databases, engineering application softwares, etc. The relation between modules are defined by specifying which outputs of a module are connected to which inputs of another module.

When a designer creates a design problem model by combining modules, the resulting model itself forms a potentially complex computing system whose computational task is to evaluate design solution alternatives. When remotely located modules are incorporated into the model through computer networks, it becomes a distributed computing system[9]. For systems comprised of interconnected components, the architecture of component interconnections and interactions becomes as significant as the local computational characteristics of each individual component in determining the overall system's characteristics (e.g., computational complexity, scalibility, etc.) (Wegner, 1993).

---

[9] The distributed computing system refers to a system whose computational functionality is accomplished by combining a number of processes communicating with each other. These computational processes often communicate through computer networks while they are running on different platforms.

There are several concerns that may arise when a designer creates a design problem model based upon DOME. When a model is created using a number of modules, how will it behave? When the model becomes larger and more complex, is the model going to behave correctly? Is it possible to scale up the model without significant performance penalties? In this respect, the motivation behind the study presented in this chapter is to understand the behavior of a model that a designer will create using DOME. Among other characteristics of DOME systems, the computational complexity or computation time was the main interest of the investigation.

Another subject discussed in this chapter is the issue of detecting circular dependencies or loops in DOME models. As designers create modules and connect them together, it is often the case that models contain circular dependencies. Circular dependencies in a model are not necessarily inadequate features. On the contrary, they are more often inherent characteristics of a design problem (e.g., design iteration, mathematical model with differential equations). However, if one tries to compute a solution alternative of a DOME model that has a loop without proper adjustment, the system will fall into a infinite loop and cannot produce final solutions. To resolve this problem, several solution approaches are proposed for the future development of DOME.

5.2 Formal description of a DOME model

Computing means being able to specify a finite sequence of actions which lead to the result. What do we mean by action? A precise answer to this question can only be given after having defined a model of computation, that is, a formal system which allows us to express the actions to be performed. In DOME, the action of providing services is performed by modules. When a designer executes the evaluation of a design solution, the DOME system follows the service dependencies defined for the modules. This consequently propagates service requests between modules.

In order to understand the computational behavior of a DOME model (computation time), it is necessary to have a formal notation which will provide the basis for further discussions and studies. The formal notation can be referred to as the computational model of a DOME model. While a DOME model represents how the design problem is decomposed, the computational model reflects how and in which sequence the DOME model will execute the requested computation.

5.2.1 Graphical notation for a DOME model

Module network is a notation for describing a module or model. It has features similar to SADT in the sense that it has a hierarchical representation in its notation and that system components are connected through interfaces (Marca and McGowan, 1986). The notation scheme of a module network reflects the

object model of the DOME framework. It is also the general form of notation that a designer can use when a design problem model is initially created and the layout of the overall problem decomposition is needed[10].

The module network itself is not a modeling methodology, but rather a formal notation for a DOME model. A module network merely shows how a DOME model is configured and how modules in the model are connected. It is more like a conceptual mirror image of the DOME model. A module network itself does not provide analytic methods. However, it can be transformed into a model defined by other formal methods to utilize the analysis tools that those methods provide. For instance, to determine the computation time of a DOME model that does not contain circular dependencies and allows parallel computation it would be more appropriate to see the service request chain in the form of a PERT chart (Wiest and Levy, 1969). In that case, the critical path for the service request chain of the model can be determined and its duration can be interpreted as the expected computation time. For the future enhancement of DOME, it would be possible to represent the interdependencies between module output interfaces in the form of a structure matrix to determine a better way to deal with the iterative computation. This is further discussed in Section 5.4.3.

Figure 5.1 illustrates the module network of a simple design problem model comprised of three modules. The lines between modules indicate that modules are exchanging design information (either directional or bi-directional). A module is indexed using an enumerated subscript[11].

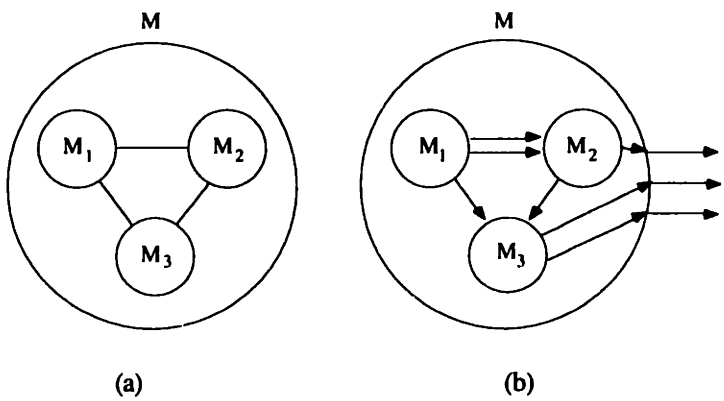


Figure 5.1 Design problem model in the form of a module network

Figure 5.1(b) depicts the flow of design information (through services) between modules. Some modules require the services of other modules as inputs, which are represented by incoming arrow lines. The design

---

[10] As more designers use DOME to create models of design problems, it will become crucial to share a formal notation such as SADT to communicate the functionality of different parts of a DOME model. In fact, when the drill design example was created, several modeling participants created the model description based on the module network formalism.

[11] If the level of the module hierarchy changes, a corresponding subscript is added (e.g., $M_{2,1}$ or $M_{3,2,1}$).

problem model uses services from its embedded modules. When a model is created in DOME, a designer requests the services of the model to evaluate design solution alternatives.

Figure 5.2 shows module M_3, which is embedded in the design problem model shown in Figure 5.1. It shows the interface configuration of the module.

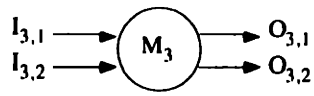


Figure 5.2 Graphical representation of module M_3 with its input and output interfaces

Module M_3 has two inputs (I's) and two outputs (O's). This information is referred to as the *input/output configuration* of a module. A certain output service of a module requires inputs from other modules as well as its embedded model. However, an output service of a module does not necessarily depends on its inputs. Therefore, in order to define the interfaces of a module, it is necessary to provide information about how a module's output relates to its inputs in addition to the input/output configuration. This additional information is referred to as the *input/output dependency*. For instance, when output $O_{3,2}$ depends upon the inputs, $I_{3,1}$ and $I_{3,2}$, as well as the embedded model associated with output $O_{3,2}$, the input/output dependency definition corresponding to $O_{3,2}$ is symbolically denoted as follows.

$$O_{3,2} \leftarrow I_{3,1}, I_{3,2}$$

This implies that, in order to invoke the service $O_{3,2}$, module M_3 is required to first obtain inputs to the module, $I_{3,1}$ and $I_{3,2}$. After obtaining the required inputs the module utilizes its internal data and/or embedded model to perform the computation needed for the service $O_{3,2}$. Therefore, the computational characteristics with regard to $O_{3,2}$ depends upon those of its associated inputs as well as the embedded model of M_3.

While $O_{3,2}$ depends both on $I_{3,1}$ and $I_{3,2}$, $O_{3,1}$ may have a different input/output dependency. For example, it may depend only on $I_{3,1}$, which can be denoted as follows.

$$O_{3,1} \leftarrow I_{3,1}$$

The fact that a module may have several different input/output dependencies differentiates the module network from conventional graphs, where an output of a node (i.e., an arc incident from a node) implicitly depends upon inputs to the node (i.e., arcs incident to a node)[12].

Figure 5.3 shows the graphical representation of the input/output configuration of module M_2.

---

[12] This characteristics does not offer an advantage over the conventional graphs, but merely represents how a DOME model is configured.

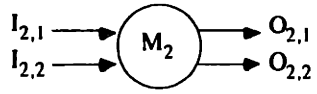


Figure 5.3 Graphical representation of module M_2

As shown in Figure 5.1(b), modules M_1 and M_2 are connected. Since modules are connected only through interfaces, the connection information associated with modules M_2 and M_3 can be symbolically denoted as follows.

$$I_{3,2} = O_{2,2}$$

This relation is referred to as the *input/output connection* with respect to modules M_1 and M_2 and is depicted in Figure 5.4.

Figure 5.4 Graphical representation of an input/output connection

Using this graphical notation, the design problem model in Figure 5.1 can be represented to show the detailed input/output connections as shown in Figure 5.5. In addition to the topological information (i.e., number of modules, input/output configuration and connection information), the input/output dependency information is needed to complete the description of the design problem model.

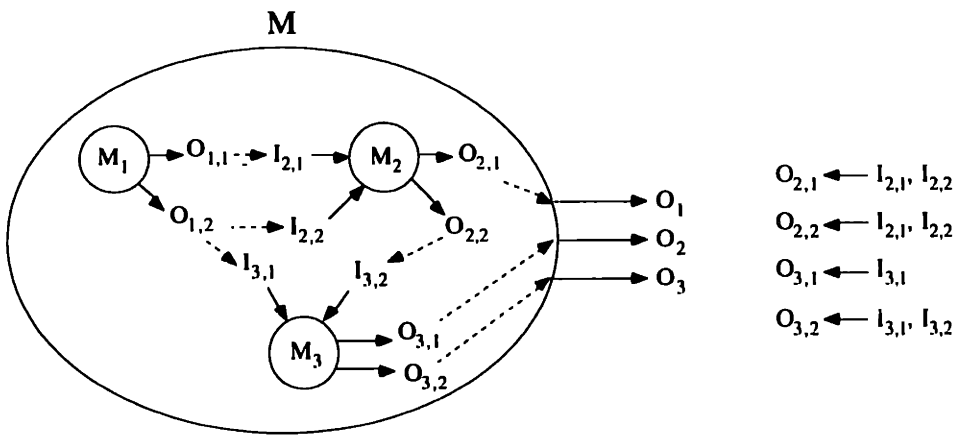


Figure 5.5 Graphical model representation with input/output connection details

To remove the input/output dependency information given in Figure 5.5, the modules with multiple input/output dependencies, can be decomposed as shown in Figure 5.6.

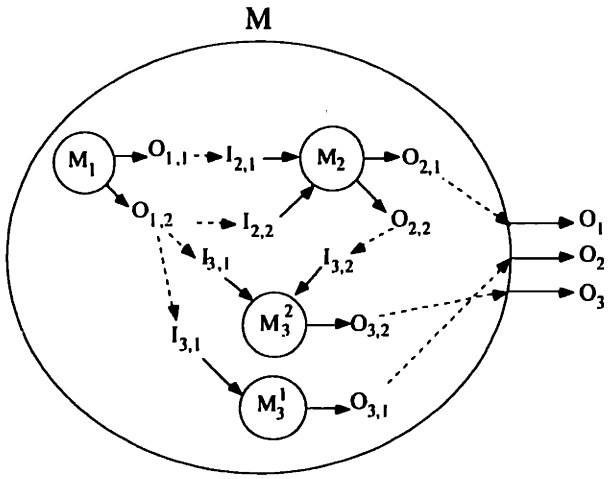
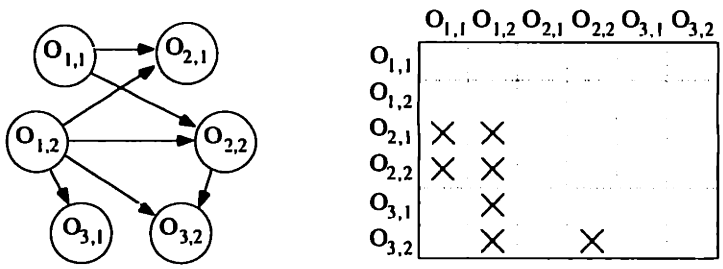


Figure 5.6 Graphical model representation with decomposed modules

When the modules are decomposed such that each module has a unique input/output dependency information for its output interface, the resulting network becomes a conventional graph. By denoting output interfaces of the network as nodes and dependencies between output interfaces as arcs, the module network can be converted to the conventional directed graph models or adjacency matrix to show the dependencies between output interfaces as shown in Figure 5.7.



(a) Directed graph (b) Adjacent matrix

Figure 5.7 Alternative representation of the module network shown in Figure 5.5

Figure 5.7(a) is the directed graph representation of the dependencies between the modules' output interfaces. In this representation, the output interfaces of the model (i.e., O_1, O_2, and O_3) are not included because the dependencies between the output interfaces of embedded modules (i.e., M_1, M_2, and M_3) are sufficient to address issues such as circular dependencies. Figure 5.7(b) is the adjacent matrix representation of the directed graph in Figure 5.7(a). The ability to convert the module network, which reflects the configuration of an actual DOME model, into different types of models will be advantageous if the DOME system can provide alternative problem solving strategies based upon the configuration of the model (e.g., the interdependencies between design parameters)

5.2.2 Simple design example for using the module network formalism

Figure 5.8 illustrates the input/output configuration and dependency information of each module in a simple motor-battery problem. It shows how each module can be denoted based upon the formalism discussed earlier. This representation of a module itself reflect the implementation of the module within the DOME system. If a module is provided from a remote site, this information is used by the module user to identify whether or not the module has matching interfaces and which inputs are required to provide the wanted services. Once modules are connected, the computation for the evaluation process is performed based upon the dependencies defined for the interfaces.

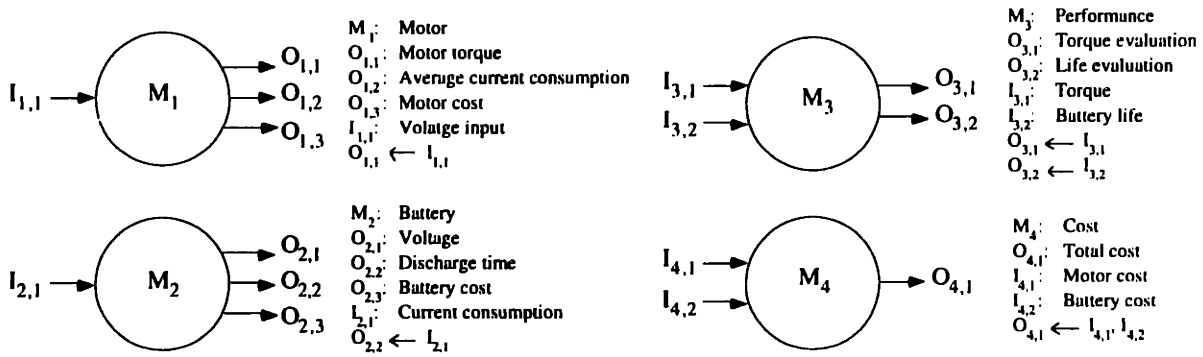


Figure 5.8 Input/output configuration and dependency information of the modules

Having obtained the modules, the designer needs to connect the modules by defining the connections between the interfaces as shown in Equation 5.1, which results in the complete design problem model shown in Figure 5.9.

$I_{1,1} = O_{2,1}$ **Equation 5.1**

$I_{2,1} = O_{1,2}$

$I_{3,1} = O_{1,1}$; $I_{3,2} = O_{2,2}$

$I_{4,1} = O_{1,3}$; $I_{4,2} = O_{2,3}$

$O_1 = O_{3,1}$; $O_2 = O_{3,2}$; $O_3 = O_{4,1}$

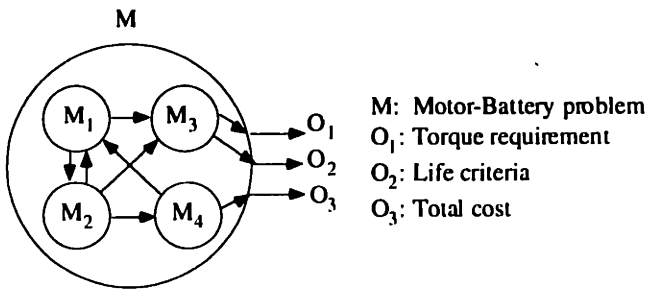


Figure 5.9 Representation of the Motor-Battery problem model in DOME

5.3 Complexity measure of a DOME model

The complexity of a computing system usually refers to the computing performance of the system in producing desired results. The time complexity, which is the time taken to produce the outputs, is often used as a complexity measure (Lynch, 1996). The time complexity of a DOME model is defined as the time that it takes to complete one evaluation.

5.3.1 Computational characteristics of an output interface

A computational model of DOME is obtained by associating each output interface of a module with computational characteristics. The computational characteristics of output interfaces are then combined with the configuration of the design problem model to determine the computational characteristics of the overall system. The time complexity measure, $T_{i,j}$, of the output $O_{i,j}$ of module M_i is depicted as shown in Figure 5.10.

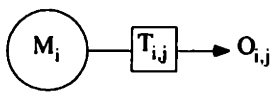


Figure 5.10 Output property of a module

As shown in Equation 5.3, the computation time ($T_{i,j}$) to produce a result through output interface $O_{i,j}$ can be divided into two parts: the time ($Tc_{i,j}$) taken to complete the computation provided all the inputs are already obtained and the communication delay (Td_i) in delivering the output (due to network latency, network instability, etc.).

$$T_{i,j} = Tc_{i,j} + Td_i \qquad \textbf{Equation 5.2}$$

In Equation 5.3, the communication delay (Td_i) is a characteristics of the module rather than each output interface. When inputs are connected to other modules' output interfaces, the total computation time (accumulated up to $O_{i,j}$) is denoted as $C(O_{i,j})$ and defined as follows.

$$C(O_{i,j}) = T_{i,j} + C(\mathbf{REL}(\mathbf{DEP}(O_{i,j}))) \qquad \textbf{Equation 5.3}$$

where **DEP**(x) is an operation that returns a set of interfaces upon which interface x depends and **REL**(**X**) is an operation that returns a set of interfaces to which interfaces in set **X** are connected.

5.3.2 Estimated computation time of design evaluation

The current implementation of DOME performs the computation of a design evaluation sequentially for a design problem model assuming there exists no circular dependency. This means that, although the service

requests can be made in parallel to reduce the overall computation time, the current DOME system invokes the service requests sequentially[13]. However, DOME implements an efficient service request method, referred to as the *Update mechanism*, to improve the efficiency of the computation. Using the update mechanism, DOME prevents multiple invocation of the same service, whose upstream design parameters have not been modified since the latest service request. Therefore, the expected computation time of a design problem model without a circular loop can be written as shown in Equation 5.4.

$$T_{total} = \sum_{k=1}^{u} C(O_i) = \sum_{i=1}^{n}\sum_{j=1}^{u_i} T_{i,j} = \sum_{i=1}^{n}\sum_{j=1}^{u_i} (Tc_{i,j} + Td_i)$$

where

n: number of internal modules **Equation 5.4**

u: number of output services of the model

u_i: number of output services of the i^{th} internal module

The computation time of a model can be determined by obtaining the computation time of each output service of the model (i.e., design evaluation services). Then, the expected computation time can be obtained by accumulating the computation time of all the output interfaces. Obviously, this will also be the upper bound of the computation time even if the parallel computation strategy is adopted to improve the computational efficiency.

When the parallel computation strategy is implemented in the future, the service requests will be simultaneously deployed and the overall computation time can be significantly decreased. In this case, the time complexity of a DOME model can be determined by utilizing the concept of task duration along the critical path as discussed in PERT chart method can be used. This implies that the slowest service will dominate the complexity measure of a DOME model.

---

[13] In the Web-based DOME, we are planning to implement parallel service requests for remote modules in order to improve the computational efficiency. This can be accomplished by adopting a multi-thread service invocation algorithm.

5.3.3 Statistical verification of the complexity measure

Several DOME models are tested to validate the complexity measure in time that is discussed in the previous section by comparing them with the actual computation time. In Table 5.1, the statistical results for the configuration and computational characteristics of DOME design problem models are provided. For the statistical results of the actual computation time, twenty executions were performed for each model. The brief background of the examples used here is provided in Appendix A.

		number of local modules	number of remote modules	number of design variables	average computation time (ms)	variation (% of the average)	expected computation time (ms)	Percent error (%)
1	Simple model 1	2	0	8	270.1	0.0	270.2	~ 0
2	Simple model 2	2	0	13	566.6	0.6	550.1	1.2
3	Truss	14	0	67	1019.0	24.8	1233.1	21.0
4	EM shielding	28	0	188	5164.0	17.2	4289.2	16.9
5	Combined cycled elect. generator	11	1	58	1992.0	46.1	1760	11.6
6	Cardboard container	7	1	53	2459.0	19.6	2152.3	12.5

Table 5.1 Comparison between the expected and actual computation time

The expected computation time of a DOME model is obtained by determining and summing up the computation time of each module's output interfaces based upon Equation 5.4. The result indicates that the proposed complexity measure does provide a reasonable estimation of the actual computation time of a DOME model, and that the update mechanism of DOME imposes a linear characteristics with respect to the computational scalibility. Therefore, when a designer aggregate modules to build a model, the computation time of the model can be estimated by adding the computational complexity of each module in the model. Before investigating the computational characteristic of a DOME model, the concern was whether or not the complexity (i.e., computation time) of a model grows exponentially as more modules are integrated.

In Table 5.1, the percent error of the complexity measures for examples 1 and 2 are small since the type of their design variables (i.e., deterministic or probabilistic) do not change for different solution alternatives. Therefore, it is possible to obtain fairly accurate estimation for their actual computation times. On the contrary, the percent error for examples 3 and 4 are relatively higher because their design solution alternatives contain both deterministic and probabilistic design variables. The high percent errors for examples 5 and 6 are due to the variation in the remote communication time and the performance of remote modules.

In the case of DOME models with remote modules, as Equation 5.4 implies, the communication delay time associated with the remote modules contributes to the overall computation time. With simple but reasonable closed-form approximation for the overall computation time, the designer can estimate how long the evaluation or optimization process of a design solution alternative will take under a certain network condition or communication delay. For example, the estimated computation time of the combined-cycle electricity generator and cardboard problems can be obtained based on Equation 5.4 by examining the computation time of a module's each output interface and the communication delay associated with the module as shown in Equation 5.5.

$$T_{total}(\text{Combined-cycle elect. generator}) = 2147.3 + Td_{simulation\ module}\ (\text{ms}) \qquad \textbf{Equation 5.5}$$

$$T_{total}(\text{Cardboard container}) = 1125 + Td_{TEAM\ module}\ (\text{ms})$$

Based on Equation 5.5, it is also possible to estimate the evaluation time of a design solution alternative when different communication media or collaboration tool is utilized. Figure 5.11 illustrates a hypothetical situation where designers evaluate a design solution alternative of the design problem model using DOME, E-mail, and telephone.

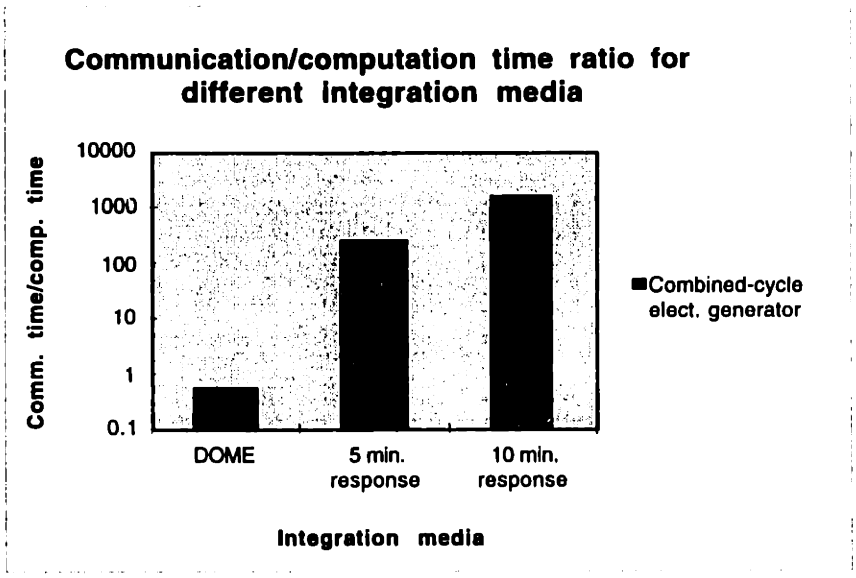


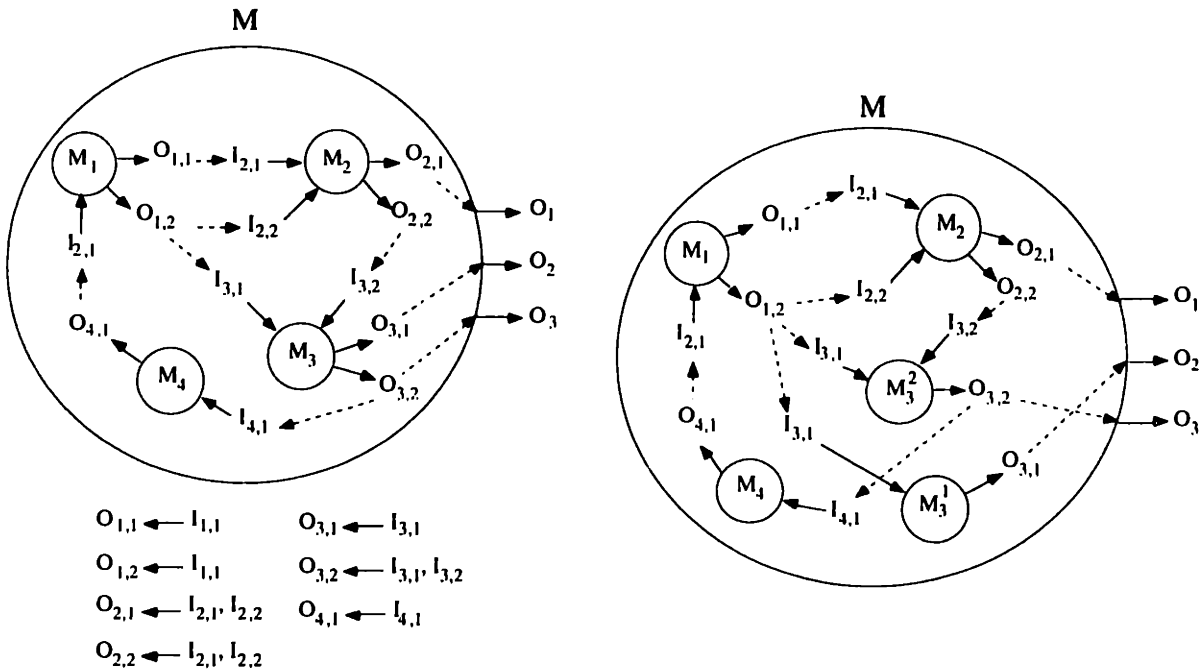
Figure 5.11 Communication/computation time ratio for different integration media

5.4 Issues of circular dependencies in a model

In DOME the evaluation process of a design solution is performed by following the dependency relations between services in order to obtain all the necessary services for the evaluation of a design problem solution. Therefore, if there exists a circular dependency, the evaluation process will not be completed as it falls into an infinite service request loop. It is necessary for the DOME framework to provide a robust solution to this problem.

5.4.1 Detection of loops

In order to handle circular dependencies or loops, the DOME system first needs to identify them in a design problem model created in DOME. Figure 5.12(a) shows an example module network of a DOME model with circular dependencies. The circular dependency is created by adding a module (M_4) to the model shown in Figure 5.5. The decoupled module network is also shown in Figure 5.12(b).



(a) Module network with I/O dependency information (b) Decoupled module network

Figure 5.12 Model with a circular dependency

Once a model is represented as a module network and the configuration information is available, there are several ways to determine the existence of circular dependencies in the model.

5.4.2 Using the incidence matrix

From the decoupled module network shown in Figure 5.12(b), the incidence matrix in Equation 5.6 can be obtained. A brief description of the incidence matrix is provided in Appendix B and, for the mathematical implication of the incidence matrix, readers are referred to (Carre, 1979).

Equation 5.6

$$A = \begin{array}{c} \\ M_1 \\ M_2 \\ M_3^1 \\ M_3^2 \\ M_4 \\ \hline O_1 \\ O_2 \\ O_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} O_{1,1} & O_{1,2} & O_{2,1} & O_{2,2} & O_{3,1} & O_{3,2} & O_{4,1} \end{array} \\ \left[\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ \hline 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{array}\right] \end{array}$$

Since the evaluation of the design solution of the model will be performed by calling three services of the model (i.e., O_1, O_2, *and* O_3), the sequence of the evaluation process can be determined by following the dependency chain in the matrix. The procedure to obtain the service request chain from the incidence matrix of the module network is provided in Appendix B. When the service is requested to the same output interface more than once in the same path, it is considered as a circular dependency. By ending the path that has a repeated service request with the italic denotation of the interface, the service request chain can be obtained as shown in Figure 5.13.

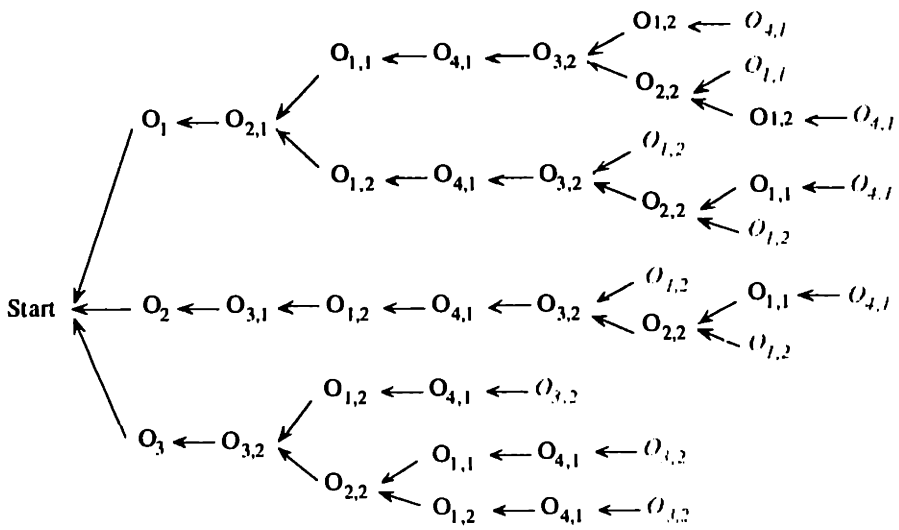


Figure 5.13 Service request chain tree for a module network with circular dependency

To implement the loop checking algorithm when the module configuration and dependency information are given, matrix manipulation techniques from conventional graph theory can be applied. This is currently

being implemented to assist designers in modeling a design problem model. Using the input/output configuration, dependency, and connection information, the system can avoid potential problems due to a loop in the model by notifying the designer.

However, when distributed modules are used in a DOME model, the information of module decomposition and interdependencies is not sufficient to determine the existence of loops in the model since the global topology of the problem or module connection is difficult to obtain. Finding loops in such a case is a subject of future research.

5.4.3 Using the adjacency matrix

From the decoupled module network shown in Figure 5.12(b), the adjacency matrix can be constructed as shown in Figure 5.14. When the module network is represented as an adjacent matrix, loops in the network can be identified by applying a path searching algorithm or powers of the adjacent matrix (Gebala and Eppinger, 1991). Although this is not further investigated, the adjacent matrix representation can also be useful when the dependency relations between output interfaces can be weighted based on a certain performance aspect. Then, based upon the interdependency between output interfaces and tearing strategies, the more effective way of resolving the computation with circular dependencies can be applied.

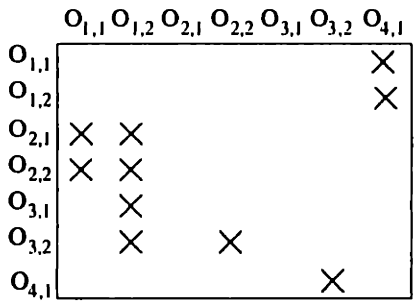


Figure 5.14 Adjacency matrix of the module network shown in Figure 5.12

5.4.4 Computational strategies for a model with circular dependencies

To enable the DOME system to handle the circular dependencies in a design problem model, several computational strategies can be considered. As briefly discussed in the previous section, the DOME system first needs to determine whether or not there exist circular dependencies in the model. This can be done by applying loop detection algorithms to the model of a DOME model as discussed earlier.

Breaking a loop

The easiest and simplest way of preventing the infinite loop is to break the loop intentionally. By inspecting the service request chain initiated by starting the evaluation of a design solution alternative, one

can determine where service are requested for the second time in the same path, which is illustrated in Figure 5.13. For those services, initial or guessed values can be passed to the modules that requested the services. This approach may require several iterations before discrepancy between the guessed values and the actual computation results converges.

If the modules which are tightly coupled and creates circular dependencies, they can be separated and grouped into a module, which is referred to here as *Phantom module*. The Phantom module then utilize an iterative computing strategy to obtain a certain level of convergence within itself before providing its services.

For example, a simple design problem example model with loops shown in Figure 5.15(a) can be transformed into the model shown in Figure 5.15(b) by grouping the modules (in gray) whose interface connections are creating circular dependencies.

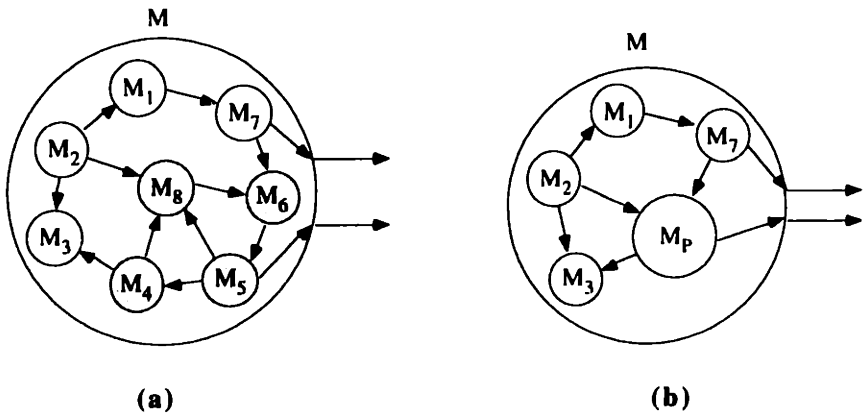


Figure 5.15 Simple problem model with loops

Simultaneous execution

The limitation of the solving algorithm in dealing with circular dependencies is largely due to the parametric solving approach in the current implementation of DOME. We are currently considering to adopt variational solving techniques, which would take the inputs to a module and performs the simultaneous solving to produce the output services of the module.

6

Implementation

The implementation of the DOME framework proposed in this thesis is an on-going effort. The initial DOME prototype was implemented in C++ on SGI platforms with IRIX® 5.3[14] in the MIT Computer-Aided Design Laboratory (CADlab). The DOME prototype has been used by both academic researchers and industrial product design engineers to model and analyze a variety of engineering design problems such as electromagnetic shielding, combined-cycle electricity co-generator, product LCA-related design problems, etc. The initial prototype clearly illustrates the concept of the design environment and shows the potential contribution to the community of product design and development.

However, it still has some limitations in supporting the collaborative design activities discussed in Chapter 4 and the accessibility of the system due to the fact that the operating environment of the DOME prototype is limited to SGI's IRIX operating system environment. For this reason, the implementation effort (henceforth referred to as Web-based DOME) has been moved to a WindowsNT®-based environment with a Web-based Graphical User Interface[15]. The Web-based DOME provides much more flexibility in terms of both modeling and system accessibility.

In this chapter, the original DOME prototype is first presented to show the various capabilities of the system. Most of the design problems modeled in CADlab have been so far created using this system (Borland, et al., 1998; Jackson and Wallace, 1997; Jackson and Wallace, 1997; Pahng, et al., 1997; Pahng,

[14] IRIX® is a registered trademark of Silicon Graphics Inc.

[15] WindowsNT® is a registered trademark of Microsoft Inc.

et al., 1998; Senin, et al., 1997). Then, the on-going implementation of the Web-enabled DOME is presented and the issues related to the current implementation are discussed.

6.1 Prototype of the DOME Framework

The modeling core of the prototype is based upon a previously developed OME library (Senin, et al., 1996). In order to search for optimal design alternatives, the prototype utilizes an optimization tool based on Genetic Algorithms (Gruninger and Wallace, 1996; Wall, 1996). Also, the current implementation incorporates an ASCII-based Model Definition Language (MDL) to facilitate the rapid creation of modules and problem models (Borland, 1997).

The CORBA standard for distributed computing environments is utilized to add distributed communications capabilities to modules. The CORBA implementation used in the prototype is Orbix® 2 from IONA Technologies Ltd. (IONA, 1997). CORBA serves as an information and service exchange infrastructure above the computer network layer and provides the capability to interact with existing CAD applications and database management systems through other Object Request Brokers (ORB), which is based on the CORBA standard (Siegel, 1996). In turn, the DOME framework provides OME with the methods and interfaces needed for interaction with other modules in the networked environment.

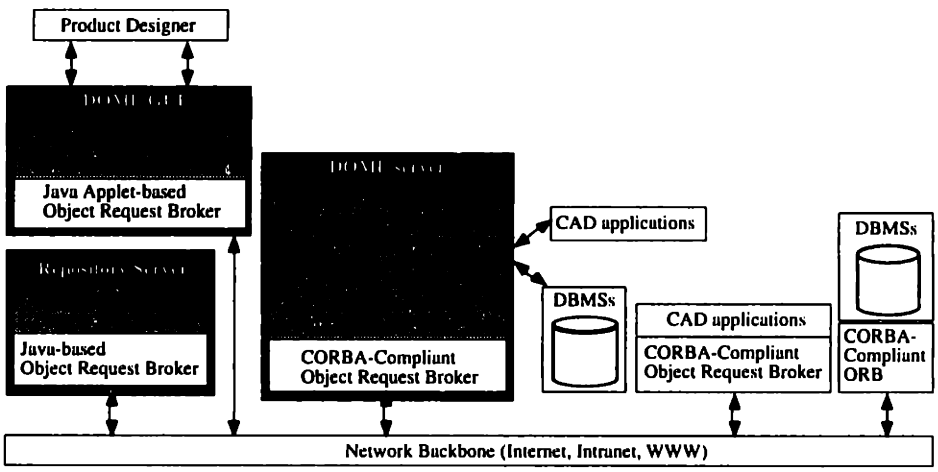


Figure 6.1 Architecture of the proposed open design environment based on DOME

Figure 6.1 illustrates the architecture of a computer-aided product development system to provide an open integrative design environment. In this environment designers can integrate DOME-based design problem models with existing application packages. The system components in gray represent those which have been prototyped in the CADlab as p .rt of the framework.

A graphical user interface provides users with the ability to examine the configuration of design problem models, analyze tradeoffs by modifying design parameters and search for alternatives and improved design

solutions using an optimization tool. The graphical user interface is implemented using the X windows system with Xmotif.

6.1.1 System Capability

Figure 6.1 shows the graphical user interface of the DOME prototype. The model shown here is the drill division's design problem model of the drill design scenario presented in Chapter 2.

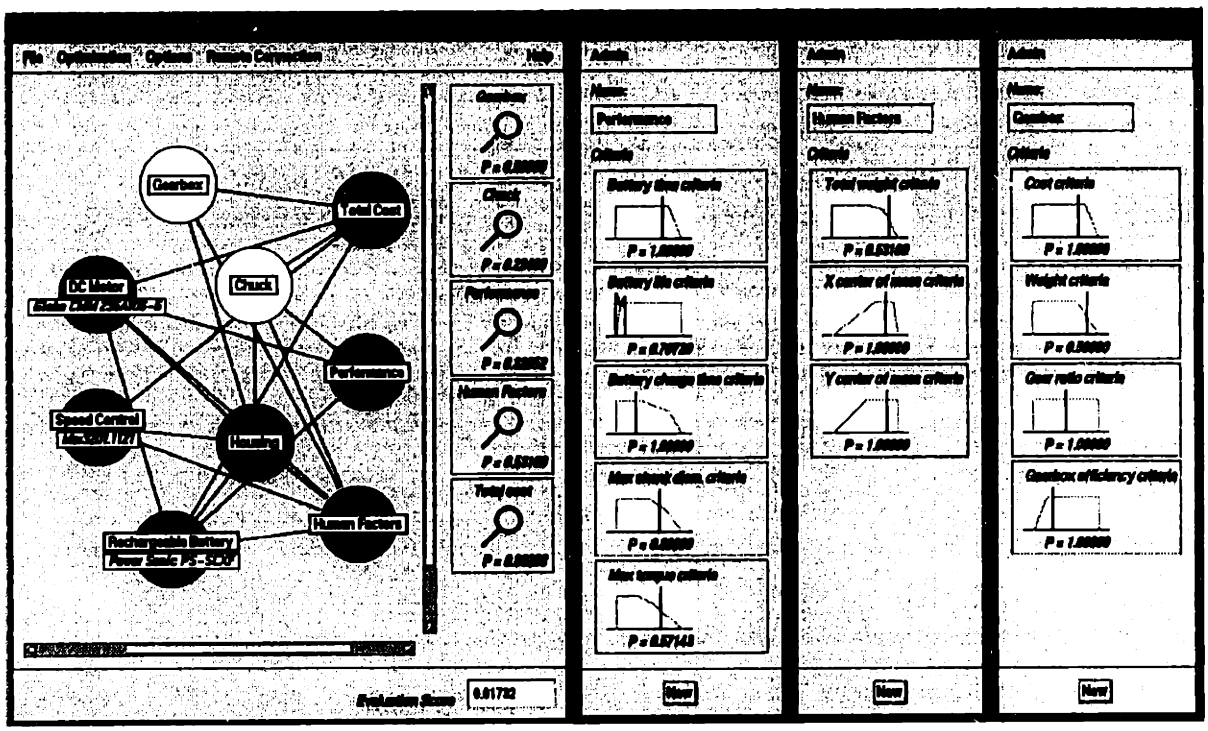


Figure 6.2 Graphical User Interface of the DOME prototype

Object-oriented modeling environment

The main window on the left shows the design configuration. Each circle represents a module. Interactions among the modules are represented by lines that link modules together. Since modules interact only through their interfaces, a module may contain an existing engineering application (e.g., motor performance analysis tool) or product data base (e.g., database for various types of rechargeable batteries and their characteristics).

Catalog selection

The motor, speed control and battery are selections from catalogs of modules. In DOME, any discontinuous parameters are modeled in form of catalogs. Using a catalog browsing window shown in Figure 6.3 a designer can easily make an alternative choice in the catalog and see the effect of the design change.

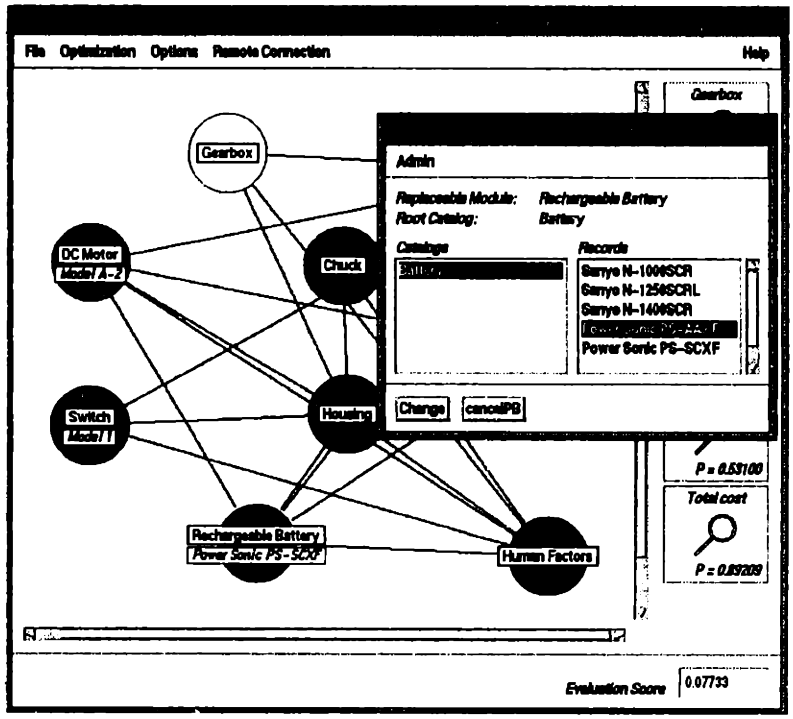


Figure 6.3 Catalog browsing window for the rechargeable battery

Remote modules

The white module in Figure 6.3 indicates that it is not defined in the local scope. This is a remote module. For instance, a designer in the drill design division can create a model of the drill without creating a gearbox model with all the modeling details, but instead simply connect to a model provided by the gearbox design division.

Specification-based probabilistic design modeling and design evaluation

The right side of the main (leftmost) window in Figure 6.2 shows an overview of the lenses (evaluation services) corresponding to modules in the design problem. The three windows on the right side of the figure show the contents of three lenses (performance, human factors, and gearbox). Each thumbnail in a lens shows the expected design *performance* (black) superimposed with the corresponding designer's *specification* (gray).

The number under each criteria indicates the score from the evaluation service (in this case the probability that the performance is acceptable based upon the specification). The score under each lens represents how well all criteria within a particular viewpoint are satisfied. In the bottom right of the main window, a number between 0 and 1 represents the overall evaluation for the integrated design model, indicating the likelihood of meeting all design objectives. Using the DOME GUI a designer can interactively modify design variables, change catalog selections and alter specifications.

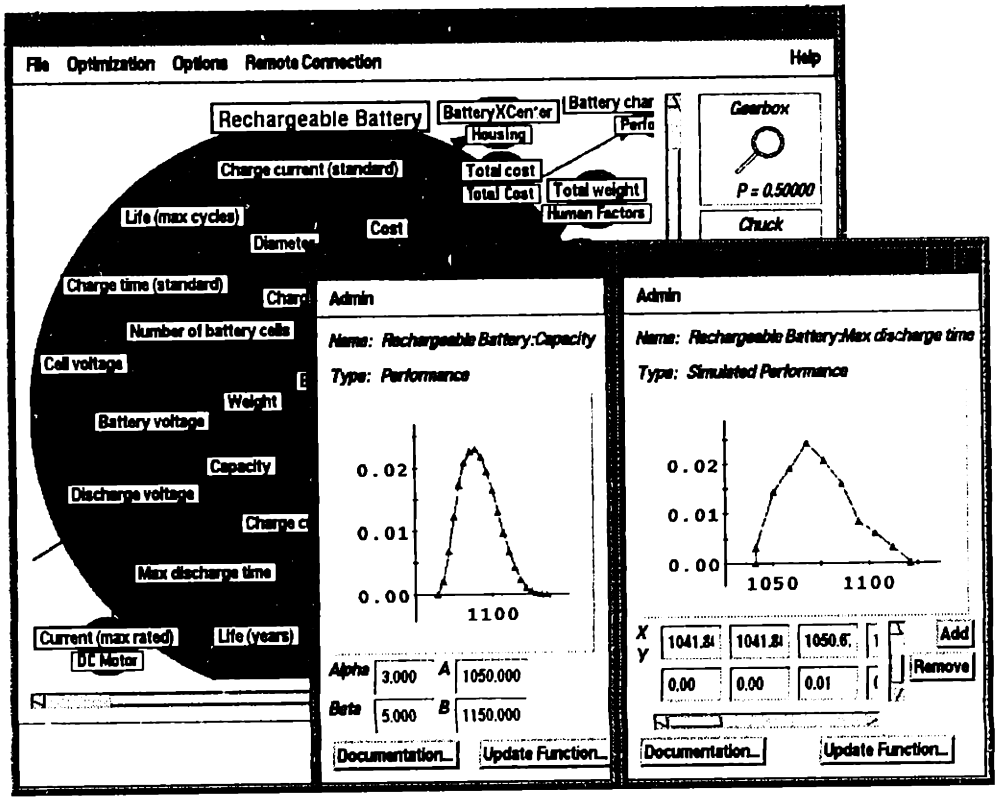


Figure 6.4 Embedded model of the rechargeable battery module

Embedded model

Figure 6.4 shows a visual representation of the embedded model for the rechargeable battery module. Small circles (gray) are design variables and the lines indicate the existence of relations. Dark gray variables are derived from relations. Circles outside the module boundary are inputs and outputs to and from the module through its interface. As shown in the figure, a design variable can be represented by a probabilistic value that indicates the probability density function with respect to the expected values of the design variable.

The windows on the right of the figure show the battery's capacity (represented as a beta function) and its predicted maximum discharge time. The discharge time is dependent upon the battery's capacity and the power requirements determined by other parts of the model. A designer can interactively change the value of the discharge time and see the effect of the design change on the predicted maximum discharge time. The battery discharge time has been generated using Monte-Carlo simulation. Since variables have the ability to determine if they need to perform simulations based upon their inputs, the designer does not worry about creating a engineering model or formula that calculates the result from the mix of deterministic and probabilistic values.

Optimization

Figure 6.5 shows the optimization window of a design problem modeled in DOME. Using the optimization window a designer can select a set of continuous independent design parameters and a set of module catalogs to be included in the search space. The optimization process window shows the evolution history of the population of solution alternatives.

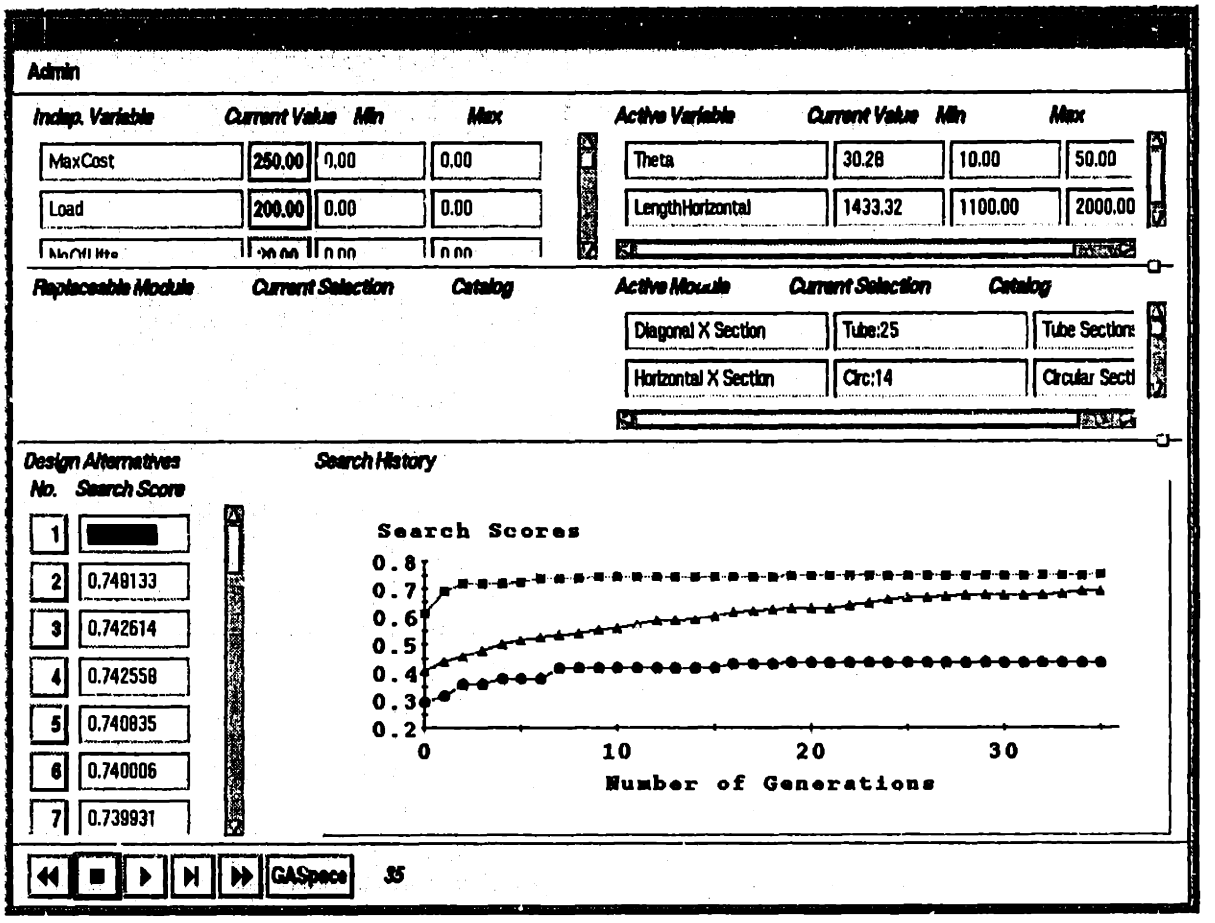


Figure 6.5 Optimization process window of a DOME design problem model

6.1.2 System Elements

When a designer creates a design problem model using remote modules, the resulting integrated model forms a distributed design system. Figure 6.6 shows the three main elements in such a system: a design problem model definition, a DOME graphical user interface, and a distributed design object.

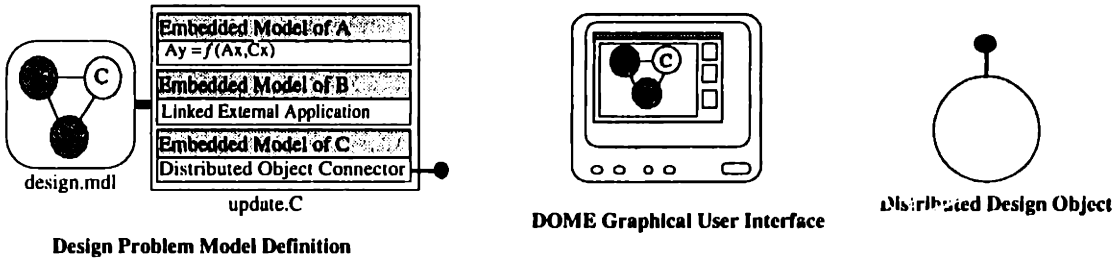


Figure 6.6 Implementation elements of the DOME framework

Design Problem Model Definition

A designer creates a design problem model by describing (in MDL) how different design variables, subproblems, and evaluation aspects of the problem are related. This model information is stored in an MDL source file (*design.mdl*). The designer can provide the embedded mathematical models either directly in MDL files or in separate embedded model resources provided as DOME-specific source files (*update.C*). The ability to separate the implementation of an embedded mathematical model from the definition or configuration of a problem model is essential for the transparent implementation of distributed modeling capabilities. When a remote module is declared within a problem model, its corresponding embedded model will have appropriate methods which implement and mediate its connection to distributed design objects. These methods form the distributed object connector shown in Figure 6.6.

DOME Graphical User Interface

Once a design problem model is defined, it can be visualized using the *Graphical User Interface* (GUI). The GUI provides the ability to explore the design problem from different perspectives (e.g., cost, performance, environmental aspects, etc.) and analyze tradeoffs (what-if scenarios). A designer can also optimize design variables (both discrete parameters and catalog selections) to generate design solution alternatives.

Distributed Design Object

If a designer chooses to make his/her problem model accessible for use as a distributed module, he/she can create a *distributed design object* from the design model. A distributed design object has several CORBA-compliant object interfaces, to which its clients can connect. When designers create a CORBA-compliant design object, they must determine the required inputs and desired outputs of the distributed module and create a corresponding interface definition (in Interface Definition Language (Siegel, 1996)).

The interface is then mapped to a problem model using the MDL interpreter and the modeling and evaluation kernel. When clients connect to these distributed objects, the design problem model definitions are loaded so that clients can invoke the services they offer.

6.1.3 Building a Simple Problem Model

Modeling Layer

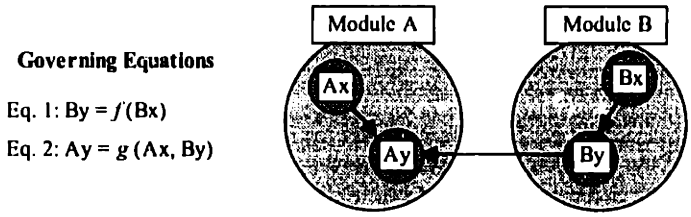


Figure 6.7 Simple problem model composed of two modules

The model of a design problem is created by decomposing the problem into modules and defining how modules are related to one another. The relationships amongst modules specify how outputs of a module are connected to inputs of other modules. The embedded model of a module will produce outputs using its internal design resources as well as inputs from other modules. Figure 6.7 illustrates the model of a simple problem with variables governed by a set of equations.

The model for the example shown in Figure 6.1 is defined below using the Model Definition Language (MDL).

```
Module "Module A" (
    Variable "Ax" ( Deterministic (8.0) ) // this variable is a single value
    Variable "Ay"    // the type of this variable is determined by the system
                     // based upon the types it depends upon
        Dependency "ax"
        Dependency "by"
        EmbeddedModel "calculateAy" ( Ay = 1/(1 + log(ax)) + by )
)
Module "Module B" (
    Variable "Bx" ( Beta (3, 2, 5, 8) ) // this variable is a beta distribution
    Variable "By"
        Dependency "bx"
        EmbeddedModel "calculateBy" ( By = pow(bx,3) )
)
Design "simple problem" (
    Module "Module A"
    Module "Module B"
)
```

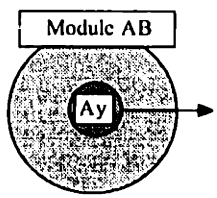


Figure 6.8 Distributed module for the simple design problem example

This illustrates how modules are defined. The interface connections between variables in different modules as depicted in Figure 6.7 are established interactively though the GUI. However, they may also be explicitly defined in the MDL. The embedded models, defined with the variable declaration in this example, could also have been created separately and linked to the model definition using keywords.

Modules A and B are local to the same design problem scope. Now a distributed problem model will be created. The problem model in Figure 6.7 will be made available for use as a distributed module with the outward appearance of that in Figure 6.8. This distributed module will allow users to utilize *Ay*.

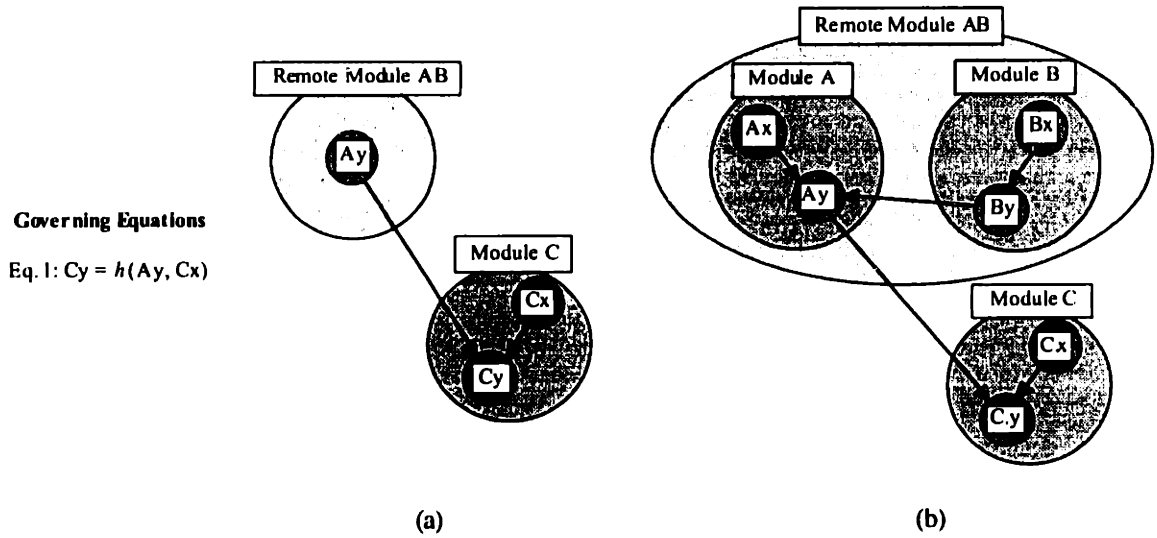


Figure 6.9 Simple design problem example with a remote module

Figure 6.9 shows a new design problem model (ABC) which uses remote module AB. Figure 6.9(a) illustrates the model from the viewpoint of the ABC designer. Module C is local to the designer. Figure 6.9(b) illustrates the true integrated model created when remote module AB and local module C are connected.

The problem model ABC is created below. It requires additional information such as the distributed module's name and IP address.

```
RemoteModule "Module AB"
    "Remote Module AB" "cadlab6.mit.edu"
(
    Receive "ay"
)
Module "Module C" (
    Variable "Cx"
    Variable "Cy"
        Dependency "cx"
        Dependency "ay"
        EmbeddedModel "calculateCy"
            ( Cy = pow(cx,3) + ay)
)
```

```
Design "simple problem with a remote module" (
    Module "Module AB"
    Module "Module C"
)
```

The example shows that the relations between modules do not need to be changed even if the embedded mathematical model of a remote module (i.e., module AB) is changed. This flexibility enables a designer to define a model independently from the actual location (i.e., local or remote) of embedded models. This example involves uni-directional information transfer but in many instances, as in the drill example in the following section, bi-directional transfer between distributed modules is required.

Implementation Layer

When the designer utilizes remote module AB in conjunction with the local module C, the resulting integrated model forms a distributed computing system comprised of two autonomous computing elements. Figure 6.10 illustrates the configuration of the integrated model using the system components described in Figure 6.6. The embedded model of the module AB in design problem model ABC contains an object connector which manages the design information exchange with the distributed design object AB. However, this underlying communication mechanism is transparent to the designer. Figure 6.10 also suggests that designers are currently viewing problem models at both locations using the GUI. However, the GUI at AB is not required for the evaluation of problem ABC.

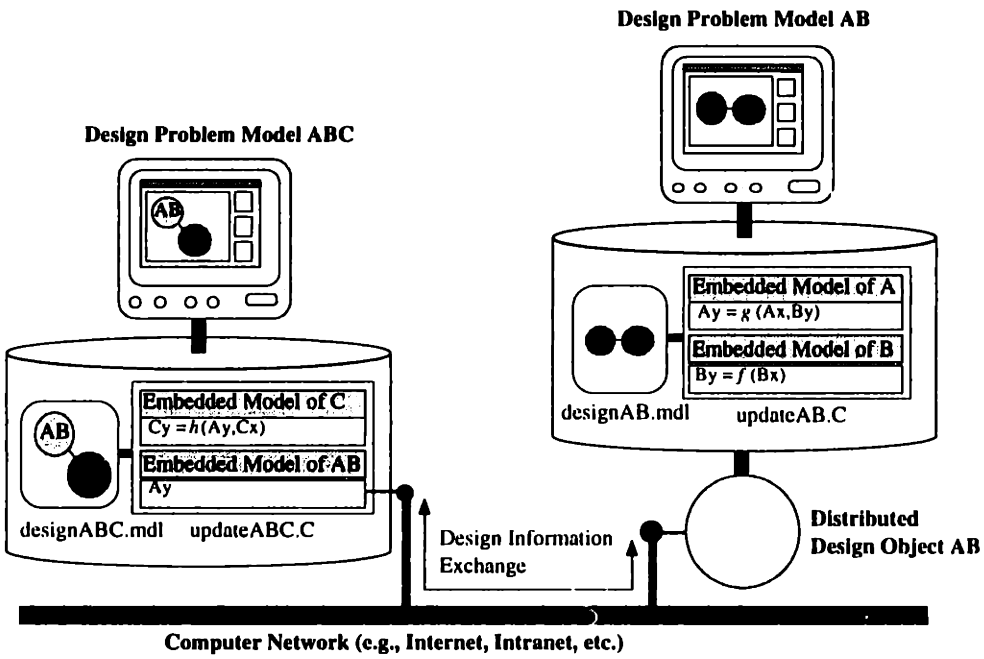


Figure 6.10 System configuration of the simple design problem

6.2 Web-based DOME

The implementation of the DOME framework presented in the previous section is now enhanced to achieve the collaborative interactivities between designers discussed in Chapter 4. The main improvement of the system is that the front-end implementation is entirely Web-based and the three-tier client/server architecture discussed in Chapter 4 better supports the collaborative designer interactions proposed in the DOME framework.

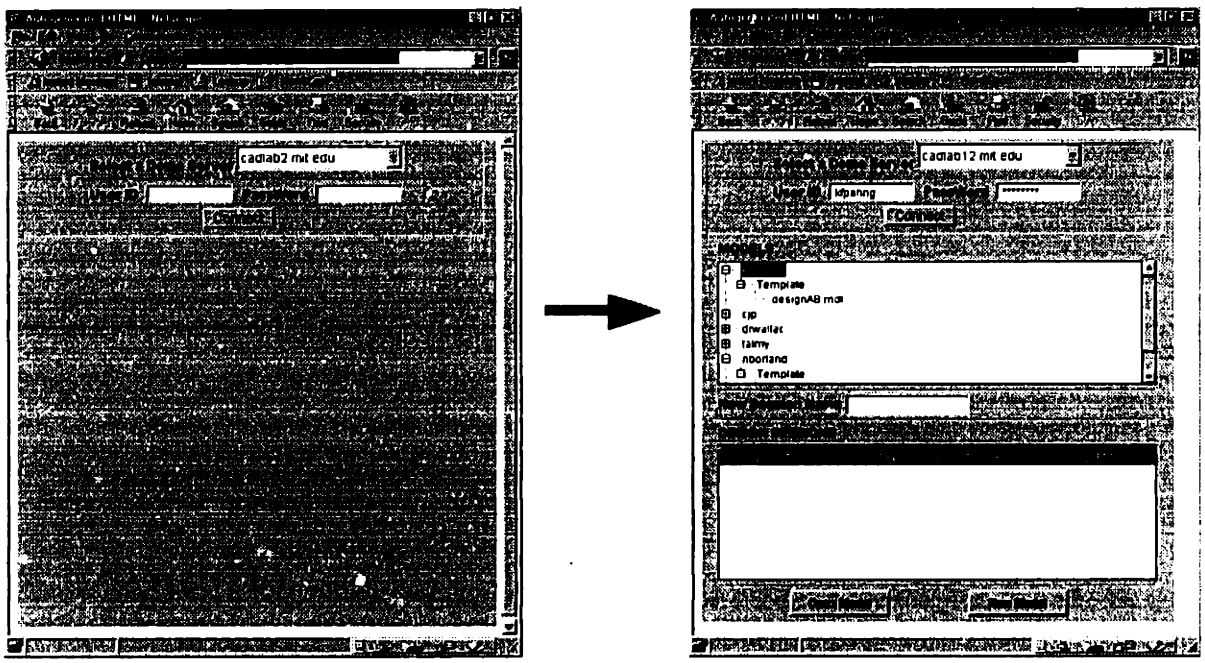
6.2.1 Implementation of the DOME system client/server architecture

In order to improve the accessibility and interoperability of DOME, the graphical user interface, in which designers create models and evaluate design solution alternatives, is implemented as a Web-based application. The front-end side of the application is implemented as the combination of HTML documents and Java applets. The front-end Java applets allow designers to browse the DOME server and repository in a arbitrary host machine and to participate in an on-going design session or select an existing model.

For the CORBA-based remote communication between the Java applets of the DOME GUI and the back-end side system components (DOME server and repository), a commercial ORB implementation of Java applets (i.e., OrbixWeb®) is utilized[16]. Since Java applets are compatible with any operating environment, a designer can use a Web-browser (e.g., Navigator® or Explorer®) to access DOME systems from any type of computer. The back-end implementation of DOME is implemented both in C++ and Java. The DOME server that provides the CORBA interfaces to the front-end GUI is written in Java.

---

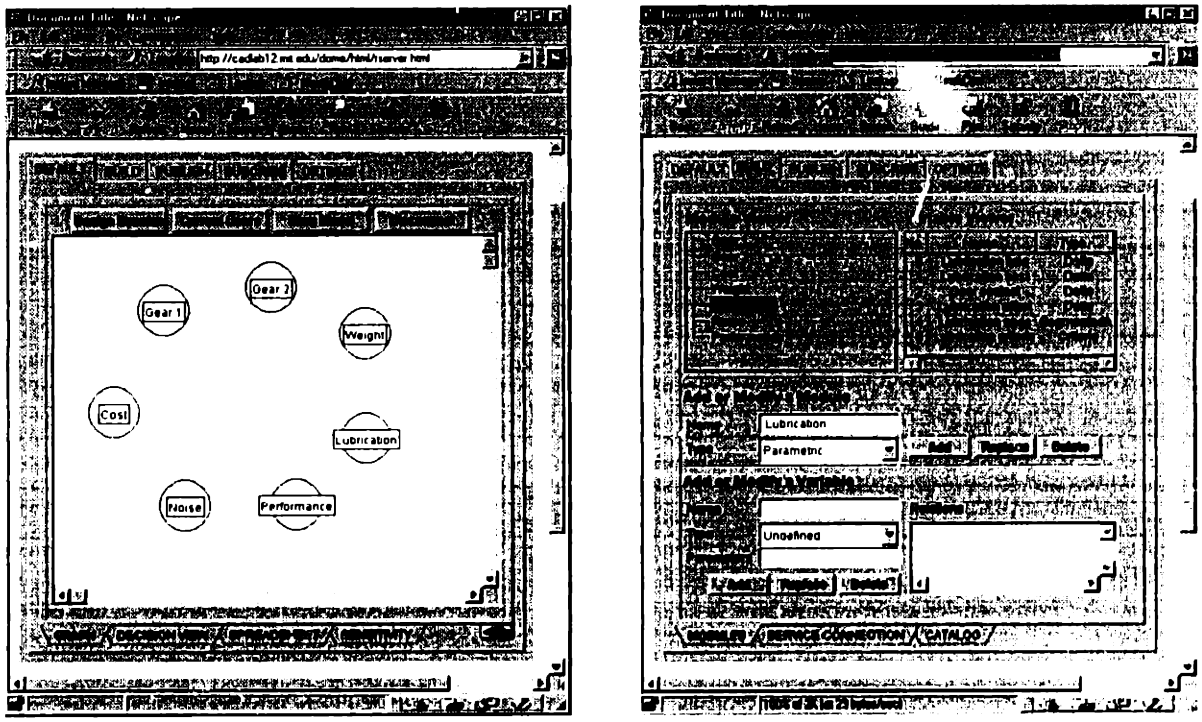
[16] OrbixWeb is a registered trademark of IONA Technologies Inc.



(a) Initial window (b) User login approved

Figure 6.11 DOME GUI for the workspace access

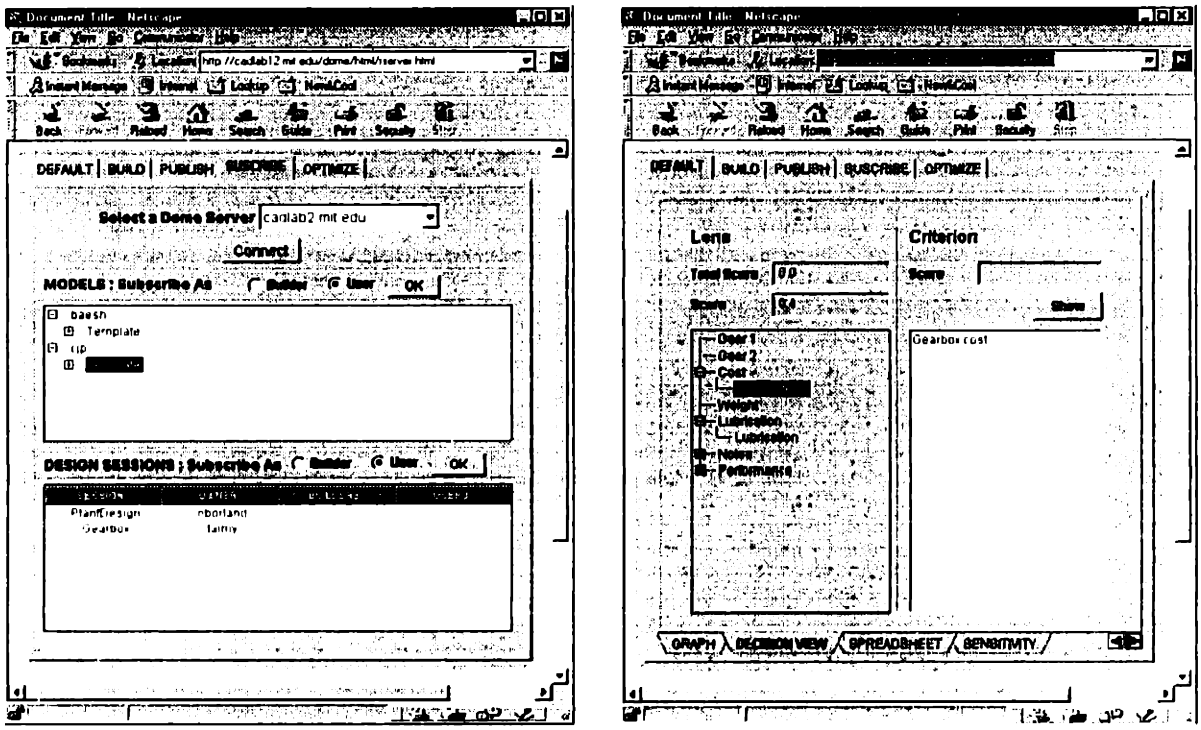
Figure 6.11 shows the initial entry Web site of the Web-based DOME. Using this GUI a designer can browse a remote site where a DOME server or repository is hosted. Once a designer successfully enters the host after the login procedure (Figure 6.11(a)), the window (Figure 6.11(b)) will display all the available models stored in the repository and currently running design sessions where designers are working on design problems.



(a) (b)

Figure 6.12 DOME GUI for problem modeling and evaluation

Figure 6.12 shows the main DOME windows, where designers builds design problem models and evaluate design solution alternatives. In Figure 6.12(a), the graphical representation of a DOME model is shown. In this window, a designer can browse through different modules and see how they are related. Figure 6.12(b) shows the editing window, where the designer can add, delete, and modify models.



(a) (b)

Figure 6.13 Web-based DOME design windows

Figure 6.13(a) shows a window where a designer can browse through the DOME server and model repository on different hosts. This functionality enables the designer to look for an appropriate module available on the distributed design environment in order to use in his/her local problem scope.

Shown in Figure 6.13(b) is the evaluation window of the DOME system. Using this window, a designer can go through different evaluation viewpoints or lens to understand the performance of the current design from various perspective (e.g., cost, safety, maintenance, environmental issues, etc.).

7

Example Problem Models

This chapter provides example design problem models created using the DOME system. The first part of the chapter provides the implementation of the drill design scenario presented in Chapter 2. It clearly shows that how designers in different design teams or companies can combine several design models to create an integrated design problem model. The example also shows how the design change made at the detail level of the system design (e.g., gear design parameter) propagates to the system level and how the modification of the design goal (e.g., drill designer's design specification on the gearbox weight) affects different levels of the product design.

Another example, a beverage container design problem, is presented to show the integrative modeling environment of DOME. In this example, the design problem model is integrated with existing engineering applications such as Ecobilan's environmental assessment package, TEAM®, ProEngineer® and Excel® (Borland, et al., 1998).

The second part of the chapter presents design problem examples implemented using Web-based DOME, which provides richer context of functionalities in terms of supporting collaborative activities in design process. Web-based DOME addresses several issues of coordinating collaborative activities. The examples show how the system can support and coordinate the interaction between dispersed designers working on an integrated design problem.

7.1 Implementation of the Drill Design Scenario

The drill design scenario presented in Chapter 2 has been implemented as a distributed design problem using the DOME framework. Networked computers each simulate a company or division participating in the distributed modeling of the drill design problem as illustrated in Figure 7.1.

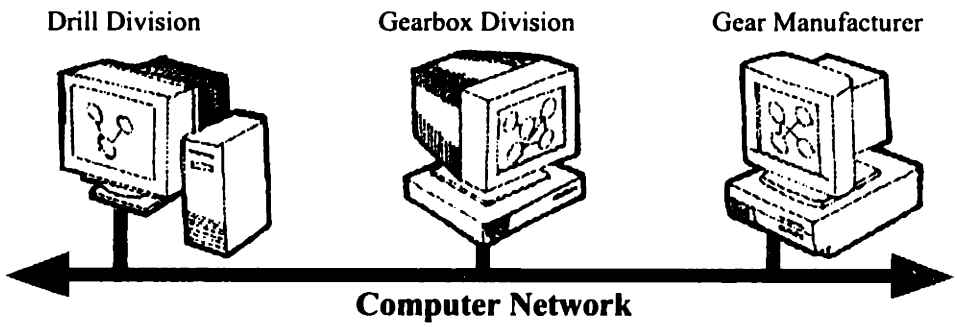


Figure 7.1 Configuration of the drill design scenario

7.1.1 Drill Division's Perspective

The DOME model shown in Figure 7.2 is the system level drill design problem in Drill Division. Using the DOME model drill designers can evaluate design solution alternatives for their design at the overall system level. As shown in the figure the gearbox module (light gray) is a remote module seen as a single module, whose embedded model is provided by Gearbox Division. The drill designers can in particular evaluate the performance of the gearbox design with respect to the design goals that they have specified using the Gearbox lens (the rightmost window). Also, any design change made at Gearbox Division will propagate to other aspects of the drill related to the gearbox design (e.g., battery life, total weight, etc.).

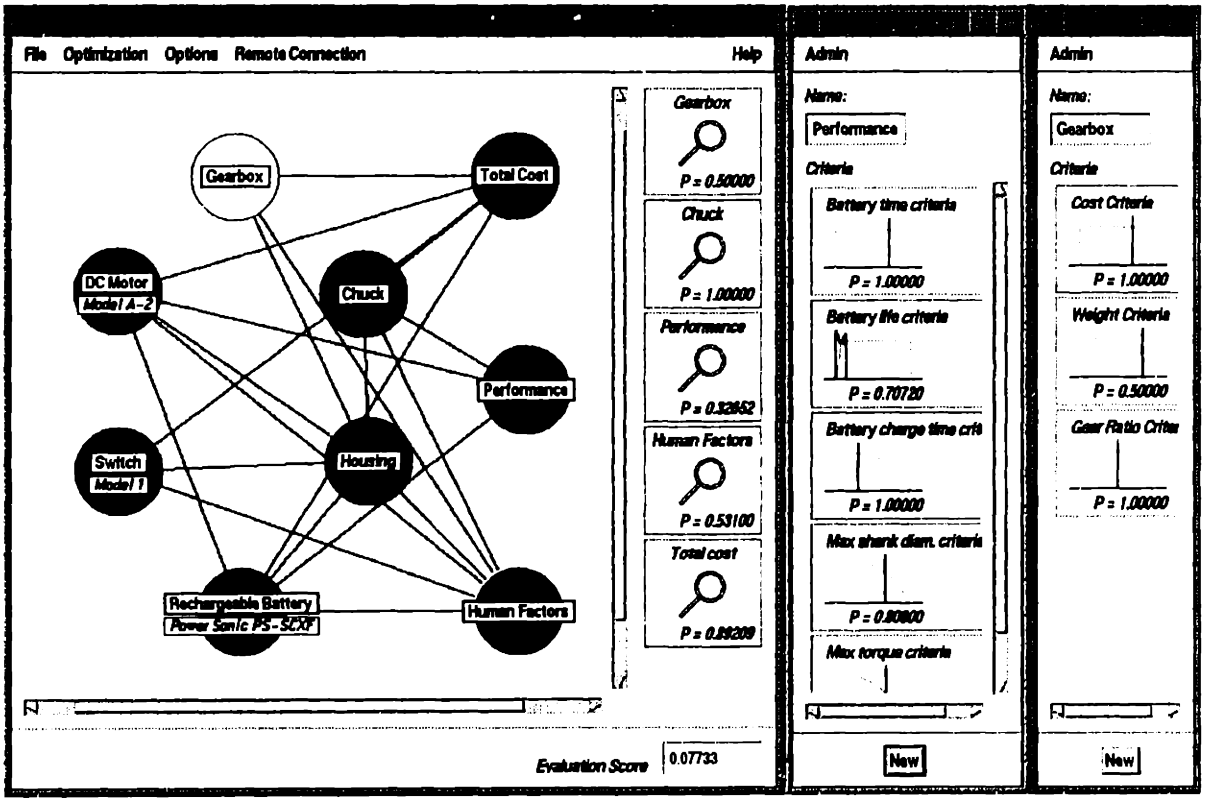


Figure 7.2 Drill design problem model from the Drill Division's viewpoint

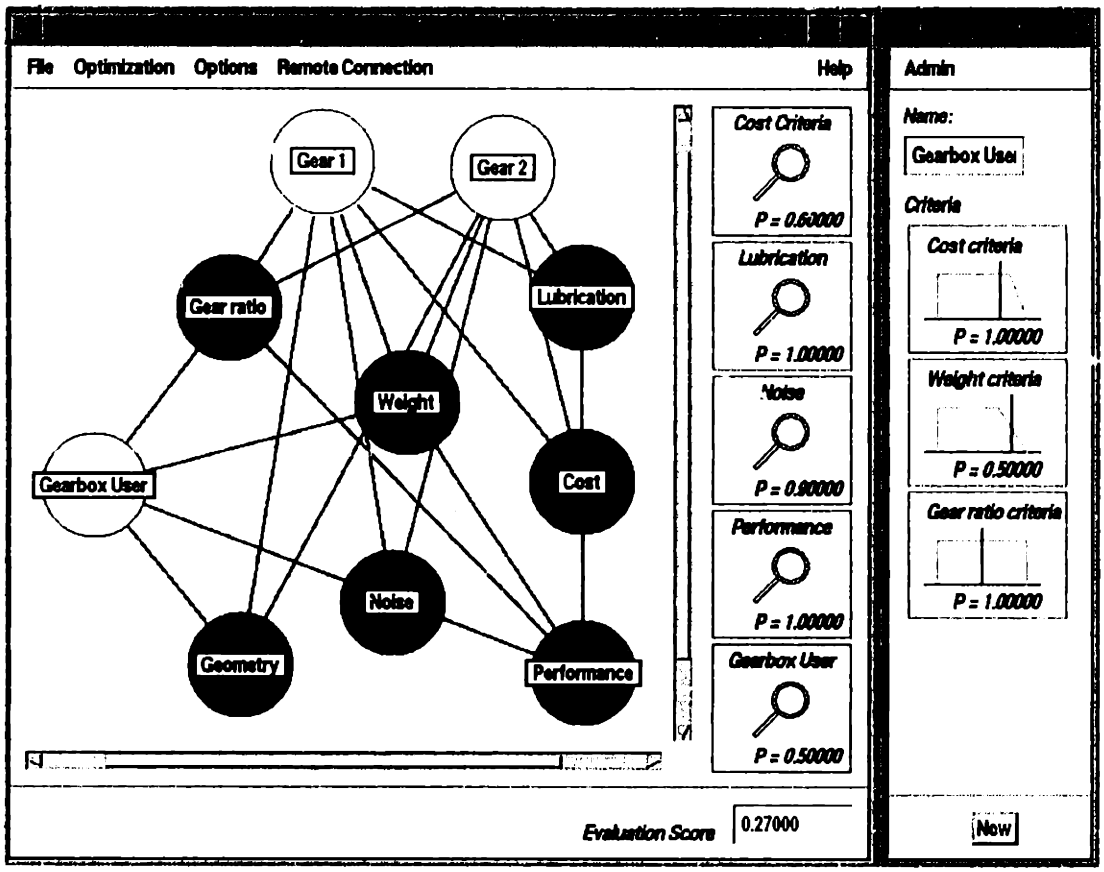


Figure 7.3 Gearbox design problem model from the Gearbox Division's viewpoint

7.1.2 Gearbox Division's Perspective

The gearbox design problem seen from the gearbox designer's viewpoint is shown in Figure 7.3. This problem model is implemented on another workstation and, as previously mentioned, is seen by the drill problem model as a single module. It is composed of several connected modules and evaluation lenses. The gearbox model also uses remote modules, which represents the gears from Gear Manufacturer. The remote module "Gearbox User" appears in the gearbox design problem because Drill Division is providing their design specifications (e.g., weight, cost, and performance specifications) to the gearbox designer. The design criteria based on these specifications are grouped in the Gearbox User lens shown in the rightmost window. The gearbox division can see how they are performing from the drill division's viewpoint and further, any changes in Drill Division's requirements will propagate to the gearbox design model.

As shown in the Gearbox User lens, the current gearbox design is not expected to satisfy the design goal with respect to the gearbox weight, which is specified by Drill Division. Understanding they need to decrease the overall gearbox weight, the gearbox designer searches for a new set of gears and contacts the

Gear Manufacturer to request light-weight gears. Since the cost criteria of the gearbox is satisfied with a reasonable margin, the designer knows that a certain range of cost variation of new gears will be tolerable[17].

7.1.3 Gear Manufacturer's Perspective

The design problem model shown in Figure 7.4 shows the model maintained by the Gear Manufacturer. The designers in Gear Manufacturer may maintain a large variety of gear data in a database and use the DOME model as a means to provide their customers with access to the information. They can also set up the model so that the customer modify some of the gear features (e.g., material, size, etc.) and see the expected gear performance (e.g., product life, estimated cost, etc.). This type of interaction will be beneficial when Gear Manufacturer provides customizable gear products. In the drill design example, since Gearbox Division seeks lighter gears, the designers in Gear Manufacturer can modify their design by changing their design parameters (i.e., weight and cost of the gear) and publish their design change to the customers. Using the windows in the lower right corner, the designers can interactively change the weight and cost of the gear.

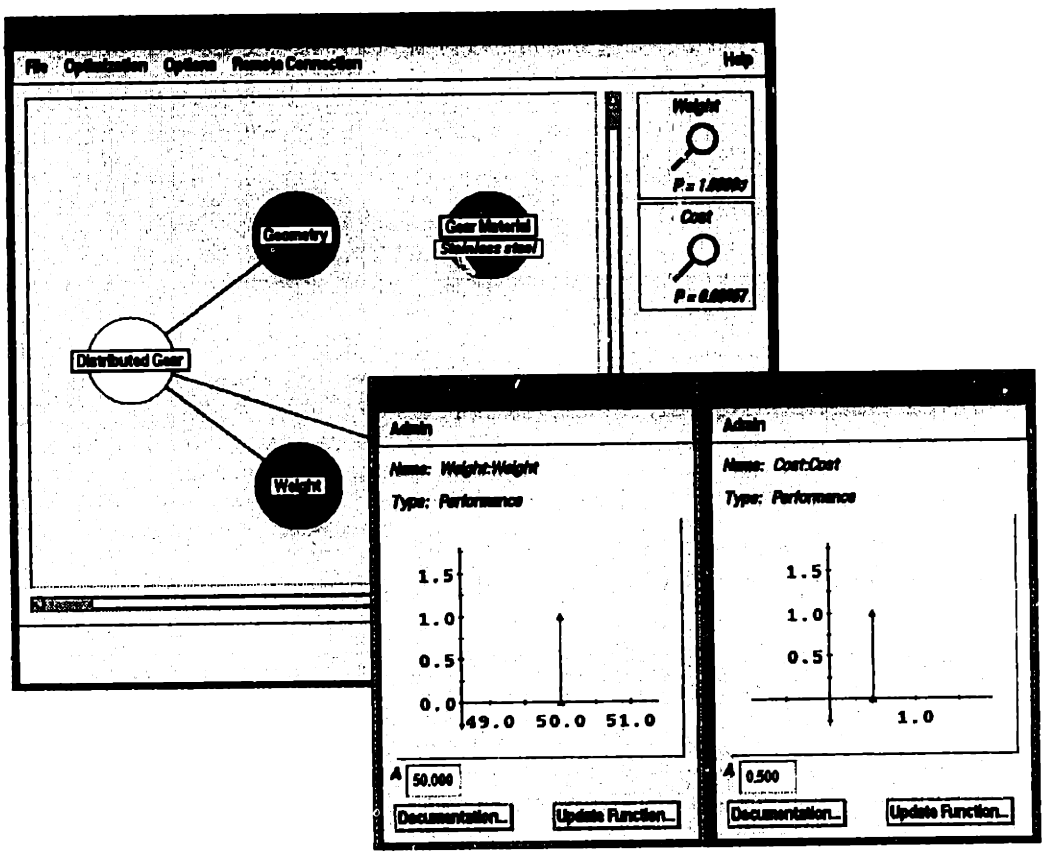


Figure 7.4 Gear Manufacturer's design problem model

[17] If a designer has a certain requirements for the desirable modules to be linked to his/her local problem model, the module search could be performed by using an agent which navigates through model repositories in different hosts. Although it is not currently implemented, the agent technology will be very useful for such functionality

7.1.4 Propagation of design changes

When the designers in the Gear Manufacturer complete their design modification, they can notify the designers in Gearbox Division. When the designers in Gearbox Division update their design, they see the expected performance of their gearbox with the new gears in place and, also, the evaluation of their current design from the viewpoint of Drill Division's designers as shown in Figure 7.5. Since the current design is satisfactory (with the acceptance probability of 0.9) with respect to the design specification from Drill Division, they now publish their design and notify Drill Division to update and evaluate their design.

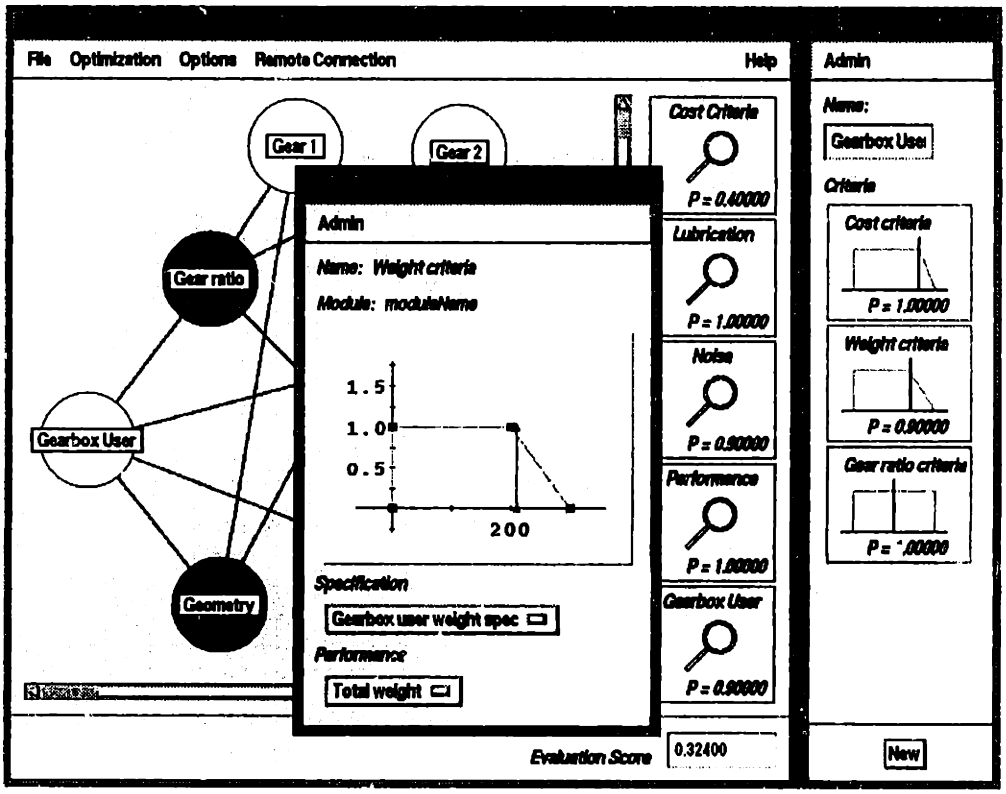


Figure 7.5 Effect of the design change made in Gear Manufacturer

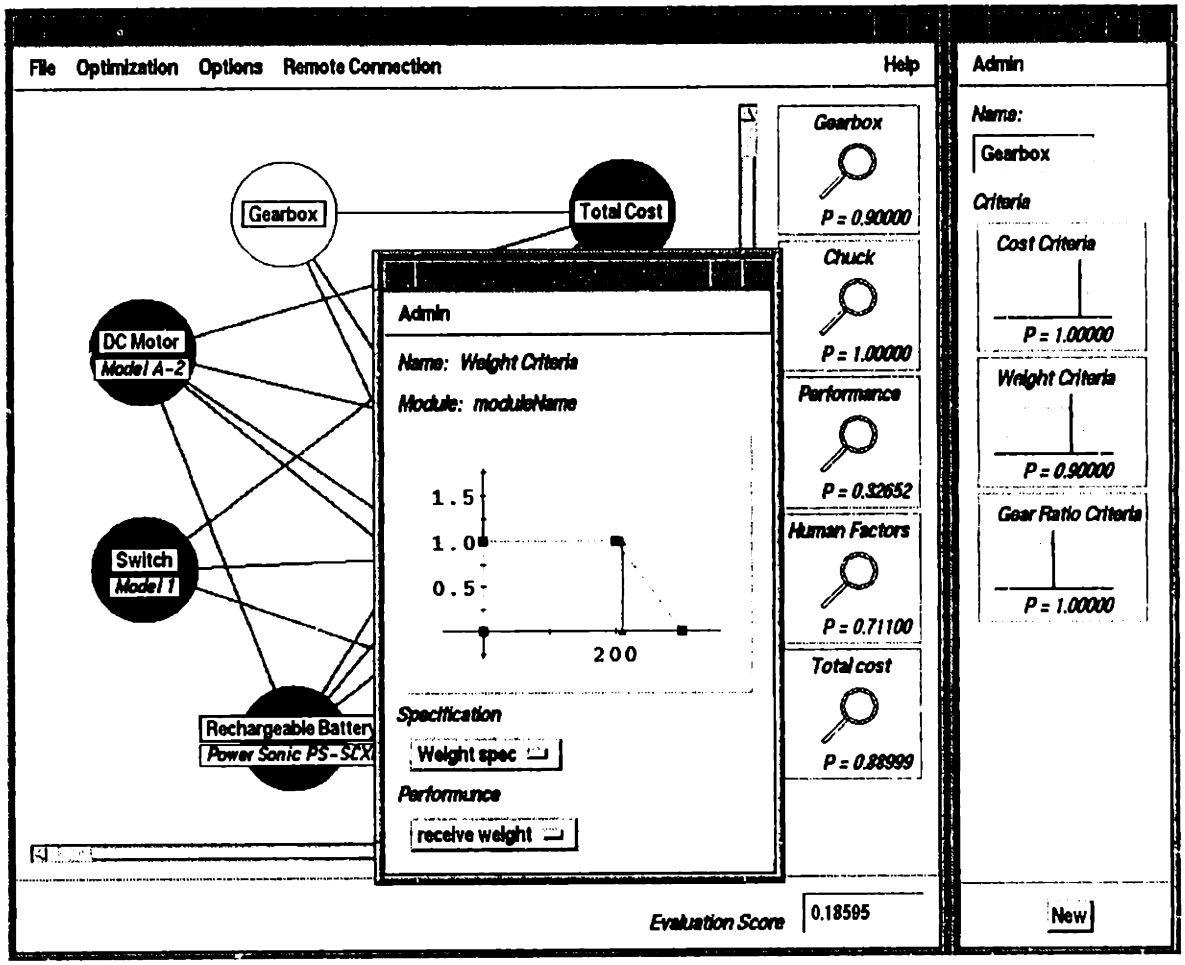


Figure 7.6 Effect of the design change propagated from Gear Manufacturer

When the designers in Gearbox Division notify the Drill Division designers , the drill designers can update their design and see the effect of design change made by Gearbox Division. As shown in Figure 7.6, the Gearbox lens shows that the current gearbox design has sufficiently satisfied the design goal that they have specified.

It is noteworthy that the gearbox model at Gearbox Division interacts with the drill model only through the service exchange. Therefore, the detail design information of the gearbox (e.g., gears used in the gearbox, gear lubricant model, etc.) is encapsulated in the gearbox module. Even if the designers in Gearbox Division facilitates a FEM application to obtain more sophisticated performance information, the gearbox module will appear the same from the Drill Division's perspective. This implies that the modular approach provides a convenient abstraction layer over a design problem model and compatibility for models with various levels of detail.

When the drill designers are notified with the improved gearbox performance, it is also possible for them to modify their specification of the gearbox in order to further improve the performance of their drill design, which can be done by changing the design goal using the design criterion window shown in Figure 7.7.

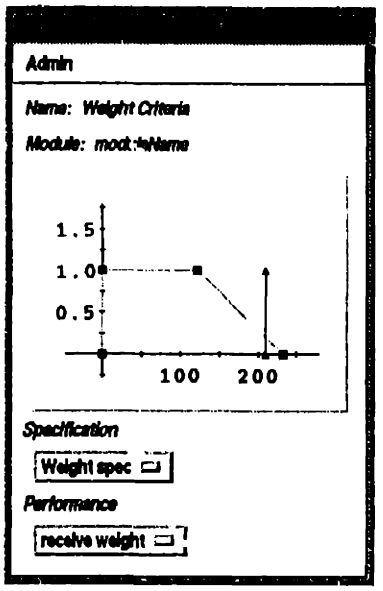


Figure 7.7 Specification change in the Drill Division

7.2 Beverage container design example

The beverage container design problem is a good example that illustrates the integrative and collaborative modeling concept of the DOME framework (Borland, et al., 1998). In this example, a model for comparing different types of beverage containers was developed. The model was broken down into four areas of expertise, distributed across several computers and integrated as illustrated in Figure 7.8.

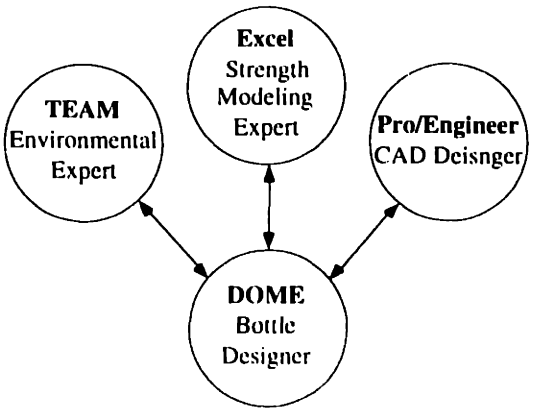


Figure 7.8 Schematic of the bottle model

The bottle designer working on the overall model is interested in producing a bottle that holds 12 to 20 ounces (350 to 600 mL) of a beverage. Cost, strength, and environmental performance are of interest to this designer, so specifications are placed upon each one: strength is to be maximized, while cost and environmental damage are to be minimized.

The designer uses DOME to model the overall system, because of its decision support and optimization capabilities, while modeling activities in environmental assessment, strength analysis and geometric modeling are delegated to experts in those fields who each use their program of choice. The strength modeling expert uses an Excel model that calculates the hoop stress based on the internal pressure, wall thickness and bottle diameter. Meanwhile, the CAD designer builds geometric models of three different bottle types in Pro/Engineer and provides calculations for material and fluid volumes. The environmental expert uses a full life-cycle model in TEAM® to predict the relative environmental impact associated with design options. These experts all have to collaborate in order to find the optimal bottle design for the given application. This combination of programs allows the bottle designer to explore the numerous possible design configurations (differentbottle types and materials, bottle dimensions, etc.) without knowing the details of all of the individual models. Tradeoffs can be made between the strength, cost and environmental performance of the product in real time. Also, an optimization tool can be used to find the most appropriate design solutions.

7.2.1 Example tradeoffs

To demonstrate the ability to make integrated assessments and tradeoffs, Figure 7.9 shows the model from the bottle designer's viewpoint. A catalog selection browser, at the bottom of the image, indicates that an aluminum can has been selected as the bottle type. The designer has defined specifications as acceptability functions. Each performance is evaluated against the corresponding specification and an acceptance probability P is calculated. Figure 7.9 shows that given the choice of an aluminum can, the cost performance is out of specification, as are two of the environmental indicators.

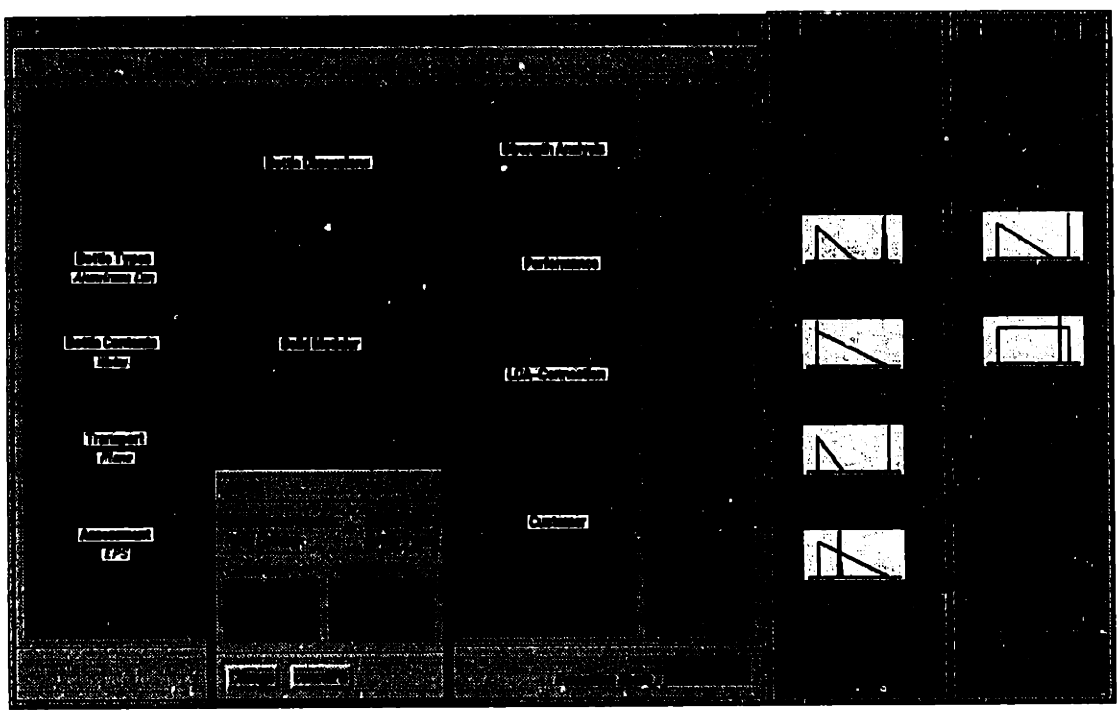


Figure 7.9 The bottle model from the design viewpoint

To remedy this, the designer decides to try a plastic bottle (Figure 7.10). According to the evaluation browsers, this improves the design considerably in both cost and environmental performance, because plastic is both cheaper and less scarce. This result might change if the life cycle model were changed (e.g. to include recycled aluminum as an option), or if the bottle designer's goals were different.

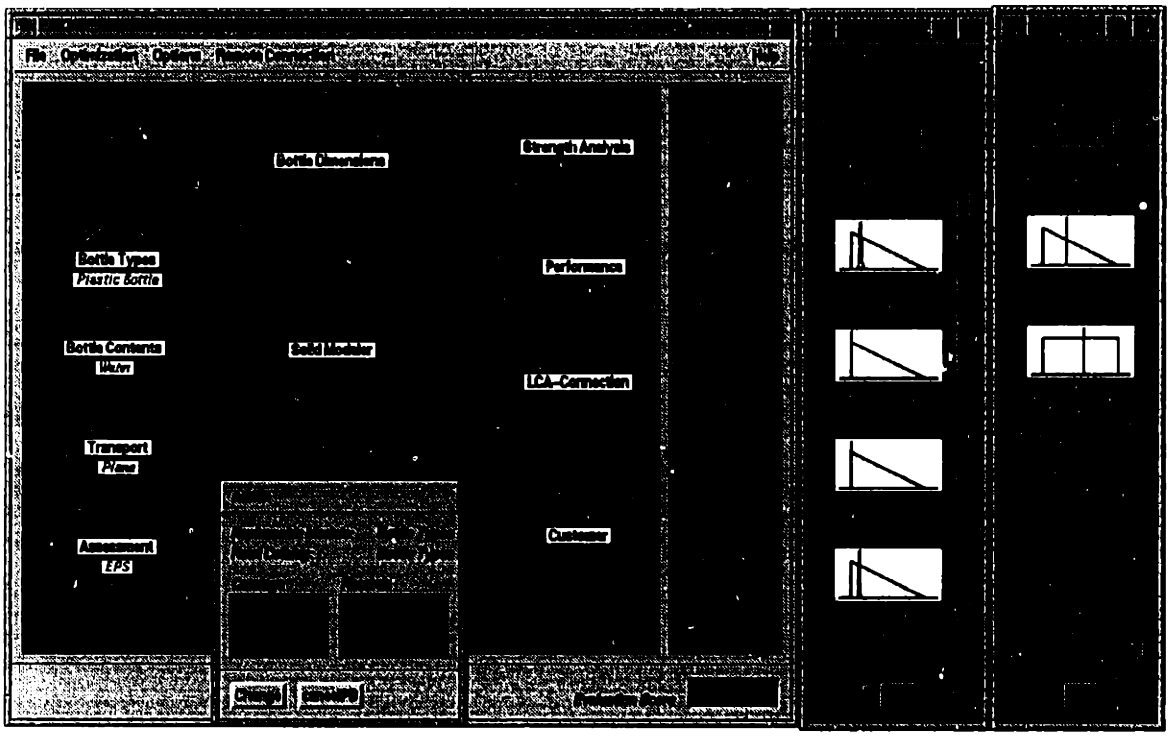


Figure 7.10 The bottle model from the design viewpoint.

The interaction between models takes very little time using this method. Building models may take hours, days, or weeks; but connecting these models together can be done very quickly and easily using this approach. Connections can be established in several minutes, and after that, communications exchanges only take a few seconds, even for remote locations. Thus designers can easily make tradeoffs between design alternatives, based on the output of detailed environmental models.

7.3 Web-based DOME examples

In the previous section the propagation of design changes throughout different levels of the design process is discussed using the drill design example. While this example illustrates well the benefit of the DOME framework for design problem modeling and evaluation in a computer network-centric distributed design environment, some of the functionalities that are essential to collaborative design activities were not fully implemented in the initial DOME prototype. In this section, design examples for the Web-based DOME are provided to describe the collaborative design activities.

While the Web-based DOME implementation is currently in progress, it provides richer context of functionality in terms of collaborative design activities. It facilitates a Web-based graphical user interface to provide designers with easy access to the DOME environment. Using Web-based DOME, designers can use any Web-browser on any type of computer operating environments. Furthermore, the three-tiered client/server architecture for creating the module network described in Chapter 4 supports various types of collaborative design problem modeling activities in the distributed design environment.

This section provides the implementation of the simple design problem example in Chapter 4 to show the different modes of using Web-based DOME. Then, the implementation of a central district heating plant design problem is presented to describe how designers from different teams, divisions, and companies may participate in an integrated modeling of a complex design problem using the Web-based DOME.

7.3.1 Simple design example for collaborative activities

Creating a new design session with an existing model

When a designer has successfully gone through the login procedure, he/she has a choice of: opening a design session to create a new design problem model; loading previously stored problem from the model repository; joining a session where other designers are already working on design problem models.

The DOME GUI window shown in Figure 7.11 shows that a designer (*kfpahng*) has selected a DOME model (designAB.mdl). As shown in the figure, the designer *kfpahng* is the owner of the model. The GUI window also lists currently active design sessions in the DOME server host (i.e., cadlab12.mit.edu). The list of models sorted by model owners is obtained from the model repository server in the same host. If another designer wants to collaborate on the modeling of a design problem, he/she selects and joins the corresponding design session provided the designer has the privilege to enter the session.

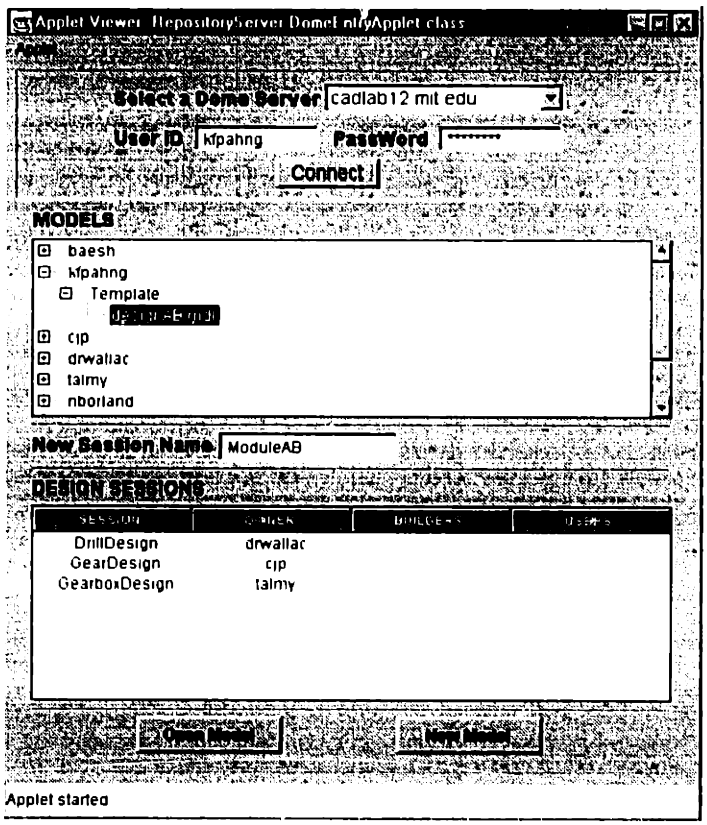
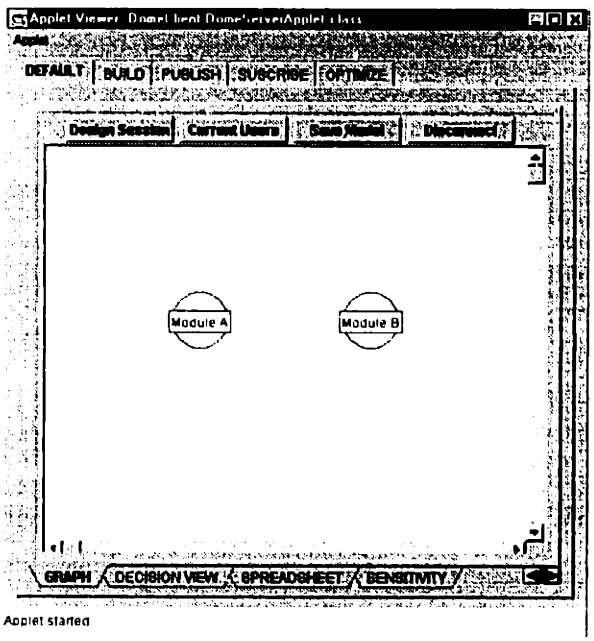
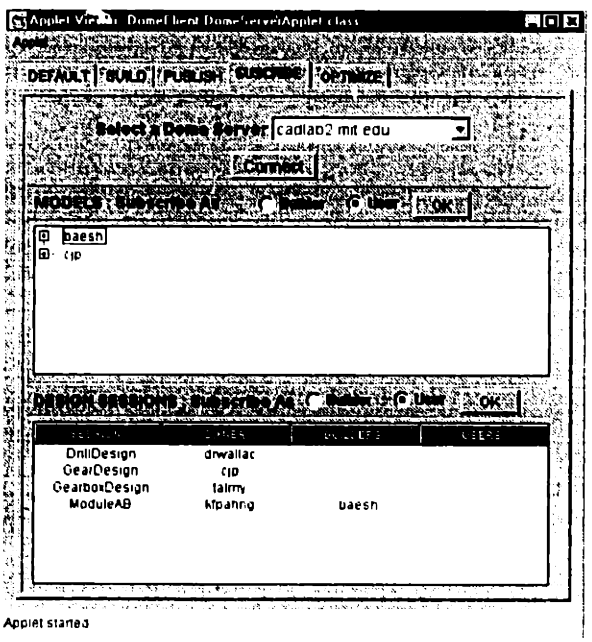


Figure 7.11 Creating a design session with an existing model



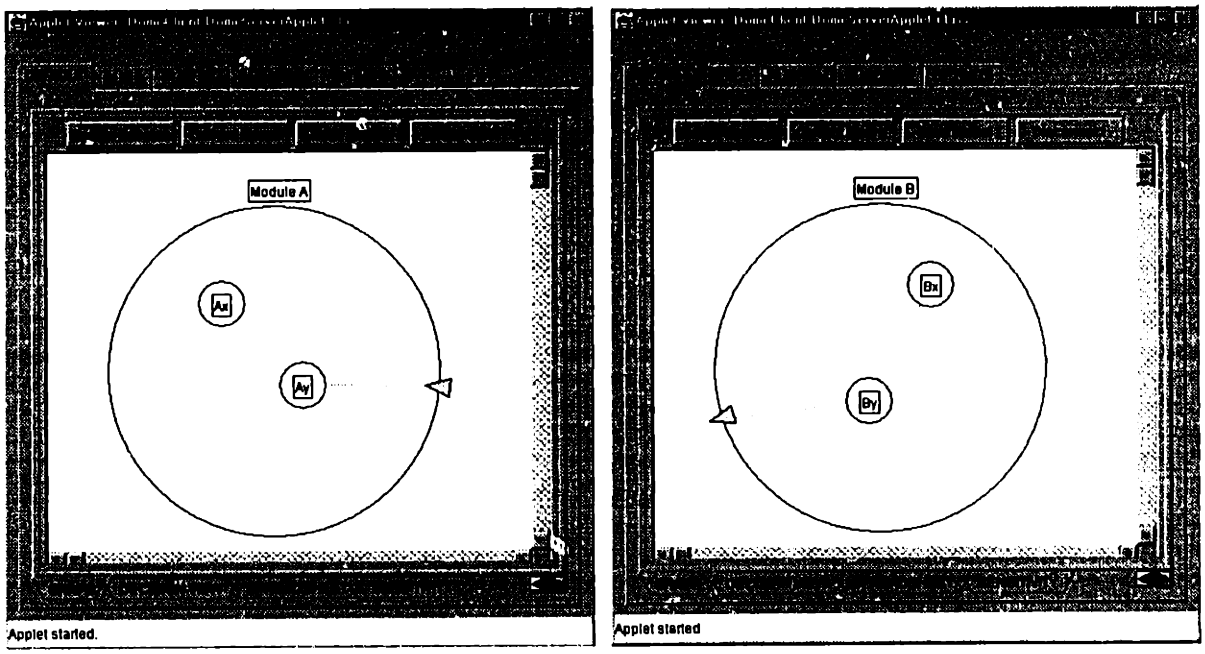
(a) Graphical view of the model



(b) Active sessions on the cadlab2.mit.edu DOME server

Figure 7.12 Simple model (moduleAB) in the active design session

Figure 7.12(a) shows the main graphical user interface of Web-based DOME with the selected model. This model has two modules (i.e., *ModuleA* and *ModuleB*). Once the designer entered the design session, he/she can modify the design problem model (e.g., creating modules, defining the relations between modules, implementing engineering models, linking to existing engineering applications, etc.) and evaluate design solution alternatives against the specifications. Figure 7.12(b) shows there are currently two designers working in the *ModuleAB* session. Since the other designer (*baesh*) is a builder, he can create a new module or modify the modules that he created. While designer *baesh* can use the services that other designers' modules provide, it is not allowed to modify other designer's modules.



(a) Embedded model of module A **(b) Embedded model of module B**

Figure 7.13 Embedded models of the modules

Figure 7.13 shows the embedded model of each module for the simple design problem model provided in Chapter 6. The model shows that *Ax* in *module A* depends on *Ax* and By in *module B*.

Setting the access privilege level

When a model is created by a designer or designers, it is possible to make the model available to other designers. The important thing to make sure before publishing the module to other builders or users is to assign the appropriate access privilege level for the services that the module provides. The owner of a module may want to conceal a know-how intensive engineering formulae or supply chain information available through the use of remote modules, which are connected to the suppliers.

Figure 7.14 shows the DOME GUI window for setting service access privilege. A designer can browse through the services of modules and set appropriate levels (i.e., owner, builder, and user) of access privilege. If the access privilege of a module service is limited to the owner, builders cannot use the service while users cannot see the service when the model is subscribed.

As shown in Figure 7.14 the user is setting the access privilege of *service Ay* of *module A* as public. This indicates that, when another designer uses the published module, he/she can use only *service A* provided from the module.

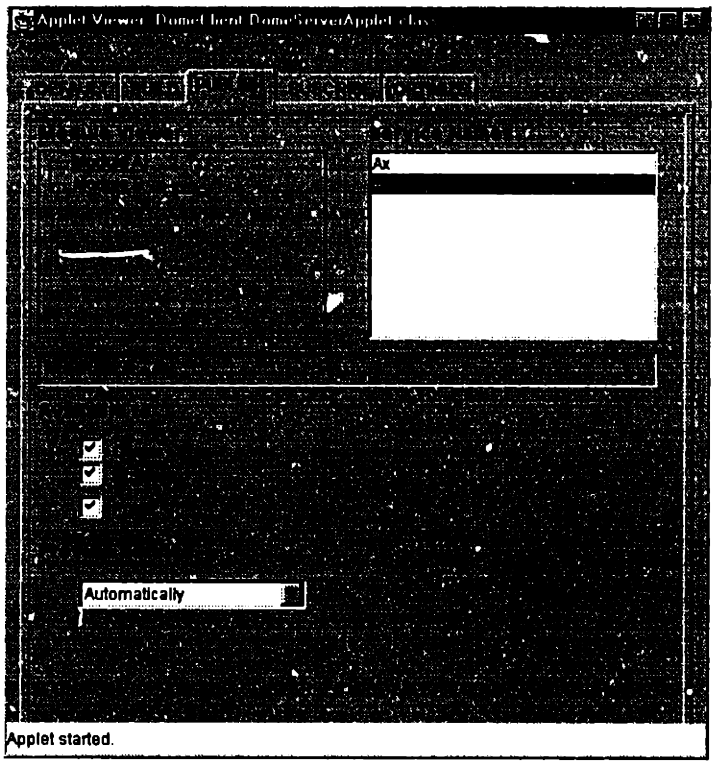
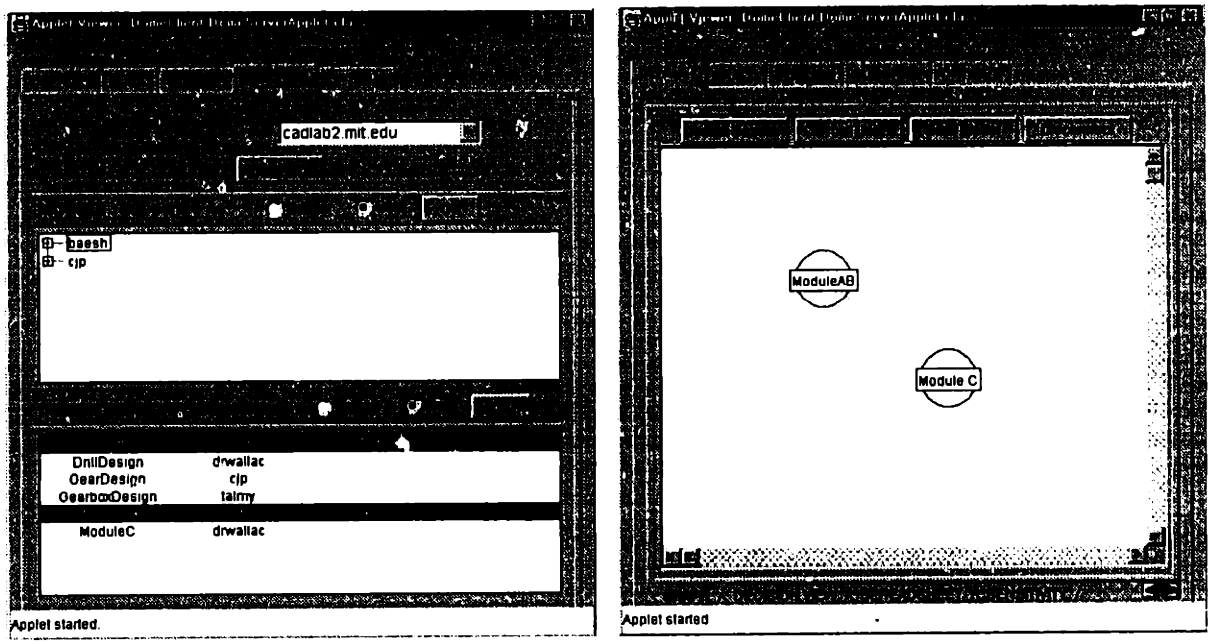


Figure 7.14 Publish window of the simple example

Subscribing a model as a user

Once a module is published on the computer network, any designer with an appropriate access privilege can use the published module as the embedded model of a remote module declared in his/her local problem scope. Figure 7.15(a) shows that another designer (*drwallac*) has selected a published module (*moduleAB*) from the active design session. Since *drwallac* has subscribed *moduleAB* as a user, the subscribed module only shows the services that are selected to be visible to public.



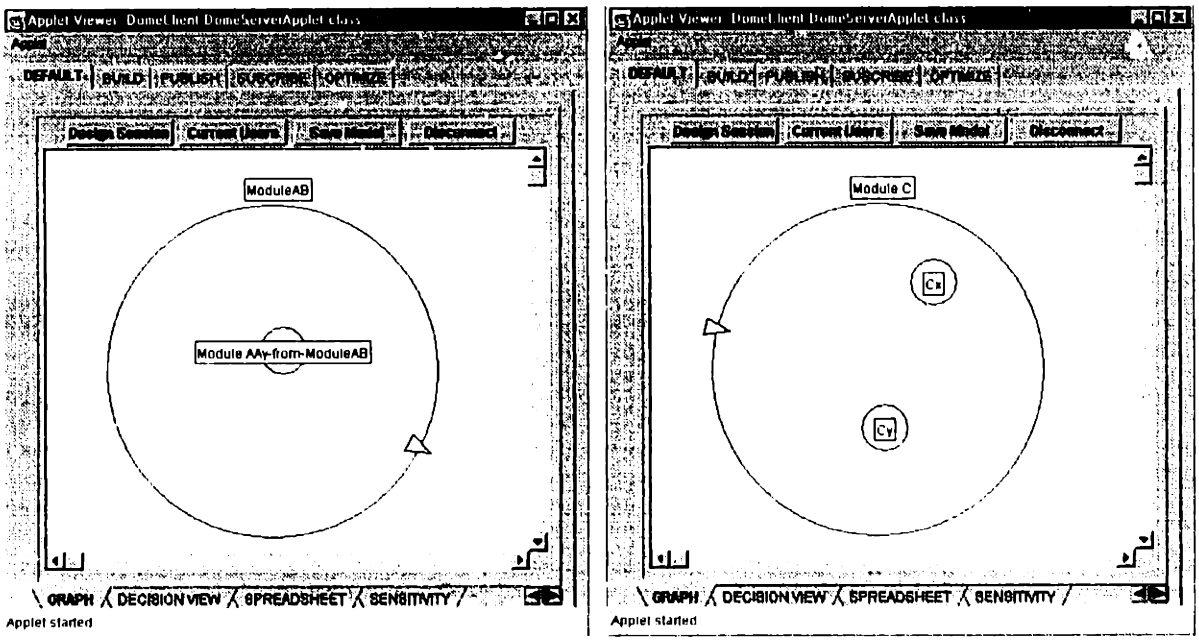
(a) Subscribing a model as a user

(b) DOME main window

Figure 7.15 Simple model (moduleC) subscribing a distributed module (moduleAB)

As shown in Figure 7.15(b), the subscribed module is seen as a single module although its actual model shown in Figure 7.12 is comprised of two modules (i.e., *moduleA* and *moduleB*). This implies that the modeling detail of a subscribed model is encapsulated and hidden from the user. No matter how the subscribed module is complex, it will be seen as a single module with a single service, as shown in Figure 7.16(a), if the owner of the subscribed module allows only one service to be available to public.

Figure 7.16(b) illustrates the design variable *Cy* depends on its local variable *Cx* and a service .provided from a remotely located model (*ModuleAB*). When the service from a remote module is provided, designer *drwallac* can see the expected value of *Cy* by requesting the service and using the mathematical formula that *Cy* contains.



(a) Service of module AB **(b) Embedded model of module B**

Figure 7.16 Embedded models of the modules in model ABC

7.3.2 Central district heating plant example for collaborative design activities

The example presented here describes how designers from different teams, divisions, and companies may participate in an integrated modeling of a complex design problem, which is the design of a district heating plant[18]. The overall topology of the problem is illustrated in Figure 7.17. The gray rectangle represents the boundary of organizations, which are connected through computer networks.

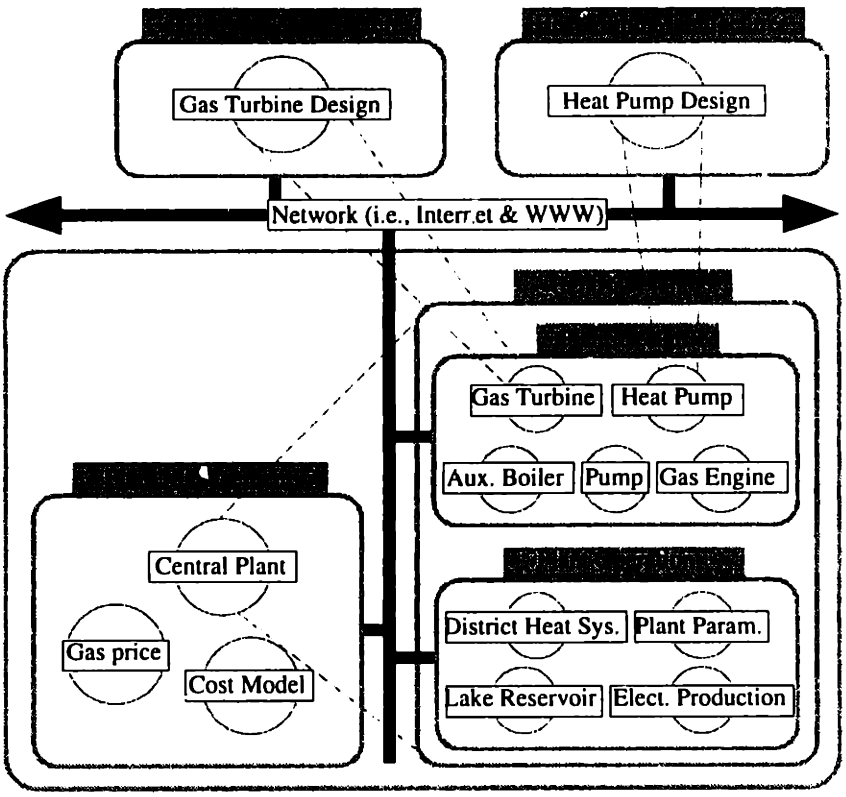


Figure 7.17 Problem topology of the central district heating plant design

In this scenario, two companies (gas turbine and heat pump manufacturers) are providing their design problem models to the plant design team, who develops the model of the heating plant. The plant design and operation teams are collaborating on the modeling of the overall central plant design. While the plant design team focuses on the system design, the plant operation team develops the model of the plant operation considering the water supply, electricity production requirement for the target district, etc. Using the central heating plant design created by the plant design and operation teams, managers in the plant management division develop the cost evaluation model considering the gas price, resource requirement, etc.

---

[18] The models used in this problem are initially constructed by Vinichio, a doctoral student at EPFL (Ecole Polytechnique Fedrale de Lausanne) under the supervision of Prof. Daniel Falrote. DOME wrappers for the simulations were created by Adam Mouneaux (EPFL), Nick Borland (MIT CADlab) and Prof. David Wallace (MIT).

In the following sections, several different user modes in using the DOME system are described through this example.

Gas turbine and heat pump manufacturer

The designers in the gas turbine and heat pump manufacturers develop the models for companies' products. When a designer works individually on the DOME model provided by the company's own DOME server and does not have to collaborate with their customers in creating the model, the designer will be modeling in *single user mode*, which is illustrated in Figure 7.18. The designer loads up the model from the local model repository and evaluates the design solution alternatives in the gas turbine design workspace running on the DOME server located at the gas turbine manufacturer site.

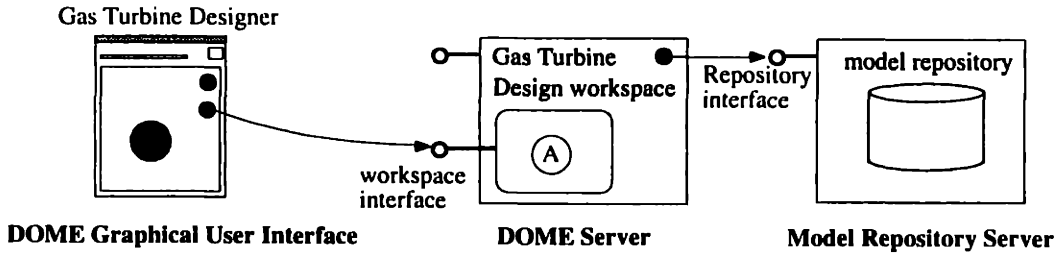
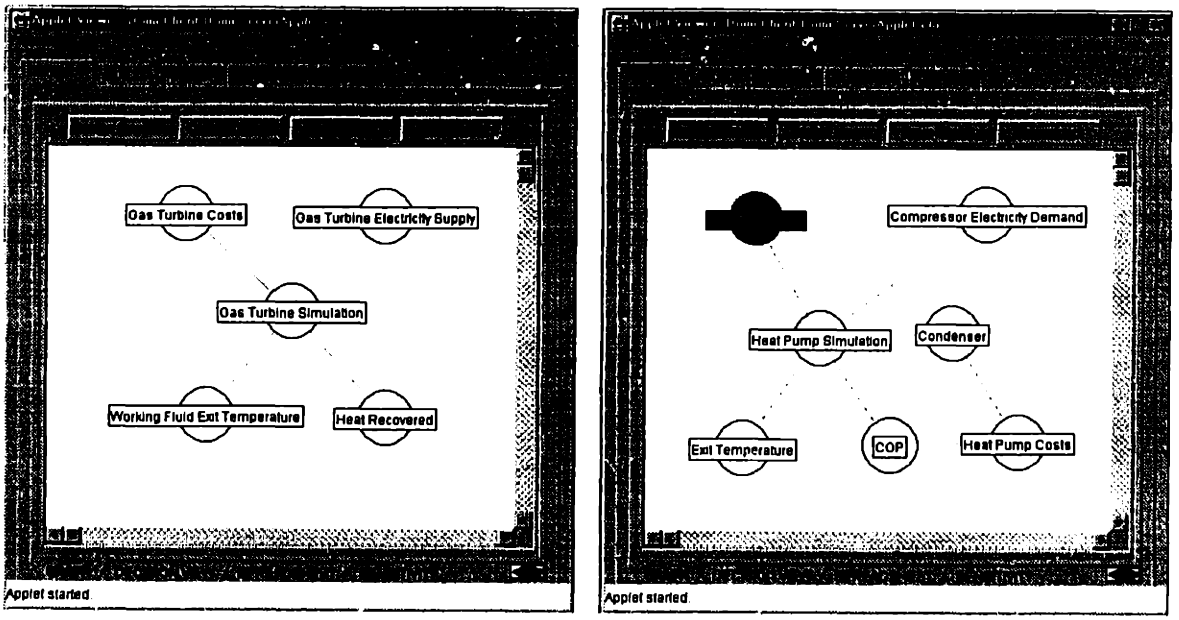


Figure 7.18 Individual workspace mode for a gas turbine designer



(a) Gas turbine model **(b) Heat pump model**

Figure 7.19 Design problem models for gas turbine and heat pump designs.

The actual DOME GUI that the designers in those manufacturers are provided in Figure 7.19. Since the GUI is written in Java applet. The designers can use any commercial Web-browser to access the model and work on the modeling and evaluation of their design problems. Since the customers will connect to their models to test the performance of the gas turbine and heat pump, the designers define what kinds of product information or simulation will be available through the services of model. When a model is published, its information is registered in the model repository server and, thus, anyone can use model if he/she has the appropriate access privilege. The owner of the module may want to conceal a know-how intensive engineering formulae or supply chain information embedded in the model.

Figure 7.20 shows the DOME GUI for the service access privilege settings. A designer can browse through the services of modules and set appropriate levels (i.e., owner, builder, and user) of access privilege. The figure shows the designer selected *Cold Exit Temperature* in *Exit Temperature* module and set the access privilege as public such that any designer can use the service.

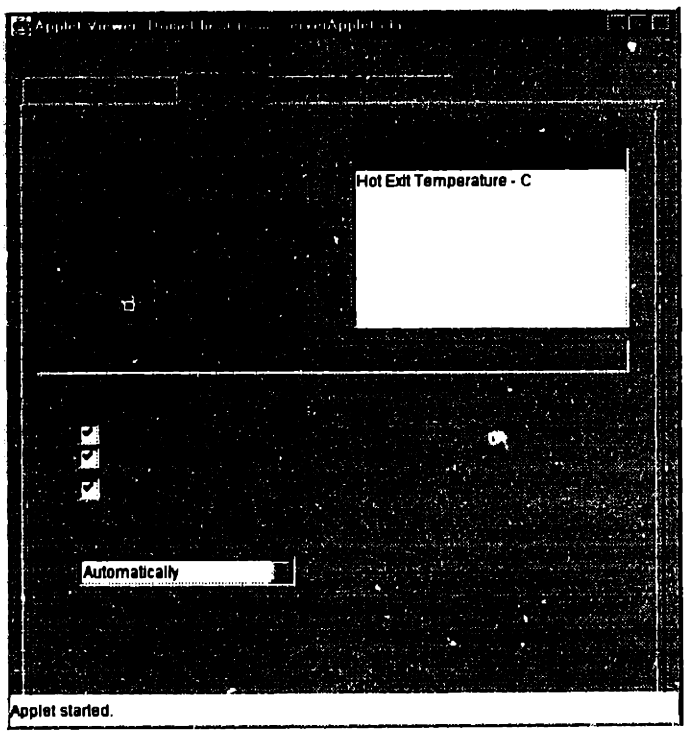


Figure 7.20 Publish window of the designer in the heat pump manufacturer

Plant design and operation teams

The plant design and the plant operation are tightly coupled and it would be reasonable approach to have a shared problem model as shown in Figure 7.21. While designers from two different groups are at remote locations, they can access into the same workspace and model the design problem together. This is referred as *Local multi-user mode*. Figure 7.21 also shows that the designer in the plant design team has connected

his/her model to the modules provided from the gas turbine and heat pump manufacturers. This interaction of utilizing models provided by other designers is referred as *subscribing a model* in *remote user mode.* While the actual models are provided from the manufacturers, the designer in the plant design team can test their plant design integrated with gas turbine and heat pump models. In the same workspace, the designers from plant operation team can participate in the design.

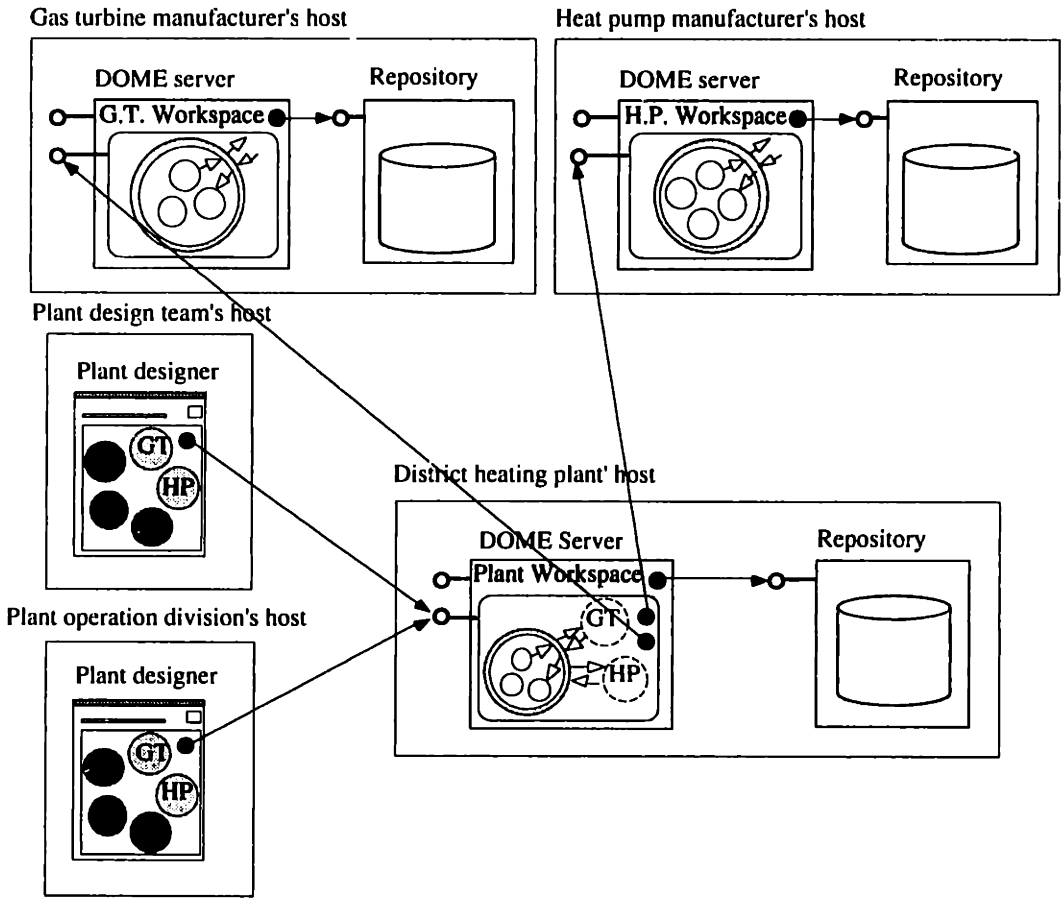


Figure 7.21 Plant design and operation teams working on a shared model

Figure 7.22 shows the design problem model viewed by the plant design and operation team designers. Modules at the upper left corner are created by the plant design team designer and the rest by the plant operation team designer. Although builders cannot modify the modules created by other builders, they can utilize the services that the modules provide. For example, the plant design team designers can use the services (i.e., design information) that *Central Plant Parameters* module provides to test the performance of their design under a certain operation condition, they cannot remove or modify the actual engineering model embedded inside the module.

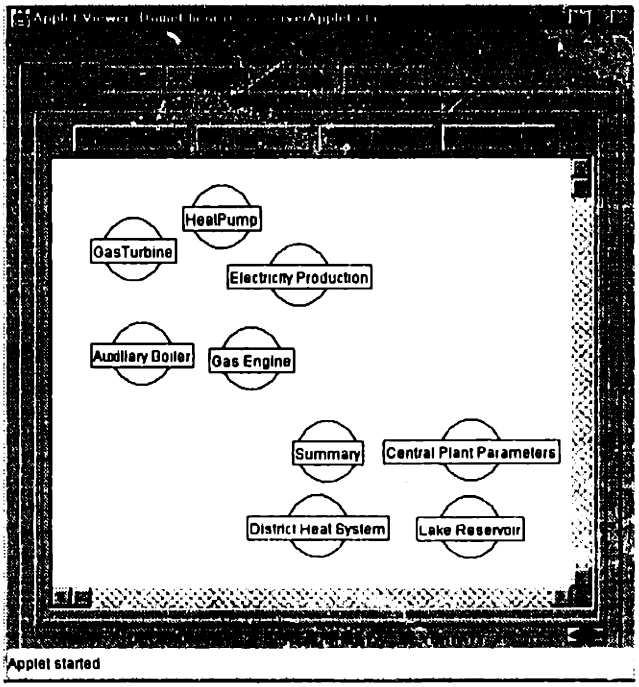


Figure 7.22 Design problem model of the central heating plant

Plant management division

While the designers in the plant design and operation teams are working on the design problem of the central heating plant, the management division may want to evaluate the plant design from the perspective of the plant management. So, they can create a model for evaluating cost estimation for the plant management and link the model to the central plant design model as illustrated in Figure 7.23.

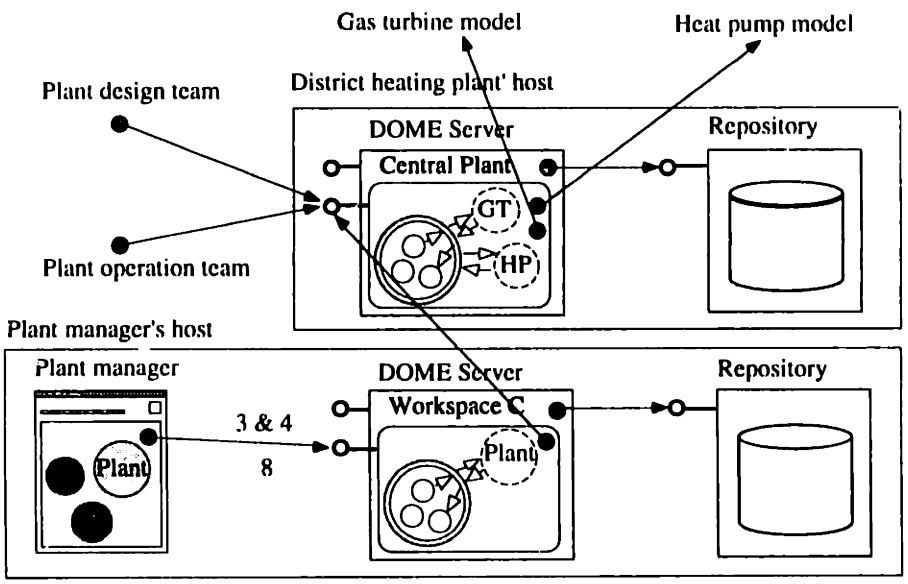


Figure 7.23 Plant management model connected to the central plant design model.

Figure 7.24 shows the plant management model, which subscribes the central plant design model provided by the plant design and operation team designers.

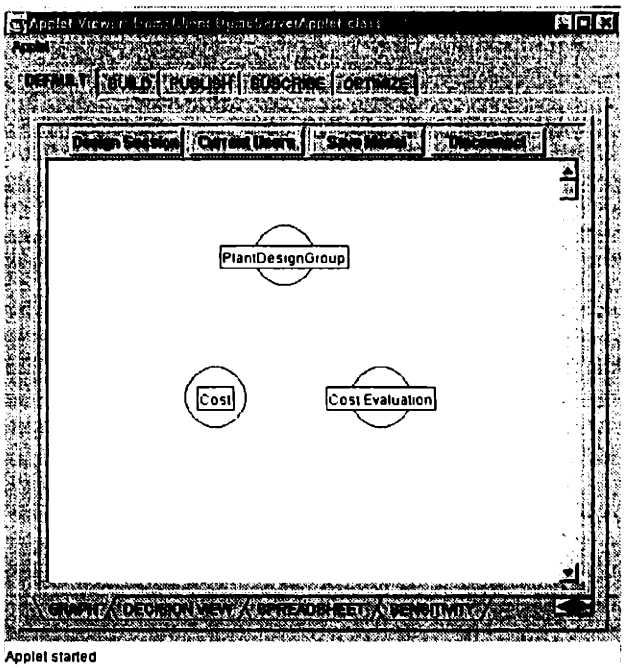


Figure 7.24 Plant management model with the plant design model subscribed

8

Conclusions

8.1 Summary

In this thesis the concept and framework for modeling and evaluation of product design problems in a network-centric distributed design environment was presented. Also, the system architecture that enables the realization of the proposed design framework is presented with preliminary implementation and examples. The intent of this effort is to reduce development time and improve product quality by helping designers to rapidly construct integrated design models that reflect interactions between subproblems.

The Distributed Object-based Modeling and Evaluation (DOME) framework is based on an Object-oriented Modeling and Evaluation (OME) framework that provides the fundamental modeling schemes for product design problems. In OME, problems can be modeled using deterministic and probabilistic variables and relations. Such models or other software applications are encapsulated in modules. Modules behave like objects and interact by reciprocal service calls. The services constitute the interface of modules and describe what each module can provide. Since each module interacts only through its interface, its embedded model remains encapsulated and invisible to the other modules. The interactions between modules can be used to model integrated design problems.

Modules can be stored in catalogs for reuse or convenient replacement when exploring design alternatives. Series or sets of modules can be created to provide transitions between computationally expensive detailed models and quick but approximate models. They can also be used to provide transitions between abstract solutions (such as a class of materials) and detailed solutions (a specific alloy). Additionally, the framework

introduces the concept of a lens. Lenses are evaluation services that act as indicators of how well design goals are met by the current status of the modules.

DOME extends the OME framework to a networked environment. Modules can be made available to users through the Internet. Conventional computer-based design tools (e.g., CAD applications, DBMSs, etc.) can also be potentially integrated in the framework by means of appropriate wrappers.

The designer models a problem by defining local fully implemented modules and remote partially defined modules. Given the designer's location constraints, compatible modules are found in the networked environment and linked to the modules defined in the local problem scope. DOME hides the details of the remote interaction mechanism from the user allowing the designer to model interactions between local and remote modules in a similar manner.

The prototype implementation of OME provides several of the desired object-oriented features. Inheritance mechanisms have not been implemented, and encapsulation needs to be strictly enforced. The preliminary implementation of DOME is built over the OME prototype and incorporates the CORBA standard for platform-independent information exchange over the network. Currently, the transparency of the modeling activity with respect to the implementation of the local or remote network-based communication mechanisms is limited.

An example problem has been implemented using the DOME prototype. The example shows how the proposed framework can be utilized in a distributed design environment where several companies, divisions or individual designers are participating in a design problem modeling activity.

8.2 Future Work

This work presented in this thesis is based upon the assumption that integrated design will require the expertise of many designers and experts and that each participant creates models/tools that, given appropriate input information, will provide information or simulation services to other participants. It is our goal that collectively, the network of participants exchanging services will form a concurrent model of the integrated design. The concept for attempting to realize this vision might most easily be described through analogy. Presently, researchers and experts publish information on web pages (Deitz, 1996). Individual web pages are then connected together to form larger topical databases (e.g., (GSSD, 1998)).

In the future, experts and designers may publish models on the world-wide-web that operate when they receive information from clients or other servers. Individual models will be connected together so that they can exchange services to form larger integrated system models. In this respect, the complete implementation of the proposed DOME framework will be able to provide a practical design tool for product designers. Some of the potential uses that constitute areas for future work are described below.

8.2.1 Interface Standards

The use and reuse of distributed modules is somewhat dependent upon the standardization of interfaces. We envision that, if experts create modules whose interfaces capture the key characteristics of their problem, this will provide valuable information to module users. Modules might also offer services of different granularity depending upon the degree to which their input requirements are satisfied.

Further, while the current implementation handles deterministic and probabilistic quantities, the exchange of standardized product model data structures such as those based upon STEP would be desirable (Owen, 1993). For instance, the employment of neutral representation for geometric information using STEP AP203 might be used to extend a design problem model based on DOME to incorporate complex geometric information.

8.2.2 Intellectual Property and Security in the Open Design Environment

Since the proposed framework enables design activities between many designers or organizations, it must address the issues of intellectual property associated with constructing integrated design problem models. Modules in the proposed framework can interact with other modules through service calls to obtain design information. This implies that there must be access control in order to prevent the loss of confidential design information. Additionally, there are many possible use modalities and levels of access to modules.

8.2.3 DOME-based applications and collaborative design

DOME does not enforce a particular design paradigm or methodology. The framework should accommodate top-down and bottom-up approaches in the context of both traditional sequential design processes and concurrent design. In a collaborative design environment other aspects such as human interaction will require the integration of additional support tools with the framework (e.g. email, video conferencing, hypertext documentation, etc.).

If a design problem modeling and evaluation framework such as DOME is integrated with a computer-based collaboration framework, it will provide designers with a powerful infrastructure for concurrent product design.

8.2.4 Computational strategy for resolving circular dependencies in a DOME model

As discussed earlier, the current DOME is limited to design problem models without circular dependencies. Several conceptual solutions to this problem are presented in Chapter 5. In order to do so, it is first required to determine the existence and location of loops within a model. The loop detection algorithm is currently being investigated. Although the loop detection in a local problem scope can be performed using several techniques (i.e., using incidence and adjacency matrices), it is difficult to find a circular dependency when

distributed modules are utilized. In this case, since the global topology of the integrated problem model cannot be obtained, a special technique is needed. This is also currently under investigation.

8.2.5 Parallel service request invocation

In order to improve the computational performance of DOME in evaluating design solution alternatives, it is being considered to adopt the simultaneous invocation of service requests. When a module needs a number of inputs for providing its service, it can simultaneously send requests to the modules that provides the needed inputs. However, to do so a careful coordination of service request may be needed. For example, it may be necessary to determine whether or not the modules providing inputs are independent of each other. In this respect, the global topology of the module interdependency in a design problem model could be used to determine the best computational strategy at run time.

Appendix A

Example Models Used in the Statistical Verification of the Complexity Measure

A.1 Combined-cycle electricity generator model

This example was made in collaboration with an energy research group at the Swiss Federal Institute of Technology (ETH, Laboratoire d'Energetique Industrielle, Department de Genie Mecanique, Ecole Polytechnique Federale de Lausanne, Directed by Prof. Dr. Daniel Favrat). This research group has developed mathematical simulations for different technologies used in combined cycle electricity production. In Switzerland, the mathematical simulations (written in FORTRAN and run on HP workstations) were provided with special DOME wrappers. At MIT, Modules containing the input parameters needed by the simulations and modules for cost analysis were constructed. These modules, exchanging design information via the Internet, form an integrated combined cycle electric power configuration tool, even though the modules are operating on different continents. This tool allows researchers to explore various technological alternatives (Pahng, et al., 1997).

A.2 Cardboard container model

In this problem, two groups of designers are working with a distributed model representing the problem. On one side are environmental experts who have access to detailed life-cycle materials inventories and make use of specialized tools for environmental impact assessment (third party TEAM® software (Ecobilan, 1996)); on the other side are traditional designers facing performance and cost issues (Senin, et al., 1997).

Appendix B

Incidence Matrix of a Module Network

B.1 Networks and its incidence matrix[19]

Let $G = (X,U)$ be a network, whose nodes and arcs are arbitrarily numbered: $X = \{x_1, x_2,..., x_n\}$ and U = $\{u_1, u_2,..., u_n\}$. Then the incidence matrix G is the $n \times m$ matrix $S = [s_{ij}]$ whose rows and columns correspond to the nodes and arcs of G respectively, and whose elements are

Equation B.1

$$s_{ij} = \begin{cases} +1 & \text{if } u_j \text{ is incident from } x_i, \\ -1 & \text{if } u_j \text{ is incident to } x_i, \\ 0 & \text{if } u_j \text{ is not incident to or from } x_i. \end{cases}$$

As an example, the network of Figure B.1 has the incidence matrix

Equation B.2

$$S = \begin{bmatrix} +1 & +1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & +1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & +1 & -1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & +1 & +1 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & +1 & +1 \end{bmatrix}$$

[19] This context is exerted from (Carre, 1979).

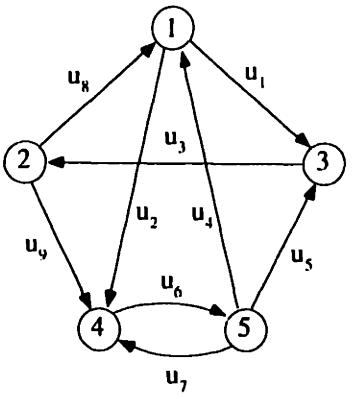


Figure B.1 Simple network

B.2 Service request chain from the incidence matrix of a module network

The service request chain can be determined from the incidence matrix of a module network. To find a path, you need to start from the initial output interface. For example, one can start from output interface O_1 and construct the service request chain corresponding to that service. First, for the selected output interface, go to the element of value -1 in the row of O_1. From that element, by alternating signs of the element, the path can be found as shown in Equation B.3. By following the path, the circular dependency can be also visually detected.

Equation B.3

$$A = \begin{array}{c} \\ M_1 \\ M_2 \\ M_3^1 \\ M_3^2 \\ M_4 \\ O_1 \\ O_2 \\ O_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} O_{1,1} & O_{1,2} & O_{2,1} & O_{2,2} & O_{3,1} & O_{3,2} & O_{4,1} \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \end{array}$$

References

Arora, J. S., *Introduction to Optimum Design*. New York: McGraw-Hill Book Company, 1989.

Bic, L. F., M. Fukuda, and M. B. Dillencourt, "Distributed Computing Using Autonomous Objects," *IEEE Computer*, August, pp. 55-61, 1995.

Bliznakov, P. I., "Design Information Framework to Support Engineering Design Process," Arizona State University, 1996.

Bliznakov, P. I. and J. J. Shah, "Integration Infrastructure to Support Concurrence and Collaboration in Engineering Design," proceedings of ASME DETC, Irvine, Ca, pp. EIM-1420, 1996.

Bliznakov, P. I., J. J. Shah, D. K. Jeon, and S. D. Urban, "Design Information System Infrastructure to Support Collaborative Design in a Large Organization," proceedings of ASME DETC, Boston, Ma, vol. 1, pp. 1-8, 1995.

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

Borland, N., H. P. Kaufmann, and D. Wallace, "Integrating Environmental Impact Assessment into Product Design: A collaborative modeling approach," submitted to 1998 ASME DETC.

Brockschmidt, K., *Inside OLE*. Redmond, Wa: Microsoft Press, 1995.

Carre, B., *Graphs and networks*. London: Clarendon press, 1979.

Case, M. P. and S. C.-Y. Lu, "Discourse Model for Collaborative Design," *Computer-Aided Design*, vol. 28, no. 5, pp. 333-345, 1996.

Chappel, D., *Understanding ActiveX and OLE*. Redmond, Wa: Microsoft Press, 1996.

Chen, H., A. Houston, J. Nunamaker, and J. Yen, "Toward Intelligent Meeting Agents," *IEEE Computer*, August, pp. 62-69, 1996.

CORNAFION, *Distributed Computing Systems*. Amsterdam: Elsevier Science Publishers b.v., 1985.

Deitz, D., "Engineering Online," *Mechanical Engieering*, no. September, 1996.

Dertouzous, M. L., "Communications, Computers and Networks," *Scientific American*, September, pp. 62-69, 1991.

Deshpande, S., "CORBA as Integrating Infrastructure in a Concurrent Engineering Environment," proceedings of Concurrent Engineering: Research and Applications, Pittsburgh, Pa, pp. 23-33, 1994.

Desrochers, A. A. and R. Y. Al-Jaar, *Applications of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis*. New York: IEEE Press, 1995.

Ecobilan, *TEAM 2.0 Users Manual*, 1996.

Eppinger, S. D., M. V. Nukala, and D. E. Whitney, "Generalized Models of Design Iteration Using Signal Flow Graphs," Massachusetts Institute of Technology Working Paper no. 3866, December 1996.

Eppinger, S. D., D. E. Whitney, R. P. Smith, and D. A. Gebala, "A Model-Based Method for Organizing Tasks in Product Development," *Research in Engineering Design*, no. 6, pp. 1-13, 1994.

Erkes, J. W., K. B. Kenny, J. W. Lewis, B. D. Sarachan, M. W. Sobolewski, and R. N. Sum, "Implementing Shared Manufacturing Services on the World-Wide Web," *Communications of the ACM*, vol. 39, no. 2, pp. 34-45, 1996.

Evbuomwan, N. F. O., S. Sivaloganathan, and A. Jebb, "A State of the Art Report on Concurrent Engineering," proceedings of Concurrent Engineering: Research and Applications, Pittsburgh, Pa, pp. 35-44, 1994.

Finger, S. and J. R. Dixon, "A Review of Research in Mechanical Engineering Design. Part I: Descriptive, Prescriptive, and Computer-Based Models of Design Process," *Research in Engineering Design*, vol. 1, pp. 51-67, 1989.

Frost, H. R. and M. R. Cutkosky, "Design for Manufacturability via Agent Interaction," proceedings of 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference, Irvine, California, 1996.

Gajski, D. D., F. Vahid, D. Narayan, and J. Gong, *Specification and Design of Embedded Systems*. Englewood Cliffs, New Jersey: PTR Prentice Hall, 1994.

Gebala, D. A. and S. D. Eppinger, "Methods for Analyzing Design Procedures," proceedings of ASME DTM, vol. 31, pp. 227-233, 1991.

Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley Inc., 1989.

Graham, I., *Object-Oriented Methods*: Addison-Wesley, 1994.

Gruninger, T. and D. Wallace, "Multimodal Optimization using Genetic Algorithms," MIT CADlab CADlab1996-01, 1996.

GSSD, "http://gssd.mit.edu/," 1998.

Hardwick, M. and D. Spooner, "An Information Infrastructure for a Virtual Manufacturing Enterprise," proceedings of Concurrent Engineering: A Global Perspective, McLean, Va, pp. 417-429, 1995.

Hauser, J. R. and D. Clausing, "The House of Quality," *Harvard Business Review*, pp. 63-73, 1988.

Hines, J., "Software engineering," *IEEE Spectrum*, vol. 35, no. 1, pp. 48-51, 1998.

IONA, *Orbix2 Programming Guide*: IONA Technologies Ltd., 1997.

Jackson, P. and D. Wallace, "An Analytical Method for Integrating Environmental and Traditional Design Considerations," proceedings of 47th CIRP General Assembly, Tianjin, China, vol. 1997.

Jackson, P. and D. Wallace, "A Modular Method for representing Product Life-cycles," proceedings of ASME DETC, Sacramento, CA, CD-ROM proceedings, 1997.

Keeney, R. L. and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*: Cambridge University Press, 1993.

Lander, S. E., "Issues in Multiagent Design Systems," *IEEE Expert Intelligent System & Their Application*, vol. 12, no. 2, 1997.

Lewis, J. W. and K. J. Singh, "Electronic Design Notebooks (EDN): Technical Issues," proceedings of Concurrent Engineering: A Global Perspective, McLean, Va, pp. 431-436, 1995.

Lynch, N. A., *Distributed Algorithms*. San Francisco: Morgan Kaufmann Publishers, Inc., 1996.

MADEFast, "http://madefast.stanford.edu/," 1998.

Marca, D. A. and C. L. McGowan, *Structured Analysis and Design Technique*. New York: McGraw-Hill, 1986.

Mason, S. J. and H. J. Zimmermann, *Electronic Circuits, Signals, and Systems*. New York: Wiley, 1960.

NIIIP, "http://www.niiip.org/," 1998.

OMG, "http://www.omg.org/," 1998.

Orfali, R., D. Harkey, and J. Edwards, *The Essential Distributed Objects Survival Guide*. New York: John Wiley & Sons, Inc., 1996.

Owen, J., *STEP - An Introduction*: Winchester, 1993.

Pahng, F., N. Senin, and D. Wallace, "Modeling and Evaluation of Product Design Problems in a Distributed Design Environment," proceedings of ASME DETC, Sacramento, CA, CD-ROM proceedings, 1997.

Pahng, F., N. Senin, and D. Wallace, "Distributed Modeling and Evaluation of Product Design Problems," *to appear in Computer-Aided Design*, 1998.

Pena-Mora, F., D. Sriram, and R. Logcher, "SHARED-DRIMS: SHARED Design Recommendation-Intent Management System," 1996.

Pena-Mora, F., R. Sriram, and R. Logcher, "Conflict Mitigation System for Collaborative Engineering," *AI EDAM - Special Issue of Concurrent Engineering*, vol. 9, no. 2, pp. 101-123, 1995.

Petrie, C., M. Cutkosky, and H. Park, "Design Space Navigation as a Collaborative Aid," proceedings of Third International Conference on Artificial Intelligence in Design, Lausanne, 1994.

Pugh, S., *Total Design: integrated methods for successful product engineering*. Wokingham, England: Addison Wesley, 1990.

RaDEO, "http://elib.cme.nist.gov/radeo/," 1998.

Saaty, T. L., *The Analytic Hierarchy Process*. New York, NY: McGraw-Hill, 1980.

Salzberg, S. and M. Watkins, "Managing Information for Concurrent Engineering: Challenges and Barriers," *Research in Engineering Design*, no. 2, pp. 35-52, 1990.

Senin, N., N. Borland, and D. R. Wallace, "Distributed Modeling of Product Design Problems in a Collaborative Design Environment," proceedings of CIRP International Design Seminar Proceedings: Multimedia Technologies for Collaborative Design and Manufacturing, Los Angeles, CA, 1997.

Senin, N., D. R. Wallace, and J. Mark, "Mixed Continuous Variable and Catalog Search Using Genetic Algorithms," proceedings of ASME DETC and CIE conferences, Irvine, CA, 1996.

Siegel, J., *CORBA: Fundamentals and Programming*: OMG, 1996.

Singh, A. K., "CONSENS - An IT Solution for Concurrent Engineering," proceedings of Concurrent Engineering: A Global Perspective, McLean, Va, pp. 635-644, 1995.

Sobolewski, M. W. and J. W. Erkes, "CAMnet: Architecture and Applications," proceedings of Concurrent Engineering: A Global Perspective, McLean, Va, pp. 627-634, 1995.

Sriram, D. and R. Logcher, "The MIT DICE Project," *IEEE Computer*, pp. 64-65, 1993.

Steward, D. V., "The Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Transactions on Engineering Management*, no. August, pp. 71-74, 1989.

Suh, N. P., *The Principles of Design*. New York: Oxford University Press. Inc., 1990.

Toye, G., M. R. Cutkosky, L. J. Leifer, J. M. Tenenbaum, and J. Glicksman, "SHARE: A Methodology and Environment for Collaborative Product Development," proceedings of Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Morgantown, West Virginia, pp. 33-47, 1993.

Wall, M., "Mattew's GAlib: A C++ Library of Genetic Algorithm Components," in *http://lancet.mit.edu /ga/*, 1996.

Wallace, D. R., "A Probabilistic Specification-based Design Model: applications to search and environmental computer-aided design," Massachusetts Institute of Technology, 1994.

Wallace, D. R., M. Jakiela, and W. Flowers, "Design Search under Probabilistic Specification using Genetic Algorithm," *Computer-Aided Design*, vol. 28, no. 5, pp. 405-421, 1996.

Wegner, P., "Models and Paradigms of Interaction," proceedings of ECOOP '93 Workshop, Kaiserslautern, Germany, pp. 1-32, 1993.

Westerberg, A. W., R. Coyne, D. Cuningham, A. Dutoit, E. Gardner, S. Konda, S. Levy, I. Monarch, R. Patrick, Y. Reich, E. Subrahmanian, M. Terk, and M. Thomas, "Distributed and Collaborative Computer-Aided Environments in Process Engineering Design," proceedings of ISPE, 1995.

Wiest, J. D. and F. K. Levy, *A Management Guide to PERT/CPM*. Englewood Cliffs, New Jersey: Prentice-Hall, 1969.