IOWA STATE UNIVERSITY
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2008

Establishing an advanced engineering framework for engineering decision making

Douglas Stinson McCorkle
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Mechanical Engineering Commons

Recommended Citation
McCorkle, Douglas Stinson, "Establishing an advanced engineering framework for engineering decision making" (2008). *Retrospective Theses and Dissertations*. 15780.
https://lib.dr.iastate.edu/rtd/15780

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

www.manaraa.com

Establishing an advanced engineering framework for engineering decision making

by

Douglas Stinson McCorkle

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Program of Study Committee:
Kenneth Bryden, Major Professor
Daniel Ashlock
Tom Shih
Eliot Winer
Carolyn Heising

Iowa State University

Ames, Iowa

2008

Copyright © Douglas Stinson McCorkle, 2008. All rights reserved.

www.manaraa.com

UMI Number: 3383366

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 3383366
Copyright 2009 by ProQuest LLC
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

www.manaraa.com

To my wife for her support, encouragement, patience, and love throughout this endeavor.

To my family for their sacrifice.

www.manaraa.com

Nomenclature .... **v**

Acknowledgements .... **vi**

Chapter 1: Introduction **1**

Chapter 2: Background **5**
2.1 Informatics 6
2.2 Problem Solving Environments 9
2.3 Object-oriented programming 12
2.3.1 Early software implementations of object-oriented programming 13

2.3.2 Object-oriented design/analysis 15

2.4 Current Uses of Object-Oriented Methodologies 21
2.5 Frameworks 27
2.6 Meta Data and Semantics 31
2.6.1 Engineering Information Storage 32

2.7 Virtual Worlds 38
2.8 Object Definitions 39
2.9 Characteristics that objects must inherently have 42

Chapter 3: Towards an advanced engineering framework **45**
3.1 Advanced Engineering Software Frameworks 48
3.2 Objects 52
3.3 Object Interactions 55
3.3.1 Emergent Behavior 60

3.3.3 Object-Oriented Principles 61

3.5 Summary 66
Chapter 4: Implementation of the proposed advanced engineering framework 68

4.1 Transparent Interfaces 69
4.1.1 Implementation of Transparent Interfaces 70

4.1.2 Summary 74

4.2 Object-Oriented Principles 75
4.2.1 Modularity, Hierarchy, and Abstraction 75

4.2.2 Ontologies 77

المنارة للاستشارات

www.manaraa.com

4.2.3 XML Schema 79
4.3 Emergent Behavior 86
4.3.1 VE-Conductor 91
4.3.2 VE-CE 91
4.3.3 Computational Unit 92
4.3.4 VE-Xplorer 96
4.4 Structure for VE-Suite Application Directory 109
4.5 Summary 111

Chapter 5: Large and Ultra-Large System Integration 112
5.1 APECS 114
5.2 Aspen Plus 117
5.3 CASI 117
5.3.1 Object-oriented architecture 120

5.4 VE-AspenUnit 120

Chapter 6: Engineering Mass Products 131
6.1 Cotton picker models 136
6.3 VE-Suite software plugins 140
6.3.1 Graphical plugins 140

6.3.2 UI plugins 143

6.3.3 Units 143

6.4 Engineer's Experience 145

Chapter 7: Results & Conclusions 147

Appendix A: VE-Suite Description 152
Model integration and communication: VE-Open 155

Graphical user interface: VE-Conductor 158

Computational Engine: VE-CE 160

Graphical Engine: VE-Xplorer 162

Appendix B: Example DOMDocument 164

Bibliography 170

www.manaraa.com

Nomenclature

APECS – Advanced Process Engineering Co-Simulator

API – Application Programming Interface

CAD – Computer Aided Drafting

CFD – Computational Fluid Dynamics

CORBA – Common Object Request Broker Architecture

FEA – Finite Element Analysis

GUID – Global Unique Identifier

IDL – Interface Definition Language

IGES – Initial Graphics Exchange Specification

NURBS – Non-Uniform Rational B-Spline

OCC – OpenCASCADE

OWL – Web Ontology Language

PSE – Problem Solving Environment

STEP – Standard for the Exchange of Product Model Data

UI – User Interface

UML – Unified Markup Language

URI – Uniform Resource Identifier

VE-Suite – Virtual Engineering Suite

XML – Extensible Markup Language

XSL – Extensible Stylesheet Language

XSLT – Extensible Stylesheet Language Transform

www.manaraa.com

Acknowledgements

I would like to thank Mark Bryden for helping mold and shape my academic and professional career. As my advisor, I would specifically like to thank him for allowing me to forge my own research path to reach this stage of my career. This opportunity can be difficult to provide to a graduate student, but is invaluable to developing as a researcher. In addition, his editing of this work enabled clear presentation of my research that would not have been possible otherwise. As my friend, I would like to thank him for his support and encouragement over the past seven years.

I would also like to thank my committee members for their participation in my research. I would like to thank the many collaborators and researchers who provided feedback and comments throughout this research, specifically the Simulation, Modeling, and Decision Sciences Program, and Professor Dan Ashlock. I would also like to thank my parents, Allen and Gayle, for their support and sacrifice in my educational journey. I would like to thank Nick and Kathleen for their encouragement and care of my family during the long hours of studying, research, and travel.

This work was funded in part by the U. S. Department of Energy National Energy Technology Laboratory and John Deere.

المنارة للاستشارات

www.manaraa.com

Chapter 1: Introduction

Engineering is a process of decision making for complex and uncertain systems. In the past, the unknowns for these systems were handled through rules of thumb, observation, safety factors, and intuition. As computational power increases, rules of thumb, observations, and intuition are being supplemented by numerical models, simulations, engineering analysis, and other computational tools. These numerical models are based on the equations that describe physical phenomena (e.g., Navier-Stokes equations are used to describe fluid flow, Fourier's Law for heat transfer, and Hooke's Law for stress/strain relationships in materials). Although these models can provide significant insight into the engineering design process, they are not currently used as design tools but rather as analysis tools. In fact, the application of computational science in engineering has not provided a clear way to deal with ambiguity and uncertainty in engineering. There are several reasons for this:

- Numerical models currently require manual integration of model-to-model information
- Human-accessible quantification of error and uncertainty surrounding models is not readily available
- Semantically rich information frameworks for managing systems of models do not exist to enable full-model pedigree information to be exchanged

المنارة للاستشارات

www.manaraa.com

- Individual models and simulations cannot be easily integrated to create complete analysis systems that capture the richness and fullness of a complex system

For these reasons, the engineering models used today result in significant disconnects in the engineering process as multiple models are individually created, revised, and manually updated.

To overcome these issues, new practices and methods must be created that enable engineers and analysts to improve the speed of the engineering process and to connect engineering analysis with the creative aspects of engineering design. This will require changing how information is fundamentally treated in the engineering process. Rather than managing information at the human-to-human level, information must be managed at a much lower level to remove the human middleware from the process. The human is the slow link in the process that is currently used. The traditional engineering process has two characteristics that are currently the limiting factors in improving the process efficiency:

- Manual integration of information
- Physical prototyping

One example of this would be an engineer working through an analyst to better understand the results of a simulation. Another example is the process of moving data from one engineering analysis package to another and integrating the results of multiple analysis packages in the engineer's mind. All of these practices require humans to become middleware in a process that ought to be directly accessible by the individual seeking information. As humans, we present and filter information based on a particular perspective developed through experience and individual bias. The human filtering process can remove valuable information that may be important to the downstream user. A

المنارة للاستشارات
www.manaraa.com

computer, when tasked with integrating information, will filter and bias the information only as the user directs.

Another limiting factor within the traditional engineering process is the use of physical prototyping and of numerical models, which are exacting. Physical prototypes are useful for integrating all the components and physical phenomena together. However, physical prototypes do not enable direct information integration from one design option to another. In addition, data measurement within a physical prototype can be time-consuming and difficult, and the quantity of interest can often not be measured directly. Because of this, physical prototyping is best primarily for confirmation and exploration and not directly as a design tool.

In contrast, numerical models are used as a very precise tool providing very detailed information about a specific component or phenomenon. However, computational models are time consuming, the connection between the model results and the engineering question being asked is often not clear, and they cannot be easily connected together to create complete systems.

A new approach is needed that can combine the breadth of physical prototyping and the richness of numerical analysis in a timely and easily understood manner. This new approach needs to empower the engineer to quickly investigate a wide range of options. It must be applicable from initial design, through final design, and then provide an engineering platform through the life of the engineered product; and it needs to explicitly address error and uncertainty. This approach must provide for model portability and enable complete systems of models to be built easily and naturally.

المنارة للاستشارات
www.manaraa.com

This thesis proposes a framework that addresses these issues. Within this engineering framework, models of specific phenomena and components are coupled together to build engineering objects. These engineering objects are then coupled together to create systems and systems of systems. The key aspects of this framework are:

- An object-oriented approach to information management
- Incorporation of emergent behavior in the modeled system
- Support for bottom-up information semantics

These issues will be implemented by applying the concepts of object-oriented programming to engineering simulation and design. Several research areas surrounding informatics will be examined (e.g., product life-cycle management, computational systems biology, and the Semantic Web). The emergent behavior that is being enabled by tools created for the Semantic Web will be utilized to enable emergent behavior in advanced engineering software frameworks. Fields in the humanities will be reviewed for insight into how humans internalize interactions with objects to provide methods for characterizing information in the engineering process (e.g., analysis data, CAD data, costing data). Each of these areas will provide a capability that will enable the creation of an advanced engineering framework that will enable engineering objects to be created that mimic their physical counterparts.

للاستشارات المنارة
www.manaraa.com

Chapter 2: Background

Computers have been used in the engineering design process since the early 1960s. An example of this is the use of computers to model manufacturing processes to optimize route planning [Dahl et al. 1966]. Originally, computers were used as a faster slide rule, in that they were expected to quickly perform analyses that could have been done by hand with sufficient time. As solvers improved, the analyses that computers were expected to perform became more and more complicated, until computers could analyze in minutes or days phenomena that were too time-consuming to ever be computed by hand. Today, this type of computing continues as scientific computing or engineering analysis, and involves solving equation sets, usually partial differential equations that describe a particular physical phenomenon. As solvers improved, computers were also being developed as a means to perform other tasks, including:

- text-based processing (1980s)
- hypertext information (1990s)
- user-enhancement applications, e.g., wikis, blogs, and mashups (2000s)

Engineering has been slow to adopt these newer information technologies. Because engineering analysis is very closely related to scientific computing, that connection is easy to make. However, the connection between engineering design and a wiki or a mashup is not as clear. Nonetheless, engineering is about making a decision, understanding risk and

المنارة للاستشارات
www.manaraa.com

uncertainty, and managing complex information, which are the very concerns that information technology and informatics work to address. The definition of information and how we manage it is changing, and the process of engineering must change with it. Today, engineers act primarily as middleware. Engineers move data from CAD packages or spreadsheets to analysis packages such as CFD solvers or FEA solvers. This is so deeply ingrained in engineering that many engineers would argue that these middleware functions are in fact the most important functions that an engineer performs. Engineering software is needed that is based on the fundamentals of informatics and that moves the engineer from the middleware process in engineering product realization to the higher-level tasks requiring creativity, judgment, and values.

2.1 Informatics

> Informatics is the science of working with and processing information. Informatics … encompasses, and builds on, a number of existing academic disciplines: primarily artificial intelligence, cognitive science and computer science. Each takes part of informatics as its natural domain: in broad terms, cognitive science concerns the study of natural information processing systems; computer science concerns the analysis of computation, and the design of computing systems; artificial intelligence plays a connecting role, producing systems designed to emulate those found in nature. Informatics also informs, and is informed by, other disciplines, such as mathematics, electronics, biology, linguistics, psychology, and sociology. Thus informatics provides a link between disciplines with their own methodologies and perspectives, bringing together a common scientific paradigm, common engineering methods and a pervasive

المنارة للاستشارات
www.manaraa.com

> stimulus from both technological development and practical application. [Fourman, M. 2002, p. 2]

A general definition of informatics is "the study of the structure, behavior, and interactions of natural and artificial systems that store, process, and communicate information" [Fourman, M. 2002, p. 2]. In the case of the research discussed in this document, the informatics technologies of interest are knowledge storage and discovery, computer-driven knowledge creation, and self-describing data encapsulation.

The issues of knowledge storage and discovery and self-describing data encapsulation are currently being addressed with ontologies. The creation of ontologies and other knowledge management tools [Rosse et al. 2003, Gehlert et al. 2007, Garcia et al. 2004] are a current area of research within the informatics field. An ontology is "an explicit and formal specification of a conceptualization" [Gruber 1993, p. 200]. Pragmatically, an ontology defines a domain of discourse with a finite list of terms and a relationship between those terms. The research surrounding ontologies focuses on the creation of ontological languages such as the Web Ontology Language [Herman 2007]. This research will be discussed later in this dissertation to provide context for the use of ontologies. Another research area is the implementation of these languages in particular domains such as engineering, biology, and manufacturing [Kerrigan 2003, Kitamura et al. 2004, Kriete et al. 2005]. In engineering, researchers are using ontologies to aid in distributed design environments. In biology, researchers are using ontologies to classify systems within the body to share research results with collaborators. In manufacturing, ontologies are being used to enable companies to better understand part usage and distribution. These examples will be examined in more detail in later sections. Other

المنارة للاستشارات
www.manaraa.com

interest areas in informatics research include the integration of artificial intelligence [Chang et al. 2004], real-time tracking with RFID [Ergen et al. 2007], and learning algorithms [Colombo et al. 2007] in a way that enables organizations to gain insight and improve the efficiency of business processes.

Engineering informatics generally encompasses the management of and interaction with data, information, and knowledge specific to managing information for manufacturing processes and managing information attached to CAD data [Bliznakov 1996, Bliznakov et al. 1996, Qureshi 1997, Wang 1993]. One aspect of this work has been the development of information management frameworks, which are software tools that process information via a given schema. An example of one of these software frameworks is to enable the manufacturing process to run more efficiently and to determine bottlenecks in the process [Wang 1993]. Other work in creating product information management frameworks has focused on attaching information to CAD entities [Bliznakov 1996, Wang 1993, Qureshi 1997]. Many of the efforts to manage information in engineering have surrounded CAD data and have been specifically focused on geometric data. The goal of this work is to provide some level of automation to the retrieval of information that is intuitive to the engineer. Whole research areas have focused primarily on manufacturing and CAD data; little research effort has been focused on time-dependent data and the hierarchy of information (e.g., computational fluid dynamics, economics, spreadsheet models, and experimental data) for one entity in a system of components. Progress has been made on several components of this problem, which are described in the following sections.

المنارة للاستشارات
www.manaraa.com

2.2 Problem Solving Environments

Problem solving environments were first conceived in the 1960s [Culler et al. 1963]. It was suggested that the link between computers and humans could be strengthened to allow engineers to more readily and easily solve engineering problems without being constrained by the knowledge of the computer code, graphics, or numerical tools necessary to solve difficult engineering problems. A PSE is a computer system that provides all the computational facilities necessary to solve a specific class of problems [Gallopoulos et al. 1994]. The PSE encompasses everything that is needed by the engineer to adequately and easily design a system. At its core, a PSE can be used to solve a variety of problems, from a simple algebraic manipulation in a spreadsheet to a multi-component system optimization. Some examples of simple PSEs that were revolutionary when first introduced are the spreadsheet, which replaced the calculator and ledger; and three-dimensional CAD modeling, which replaced prototyping phases in the manufacturing process.

Currently, there are many PSE software packages, such as Refiner, providing a graphical user interface to construct mathematical solvers [Hunt et al. 2002]; PYTHIA, utilized to aid in the selection of tools for solving a systems of equations [Weerawarana et al. 1996]; CARM-PSE for studying reduced chemical kinetic mechanisms [Montgomery et al. 2002]; and iSIGHT for performing systems analysis [Engineous Software 2007] for scientific research. These software packages enable scientists and engineers to solve problems and design systems more rapidly and not be concerned with the underlying algorithms or APIs. These scientific PSEs are becoming common within the engineering design process. Previously, compute resources were the limiting factor in the usability of

www.manaraa.com

PSEs; today, the algorithms and numerical techniques necessary to build a robust PSE are the limiting factors.

One application is coupling a PSE with a Domain Knowledge Based (DKB) Search Advisor for use with a Design Exploration Systems (DES) [Ong et al. 2002]. A DKB Search Advisor contains information for a specific problem that helps the engineer specify the optimal solutions for solving an engineering problem. This feature, coupled with a DES, enables engineers to solve problems more efficiently. As Ong et al. note, if both positive and negative design results are stored for the respective engineering decisions, engineers can avoid using the same design variables in the next design cycle. This type of design process would enable problems to be solved in an environment where an engineer can positively affect the outcome of a product through the incorporation of past design experiences without requiring the presence of past team members.

There are user-interaction limitations (e.g., interrogating large three-dimensional datasets) that can be solved be utilizing virtual reality [Belleman et al. 1998] and other human-computer interaction devices [Drashansky et al. 1996]. A PSE coupled to a three-dimensional immersive environment is more useful to the designer because the designer is now in the solution and a part of the solution. This is the primary advantage of incorporating virtual reality into engineering because it provides a medium through which information can be presented to many audiences in a meaningful and quickly understandable manner. When this medium is coupled with an expert in the area of interest (e.g., a plant engineer, designer, or construction manager), virtual reality facilitates breakthroughs in the engineering process because the large datasets created by analysis become readily accessible to the engineer [McCorkle et al. 2003].

المنارة للاستشارات
www.manaraa.com

Current PSEs, while excellent tools for solving specific problems, do not address all the tools necessary to engineer a large-scale system. These environments aid an engineer in solving a problem by handling much of the work that the engineer previously completed by hand. The aspect of creating software frameworks to help manage difficult tasks for the engineer will be in the research discussed here.

Shape optimization [Mohammadi et al. 2002, Mohammadi et al. 2001] has become a widely accepted design technique in engineering and a key component of PSEs. Shape optimization problems deal with geometric shape changes and design variables that are tied to the solution of a problem such as airfoils [Makinen et al. 1999, Jang et al. 2000, Quagliarella et al. 2001], heat exchangers [Fabbri 1998, Schmidt et al. 1996], two-dimensional blade profiles [Trigg et al. 1999, Fan 1998], missile nozzle inlets for high-speed flow [Blaize et al. 1998, Zha et al. 1997], three-dimensional shape optimization [Foster et al. 1997], sailing yacht fin keel [Poloni et al. 2000], and stoves [McCorkle et al. 2003, Bryden et al. 2003]. Many engineering optimization applications can be reduced to shape optimization problems because the primary problems confronting engineers are the development of physical parts required to meet specific design constraints. The primary interface to most engineering problems is through their geometric representations with CAD data. The ability to interactively change geometric representations and have that information coupled to the underlying physics models is important to the development of an engineering PSE so that engineers can improve the product realization process.

This research also discusses extending the ability to do shape optimization interactively with high-fidelity models that require intricate meshing routines and model preparation [Abodeely 2007, Engelbrecht 2007]. The benefit of this is that shape

المنارة للاستشارات
www.manaraa.com

optimization tools provide significant capability to design engineers to solve complex problems on their desktop. Others [Kanukolanu et al. 2006] are investigating the use of visualization techniques to enable the engineer to better understand how constraints on a system under investigation trade-off with performance of the system enabling optimal solutions to be identified faster. Shape optimization with finite element analysis as the fitness evaluation is being used across a family of products [Torstenfelt et al. 2007]. The use of surrogate models to enable shape optimization of two-dimensional airfoils is being investigated to improve the overall performance time of the shape optimization algorithms [Jouhaud et al. 2007]. These tools continue to provide examples of modules that must be accessible from within an advanced engineering framework.

2.3 Object-oriented programming

Object-oriented programming is a software engineering paradigm for managing large amounts of data through software interfaces in a structured manner. The first applications of object-oriented programming were in simulation (i.e., SIMULA) and graphical user interfaces (i.e., SmallTalk). Each of these applications focuses on working with large amounts of structured data. Supporting the object-oriented programming paradigm requires utilizing a language that supports a hierarchical and modular method for software development, such as C++ and Python. Typically, an object-oriented language will define an object (i.e., class), which is the wrapper for data (i.e., variables) that will be utilized and available to other objects. The data and functionality contained within the object are then made available to other objects through methods (i.e., functions). These methods have explicit interfaces that must be utilized to access the data within the object and make the object perform a specific task.

المنارة للاستشارات
www.manaraa.com

The methods with which these objects are implemented become important to the reuse and robustness of the software being developed. In the past, a top-down approach was taken with software development that resulted in a process-oriented view of an application. This type of method does not promote software reuse or enable data to be managed hierarchically. In part, this design approach was required due to the programming languages available at the time, such as FORTRAN. An object-oriented method overcomes these two limitations and is enabled through object-oriented languages. In addition, the object-oriented methods utilized in software development focus on low-level object-to-object interactions, thus resulting in a bottom-up design approach with a high degree of modularity [Baldwin et al. 2000, Baldwin et al. 2006] and reuse available from the resulting software. An abstract method for managing the development of robust objects is through the use of object-oriented design [Booch 1982] and design patterns [Gamma et al. 1993]. Design patterns (e.g., singletons, null object, factories) are abstract solutions to data management problems that have been tested and implemented across a broad range of problems.

In the following sections, a brief overview of some early object-oriented languages will be given as well as a brief discussion of the development of objected-oriented design (e.g., design patterns).

2.3.1 Early software implementations of object-oriented programming

In 1965, Kristen Nygaard and Ole-Johan Dahl developed the first object-oriented programming language, SIMULA [Dahl et al. 1966, Dahl et al 1968]. The SIMULA language was developed to enable the "concise description of discrete event systems" [Dahl et al. 1966, p. 671]. An example of such a system is a job shop where multiple

المنارة للاستشارات
www.manaraa.com

machines and the workflow can be easily modeled through an object-oriented approach rather than a process-oriented approach. The use and creation of an object-oriented language to model the job shop workflow enabled reuse of code that would have previously been repeated over and over again in the process-oriented approach. In addition, the object-oriented approach enabled an easier leap from conceptualizing the job shop problem to implementing the simulation in code. Without the object-oriented approach, the code implementation of the simulation is more complex for the developer. When using an object-oriented programming language to create a simulation of a system, it is important to enable the programmer to easily construct a map between programmatic entities and the real world. Even in the late 1960s and early 1970s, simulations had a significant impact on the scientific community as a way to look into the future to see how a system (i.e., disease epidemics, traffic flow) might perform under given conditions [Dahl et al. 1966].

At the same time that SIMULA was being developed, SmallTalk was being developed as another object-oriented language [Goldstein 1980, Kay 1993, Shoch 1979, Kay 1977]. The primary purpose of SmallTalk was to create a "tool utilized in the construction of an interactive computer system, used by both children and adults for problem solving, simulation, drawing and painting, real time generation of music, information retrieval, and other tasks" [Shoch 1979, p. 64]. Because Kay's background was in biology, he wanted to create a language in which characteristics of the physical world were also characteristics of the computer language. The purpose of this connection, much like SIMULA, was to provide a language that the programmer could easily adapt to and understand to enable easier implementation. While creating SmallTalk, Kay was also approached by companies to create tools that would enable their engineers to access the

المنارة للاستشارات
www.manaraa.com

power of the computer by creating higher-level computer programming languages that enabled non-computer experts to harness the power of the computer [Kay 1993]. At this point, computers were becoming smaller, faster, and less expensive, and thus more accessible to the average company, prompting engineers to speculate about the computer's use in the engineering process. This is another example of the necessity for engineers to hide some of the complexity of problems under investigation so they can focus on understanding what the information is trying to tell them rather than on the nuances of the problem. The use of SmallTalk in the development of graphical user interfaces to simulation or text manipulation applications provides an abstraction layer for the developer so that he or she can focus on developing the application rather than on the lower-level interaction with the computing hardware.

2.3.2 Object-oriented design/analysis

In the early 1980s, Grady Booch defined object-oriented design, which specified a method by which software developers could analyze a problem to break it down into software objects. The resulting programs would then be more portable and more understandable from an algorithmic standpoint because the programs would follow key design principles [Ross et al. 1975] such as:

- Modularity
- Abstraction
- Localization
- Information hiding
- Completeness
- Confirmability

المنارة للاستشارات

www.manaraa.com

This methodology was developed because the complexity of computer programming caused software to be characterized as “late, erroneous, and costly” [Booch 1982, p. 64]. Similar descriptors can be used about the current state of our engineering product development process: high-end computers, networks, and software are being used, but many products are still delivered late and outside current customer design requirements. These problems arise because of the complexity of the engineering products being developed. The parallels between the software engineering domain and product engineering domain can be seen, which leads to the observation that many of the tools utilized to solve the software engineering domain can be applied to the product engineering domain.

In addition to the points about object-oriented design, Booch discusses the importance of a different approach to design other than top-down. As Booch notes, “In an object-oriented design approach, we take a broader view of modules as collections of computational resources. Such modules may represent abstract data types in addition to abstract operations” (1982, p. 65). Taking this approach, he further elaborates that the resulting solution is more robust and results in a modular design for increased software reuse. These attributes help overcome the limitations of a top-down approach to software development. Similarly, in the engineering design process, many of the processes implemented in companies and taught is a top-down approach to product realization. Again, the history of software engineering would predict that this is a portion of the cause for the deficiencies in the products being engineered today. Software engineering history would then predict that the engineering design process must change to a bottom-up approach. Currently this is not possible due to the lack of enabling technologies in place to

المنارة للاستشارات
www.manaraa.com

support such an effort. This research will draw on the work of Booch to help begin to create some of these enabling technologies. The process of creating software from the bottom up can be enabled by the use of design patterns [Gamma et al. 1993]. According to Gamma et al., "They [design patterns] preserve design information by capturing the intent behind a design" (p. 407). Currently, design patterns are being investigated for use in the product life cycle management process [Framling et al. 2007]. The principles outlined by Booch and Gamma will be leveraged in the creation of the engineering framework presented in this research by providing proven methods for preserving design information, creating robust designs, and enabling a bottom-up design approach to product design.

A byproduct of Booch's work in object-oriented design is the development of the UML. UML enables the user to map out the connections between software objects, which is done to show the relationship between objects either for composition or for hierarchy purposes (Figure 1). Again, a tool similar to UML or that utilizes UML could be used in engineering to help enable some of the qualities that Booch outlined in engineered products. The language also enables the relationship of variables between objects and the method of access to those variables to be depicted (Figure 2). Some recent researchers have proposed the use of the UML language to describe physical systems [Fishwick et al. 1996a, Fishwick et al. 1996b] to aid in the virtual prototyping of products. This is an example of applying proven tools to help enable an improved engineering process. While it is a start, there is much more that needs to be done to enable engineers to create products through a bottom-up approach, resulting in more modular and robust designs. Specifically, it is proposed that the UML language enables a macro-view of a system and provides an object-to-object relationship rather than just a functional relationship as many systems

المنارة للاستشارات
www.manaraa.com

analysis tools provide [Huang et al. 1993]. This is being realized to some extent in the creation of the SysML [Object Management Group, Inc. 2008] specification, which not only characterizes the specific attributes of individual components of the system, but also adds the connectivity of these components to each other. The functional relationship does not capture a comprehensive representation of a component in a system; it simply provides its capability to the rest system.

One object-oriented language that is commonly used today is C++. It was developed in 1984 by Bjarne Stroustrup to provide a higher-level language by which object-oriented ideas could be implemented. C++ is based on SIMULA and was implemented to give users access to the efficiency and flexibility of C while maintaining the modularity and object-oriented nature of SIMULA [Stroustrup 1993]. During the mid- to late-1980s, C++ was still being refined and was not readily accessible. It was not until the early 1990s that C++ became accessible to a broad range of users. The accessibility of the language was probably due to the increased use of personal computers and the development of commercially integrated development environments such as Visual Studio™ from Microsoft™. The combination of software tools and decreased hardware computing costs enabled researchers to begin utilizing object-oriented software to manage complex problems.

المنارة للاستشارات
www.manaraa.com

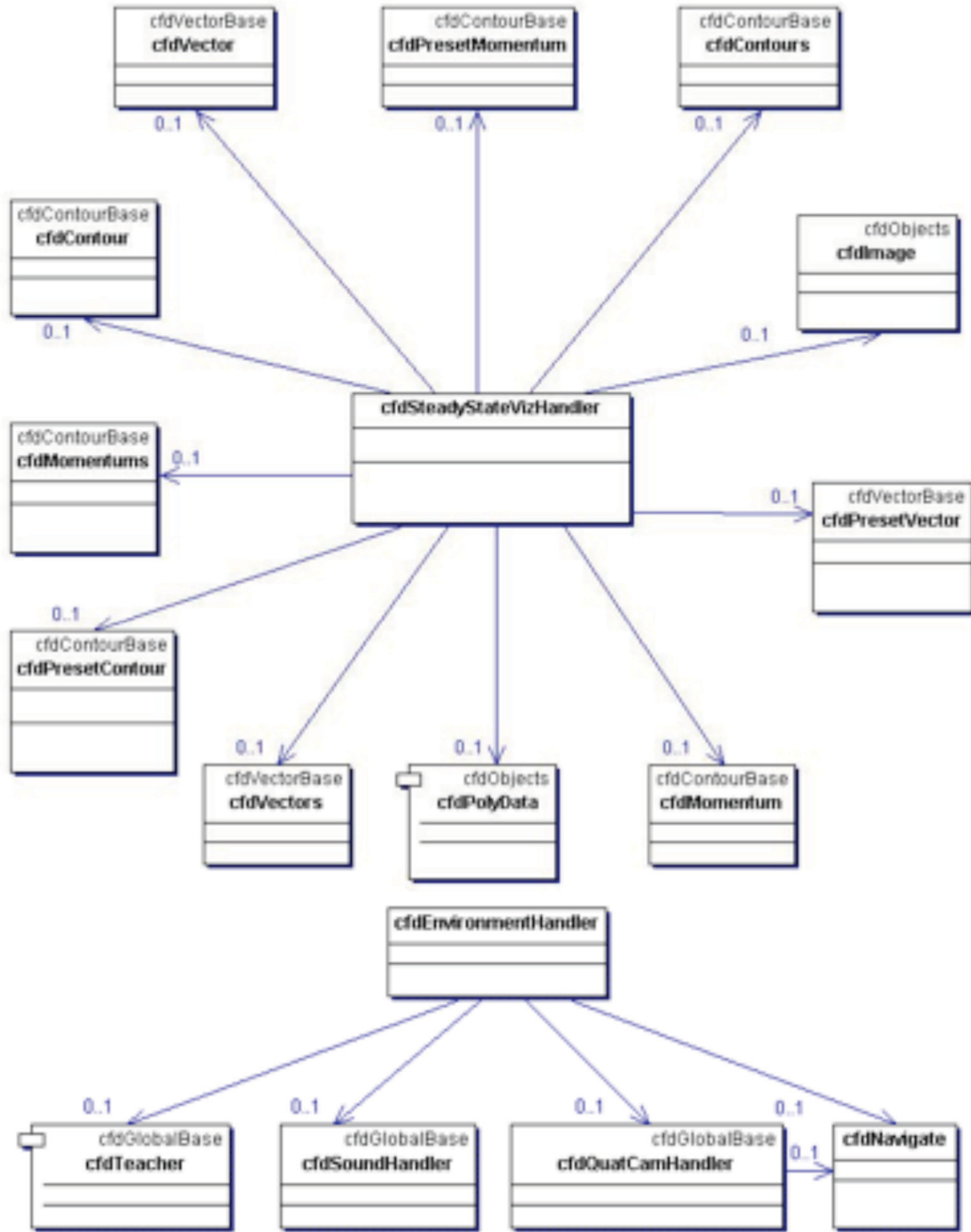


Figure 1. UML class relationship diagram

المنارة للاستشارات
www.manaraa.com

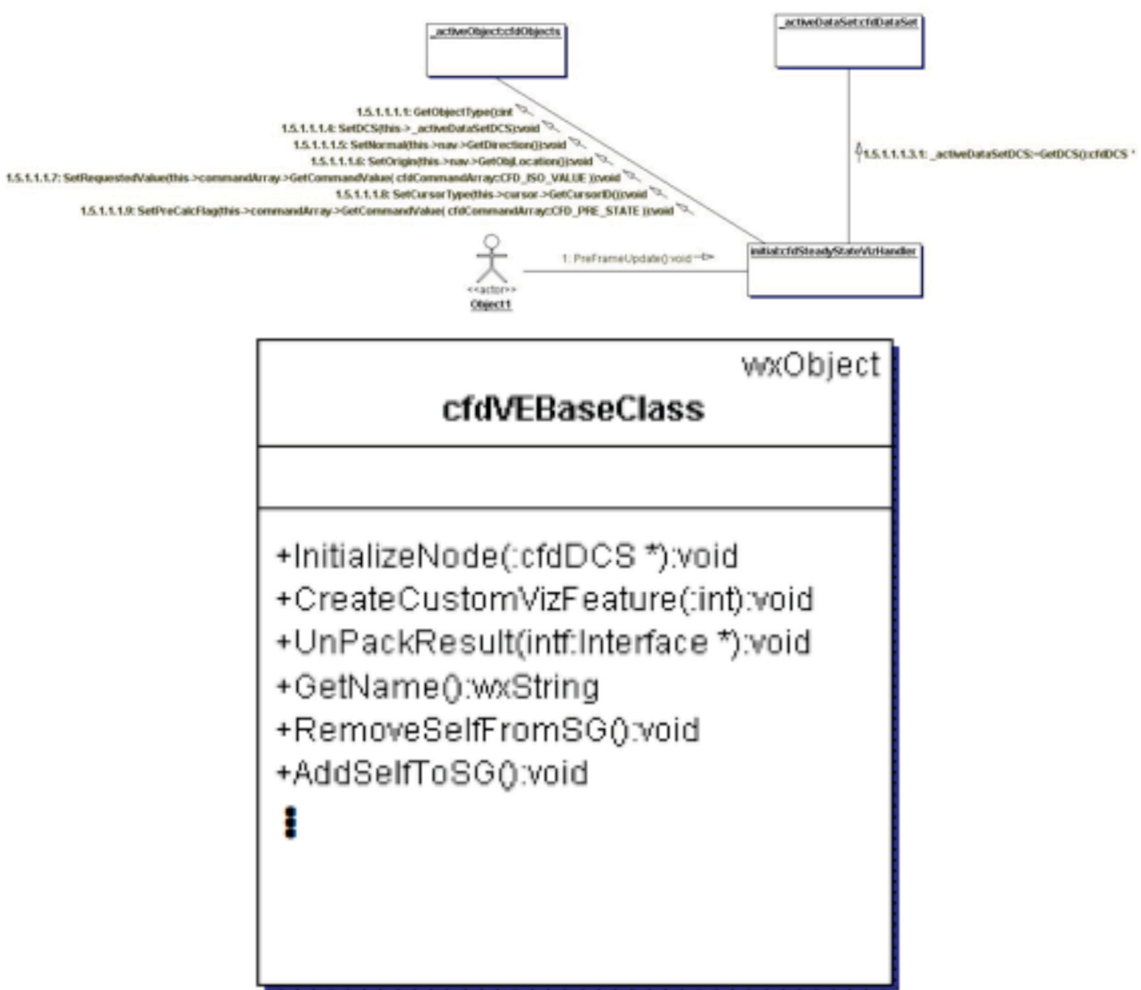


Figure 2. UML variable relationship diagram

المنارة للاستشارات
www.manaraa.com

2.4 Current Uses of Object-Oriented Methodologies

As object-oriented programming languages have become more prevalent, many engineering disciplines have begun to adopt the methodologies (e.g., object reuse, hierarchical data storage, object inheritance and composition) upon which object-oriented languages are founded as a mechanism by which engineers can more easily solve engineering problems. These methodologies include PACT, which was created to enable concurrent engineering systems [Cutkosky 1993]; NetBuilder, which is used to construct collaborative engineering environments [Dabke et al. 1998]; NODES, which supports the conceptual engineering design process [Duffy et al. 1996]; STEP, which generalizes product description [Männistö et al. 1999]; mechanisms that link CAD to disparate engineering processes [Martino et al. 1998]; SHARED, which is used to construct graphical collaborative engineering environments [Toye et al. 1994, Wong et al. 1993]; web-based collaborative concurrent design tools [Xue et al. 2003]; and software tools that integrate design and assembly planning [Zha et al. 2000]. These languages typically enable developers to use a modular approach to segment information into a format that loosely couples the modularity to information in the real world. In addition, it permits a structured approach to querying for information throughout an application, allowing a semi-intuitive approach to hierarchically present information and accessibility. Some examples that implement this technology follow.

www.manaraa.com

```
dynamic class samplecircuit {
  attributes
    resistor R;
    real [] voltages;
    voltagesource B;
    capacitor C1
    inductor I1
    componentEnd R1,R2,B1,B2,C1,C2,I1,I2;
    node N1,N2,N3;
  constructors samplecircuit() {
    R = new resistor(10);
    C = new capacitor(0.2);
    I = new inductor(0.1);
    Time[ 1] = 0;
    Voltages = 10 * sin(0.1 * Time);
    B = new voltagesource(Voltages);
    B1 = new componentEnd(B,1);
    B2 = new componentEnd(B,2);
    R1 = new componentEnd(R,1);
    R2 = new componentEnd(R,2);
    C1 = new componentEnd(C,1);
    C2 = new componentEnd(C,2);
    I1 = new componentEnd(I,1);
    I2 = new componentEnd(I,2);
    N1 = new node([ C1,B1] );
    N2 = new node([ B2,R1,I1] );
    N3 = new node([ C2,R2,I2] );
  }
}
```

Figure 3. Constrained object description file [Pushpendran 2006, p. 25]

www.manaraa.com

The term "constrained objects" is derived from the fact that an object's physical constraints are programmatically encoded into the class that represents the physical object. Constrained objects were first implemented to extend programming languages and to enable the user to create a lightweight ASCII text file defining the numerical/physical constraints of an object (Figure 3). Some examples of these constraints are body forces on beam trusses [Wilson 2000, Wilson et al. 2001], Ohm's Law for electrical circuitry [Tambay 2003], and other physical phenomena-governing equations [Horn 1993, Peak 2002, Pushpendran 2006]. The object in this case, although very similar in form to a programmatic object, is a copy of the physical object that it is representing. The representation of the physical object is only as good as the constraints that are implemented in the object. These constraints then become the limiting factor of the objects. Enabling a high-fidelity representation of an object would require a sophisticated modeling language and software framework adhering to the formatting in the ASCII text file illustrated above. This type of implementation is very similar to the modeling language Modelica [Modelica Association 2008], which targets control-type problems [Wilson 2000, Wilson et al. 2001].

Fishwick [Cubbert et al. 1998, Fishwick 1996a, Fishwick 1996b, Fishwick 2006, Hopkins et al. 2001a, Hopkins et al. 2001b] proposes objects as a method for simplifying the compilation of numerical models when trying to construct multi-model environments. The environment directly leverages the tools and design patterns developed for C++, such as object-oriented programming and UML tools. Through these mechanisms, the object-oriented physical multimodeling and multimodeling object-oriented simulation environment (MOOSE) environments enable users to construct a simulation environment

المنارة للاستشارات
www.manaraa.com

through similar hierarchies and inheritance schemes that are available in other object-oriented languages. The proposed software architecture would then enable users to construct a wheel from a series of other objects such as a nut, rim, and tire. Each of these objects would then provide their physical characteristics through numerical simulations to the other objects in the environment. This work highlights the positive impact that object-oriented methodologies can have on domains other than computer science [Fishwick 2004]. However, this research does not propose a solution for enabling the overlaying of disparate sources of information needed to describe an object.

Product-centric objects leverage the benefits and philosophies of object-oriented programming by creating product agents that provide an interface to individual product information on a physical object basis. One software toolkit built on these principles is Dialog [Dialog 2007]. The implementation of these software agents utilizes object-oriented principles, but more importantly, the agents are what the Dialog framework accesses to gather object-specific information as requested by the user. This is done through the use of URIs and a GUID. The URIs and GUID are essentially a unique data tag that enables information to be queried from anywhere on the web by specifying a location (i.e., the URI) and the GUID of the object being queried. The motivation behind this research is to give product manufacturers, product distributors, and original equipment manufacturers seamless access to per-part information from any location to enable streamlined product delivery systems. The benefits of the product-centric process are that it is able to scale to large systems of part databases, is open source, and can be implemented within a company with very few changes to information technology infrastructure. These attributes make it accessible to large and small companies and enable the inclusion of a broad range of

المنارة للاستشارات
www.manaraa.com

product database implementations. While the product-centric objects enable access to part-level information such as dimensions, quantity on hand, time-to-ship, and other manufacturing level data, these objects do not propose to address issues surrounding modeling and simulation.

Pattie Maes at the MIT Media Lab has developed a series of applications—Invisible Media [Merrill et al. 2005], ReachMedia [Feldman 2005, Feldman et al. 2005], and Galatea [Gatenby 2005]—that provide a software toolkit to enable users to interact with physical objects and gather meta-information that is not available through traditional interfaces with physical objects (e.g., touch, sound, sight). These tools provide the user with a more intuitive interface to and information about an object than is possible through non-augmented interfaces. Some examples of this type of information are repair history and part traceability. These objects, which have been used in frameworks developed by Maes et al., illustrate the improved knowledge and assistance that is available when the computer can augment the user's expertise and utilize environmental information to solve problems. These objects provide an illustration of how overlaying multiple pieces of information (e.g., working with physical objects and overlaying repair instructions) can aid the user in gaining better insight into the object under investigation.

Another current implementation of object-oriented concepts is the Common Information Model (CIM), which "describes management information and offers a framework for managing system elements across distributed systems" [Bumpus et al. 2000, p. 1]. CIM has a specification and schema that allow it to be applied to a wider variety of problems, to adapt to resources that change within the framework, and to change the information that the resources provide. Again, the object-oriented methodology was

المنارة للاستشارات
www.manaraa.com

applied to enable developers to map from the real model to a conceptual model that could then be created programmatically. CIM uses object-oriented methodology as it is used in the programming world: to define the basic unit from which all other entities within the framework are created. The CIM framework provides a further example of object-oriented methods, enabling complex information to be handled through object interfaces.

Knowledge objects are entities that hold business-related organizational information but have also been used to hold technical information [Simpson 2004]. A knowledge object is defined as "a highly structured interrelated set of data, information, knowledge, and wisdom concerning some organizational, management or leadership situation, which provides a viable approach for dealing with the situation" [Bellinger 2004]. Knowledge objects provide organizations with tools and guidelines to construct concise packets of information. The objects contain organizational information so that future business decision makers can benefit from the past experiences of others and gain insight into the positive and negative outcomes of previous endeavors. The construction of these objects is based on a set of rules determined on a per-company basis based on their experience of what has aided decision makers in gaining insight into past successes. Again, the goal is to use these objects to encapsulate information and provide an intuitive interface for users to gain a level of understanding that would previously have been unattainable due to past information being lost through employee turnover.

Knowledge objects are constructed to implement a higher level of engineering effort, referred to as "knowledge engineering." The goal of knowledge engineering is to encapsulate knowledge that organizations create or obtain so that engineers working on similar future projects can avoid the shortfalls of the teams before them. Knowledge

المنارة للاستشارات
www.manaraa.com

objects are also simple in that only pertinent information is stored, making the object extremely compact. Much of the current research in this area is now focused on ontology development. This is due to the fact that the foundation of knowledge objects is an assumed comprehensive capture of someone's knowledge. This is only possible if the data obtained from someone can be interpreted five or ten years in the future without the person present. One of the current methods for doing this is through the use of semantically rich ontologies. Unfortunately, this approach can often require extensive work to provide a comprehensive schema for small problem domains. Even though these objects may not be well suited for engineering modeling and simulation information, they do provide an example of and illustrate the value of capturing information on a per-object basis when creating a product for a company.

2.5 Frameworks

The definitions of "framework" are varied and can refer to software libraries, software applications, structural components of a building, and everything in between. A general definition is "a basic structure underlying a system, concept, or text" [Soanes et al. 2005, p. 368]. Regarding the discussion in this research, *framework* will refer to a software application that is the basic structure utilized to understand complex systems. Currently available frameworks include a host of open-source and commercial packages. Examples of open-source frameworks include:

- the University of Utah's SCIRun package used for scientific visualization and computational steering [SCI Institute 2008]
- dataflow visualization-oriented packages such as OpenDX [OpenDX.org 2006]

المنارة للاستشارات
www.manaraa.com

- the Common Component Architecture (CCA)-capable CCaffeine [Allan et al. 2005] used for the numerical integration of large distributed simulation (e.g., nuclear simulation, munitions simulation)

Examples of closed-source packages include:

- Matlab's Simulink [The MathWorks, Inc. 2008], used to integrate third-party software such as LMS Virtual.Lab [LMS International 2008] with the Matlab
- Fiper [Engineous Software 2007], used for distributed collaboration of design teams. This package has been customized primarily for GE.
- Aspen Plus [AspenTech 2008], utilized for chemical process plant simulation
- ModelCenter [Phoenix Integration 2008], used to integrate a wide range of third-party solvers (e.g., Excel™, user subroutines) with optimization and design space exploration
- Protrax [Pro-Trax Off-Road Adventures 2008], used to model large plants at a system level

These packages tend to be targeted to specific applications (e.g., Aspen Plus to chemical process modeling and CCaffeine to terascale-level high-performance computing) and do not address the general engineering process. SCIRun has computational steering capability and visualization support but does not provide an extensible method for integrating generic simulation and modeling tools. ModelCenter, Fiper, Protrax, and Matlab's Simulink all have support for the integration of specific sets of tools or for high-level systems modeling capability. Each of these packages fills a specific commercial need and provides a desired set of tools for a specific clientele.

المنارة للاستشارات

www.manaraa.com

Padula et al. [2006] noted that the main issues facing the development of software frameworks are:

1. the verification and validation of federated simulation environments
2. knowledge capture stemming from these large federated simulation environments
3. easy access to construct large simulations through graphical displays

One of Padula et al.'s key ideas is that many frameworks center around creating data repositories that tie information to the components they represent. These repositories then enable the users of the frameworks to seamlessly query information on a per-component basis. This work highlights the difficulty in creating a software framework to begin to address the other issues outlined by Padula et al. when the primary work to date has focused on creating a sufficient software structure to enable the ease of access to component-level information for large simulations.

One software engineering toolkit that takes advantage of the object-oriented methodology is the Distributed Object-based Manufacturing Environment (DOME) [Abrahamson et al. 1999, Abrahamson et al. 2000, Pahng et al. 1997, Pahng et al 1998, Senin et al. 1999a, Senin et al. 1999b, Senin et al. 2003a, Senin et al. 2003b, Wallace et al. 2001, Cao et al. 2005]. This software uses CORBA [Object Management Group, Inc. 2008] combined with customizable graphical user interfaces to set up simulations with multiple models and access variables within the DOME framework. It maps objects, which are very closely tied to the real world rather than the programmatic or algorithmic world, to their mechanical characteristics to enable distributed simulation. With the DOME framework, the developer can wrap and hide unnecessary proprietary information within a

المنارة للاستشارات
www.manaraa.com

module while exposing the necessary information to other collaborators on the distributed framework.

Another interesting concept that the DOME framework proposes is the World Wide Simulation Web (WWSW). The goal of the WWSW is to be the structure by which numerical simulations can transfer information from one location to another, much like the World Wide Web does with hypertext.

The Building Design Advisor (BDA) employs object-oriented techniques to create a software framework that integrates various numerical models for building construction [Papmichael et al. 1997, Papmichael et al. 1999, Reichard et al. 2005]. These models are integrated together on a per-object basis in a building, such as a door, window, or roof, as objects in the modeling advisor. This approach is taken to help the end-user better identify the model that is actually being designed with the real world. In this design environment, the BDA's goal is to guide the decision maker from a conceptual design to a very detailed design. This framework provides a good example, albeit to a specific domain, of how managing models on a per-object basis provides flexibility in the software framework.

Reed [Reed 1998, Reed et al. 1994, Reed et al. 2000a, Reed et al. 2000b] proposes the use of object-oriented principles for enabling the integration of turbine engine models in a distributed manner. The ONYX [Reed 1998] framework treats each of the major components of the turbine engine as objects in a larger simulation. Each of these objects is then represented in the framework by a numerical model. Each model can be of varying fidelity and a particular object can be comprised of multiple numerical models. In this software framework, the data types and integration interfaces are all predefined to support the turbine engine design problem. The notion of a general software framework to address

المنارة للاستشارات
www.manaraa.com

large time-dependent simulations or integrated visualization capability is not proposed. This work provides a concrete example of the use of object-oriented principles to enable the computer to manage some of the integration tasks for the design engineer.

2.6 Meta Data and Semantics

Common methods in the engineering community to classify and store information have focused heavily on the graphical representation of systems, often referred to as CAD data. This has resulted in CAD formats such as STEP model data. This format offers software package-independent solutions for the storage and representation of information in the engineering process, but only provides information similar to what was discussed with product-centric objects. These CAD representations often refer to the surface geometry representation of a particular object. While this representation is the most visible to an engineer, it holds relatively little information about an object. The geometry object only defines an object's boundaries and the space it occupies.

Before an engineer's inquiries can be satisfactorily answered in the virtual world, appropriate representations (e.g., economic, pedigree, experimental, numerical models, geometric) for the problem at hand must be provided. For example, supplying fluid properties to a graphics program probably has little benefit for the graphical environment. A more appropriate representation may be a polygonal mesh that can be rendered and would display the physical domain within the virtual world, providing the engineer with the appropriate information given his or her requests. For a purely results-based request, a single scalar value would be returned to the user. For a "why" or "how" request, the entity that generated the request would need to provide the foundational information such as the finite element analysis results or computational fluid dynamics results.

المنارة للاستشارات
www.manaraa.com

These representations that are attached to the virtual object are meta-data providing more meaning to the object than just the geometric mesh. This approach to building environments where meaning is being attached to an object is not new. The Semantic Web's purpose is to attach meaning to the current World Wide Web of data (i.e., web pages, intranets, and wikis); hence the term "semantic" in Semantic Web. Some of the current research in creating meta-data-rich environments will be reviewed below.

2.6.1 Engineering Information Storage

Horváth proposes that objects serve as the basic data structure for providing mapping to a CAD representation of a product [Horváth 1997, Horváth et al. 1994, Horváth et al. 2001, Horváth et al. 2003, Horváth et al. 2004a, Horváth et al. 2004b, Horváth et al. 2004c, Horváth et al. 2004d]. Horváth also notes that there are multiple representations for a component beyond just the geometric information being displayed in a CAD program. In this research, CAD data is brokered between various components, and file specifications such as STEP and EXPRESS are utilized to store lifetime information about an object. This work highlights the requirement that lifetime information about an object is critical to the design and retrieval of design intent after engineers have finished a project.

A framework developed by Wang [Wang 1993] addresses the need to provide a means to map exceptions thrown during the manufacturing process and notify the engineering team. The development of a method to map information to objects makes this possible. The information is then tracked through the manufacturing process, giving the engineering team a clear picture of their product's quality.

المنارة للاستشارات
www.manaraa.com

Bliznakov [Bliznakov 1996, Bliznakov et al. 1996] addresses the need to develop a taxonomy to store information about parts during their lifecycle. This is needed to enable comprehensive part tracking to improve communication about product development and to improve the company's knowledge storage capability. This information is critical to enable companies to fix or avoid problems that plagued previous products in future products.

Qureshi [Qureshi 1997] notes that there are two main types of integration: static and dynamic. Static integration requires explicit definition of objects that belong to the integrated information. Dynamic integration follows a predefined scheme to dynamically generate the definition of objects belonging to the integrated information. The research in this dissertation focuses on using dynamic integration. Qureshi also notates several integration categories:

1. No integration
2. Direct interfacing (need-based integration)
3. Neutral format-based integration
4. Loosely coupled integration (open architecture)

Qureshi uses this work on integration to generate a specification for integrating information throughout the design process to not only record explicit information, but also implicit information about the process to generate a comprehensive record of the product being designed.

The trend in engineering informatics over the past few years has been product life cycle management, which focuses on managing a product's descriptions and properties throughout its development and useful life, mainly from a business/engineering point of view. Product life cycle management tools primarily exist as enterprise-wide software

المنارة للاستشارات
www.manaraa.com

toolkits that span disciplines and provide a common interface for a product. Some examples of these tools include the Federated Intelligent Product EnviRonment (FIPER) [Sampath et al. 2002, Wujek et al. 2000], TeamCenter, and Dassault Systems. These tools primarily present information in a format that requires the engineer to dig for a simple dimension. For example, if an engineer wants to change a diameter on a component in a complex CAD assembly, he or she must navigate a deep hierarchical tree to potentially find the parts and features that need adjustment. Then, to make the same adjustment in other models associated with the same CAD file, the engineer must repeat the process for each model because the CAD geometry typically does not automatically update the other associated models.

Other techniques for managing information in engineering are primarily constrained to geometric information (e.g., the STEP/IGES specification), which provides a mechanism to allow disparate software packages to interoperate and exchange information. This requires an information framework that is open and accessible to all entities.

The Semantic Web, which is often referred to as Web 3, has been under development for the past 5–8 years [Antoniou et al. 2004], and is proposed by the creator of the first web, Timothy Berners-Lee. The Semantic Web would provide context and meaning for data (e.g., web pages) on the current web. For example, when visiting a webpage about a conference, the browser would check the conference dates against a personal calendar and inform the user of any scheduling conflicts. If no conflicts were found, then that particular webpage would provide the user with local information such as nearby hotels, restaurants, and other attractions.

المنارة للاستشارات
www.manaraa.com

The ability of computers to perform intelligent tasks such as checking for schedule conflicts is accomplished through a series of open interfaces and schemas that are implemented via open source libraries and standards. The Semantic Web's core technology is "the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming" [W3C 2008]. XML provides a format that allows data structures to self-describe and provides a means to represent the data that it contains in a format readable by humans. The ideas upon which the Semantic Web is founded, along with the technology that is used to implement it, provide a platform on which virtual engineering tools and interfaces can be extended to create a web in which contextual information is readily accessible to engineers. They also provide a means by which the product development cycle can be completed in a manner unlike any before. When the Semantic Web and virtual engineering methods are fully realized, computer hardware and networking capabilities will work to provide information and tools to access information meaningfully. In today's computing age, the following question must be answered: How will information be integrated so that commercial and proprietary software tools can remain separate while also being integrated so that the end user can control and query these tools with little to no knowledge about their implementation or inner-working details? The answer to this question will depend largely on the ability to harness a large group of individuals to implement the tools necessary to complete the work, which will require open interfaces and schemas that can evolve over time as well as open source toolkits that enable development teams to collaborate at a high level.

www.manaraa.com

Physiological level	Types of model	3D Imaging devices
Organ systems cardiovascular, respiratory, musculo-skeletal, digestive, skin, urinary, nervous, endocrine, lymphatic, male reproductive, female reproductive, special sense organs	**Systems theory**	**MRI, CT, PET** **3D Ultrasound** 1mm or 10^{-3}m
Organ 50 organs	**Continuum theory** Conservation of mass Conservation of momentum Conservation of charge	
Tissue Epithelial Connective Muscle Nerve	**Conservation equations** Passive flux equations Carrier mediated transport Electroneutrality constraints	**MicroCT, Optical Coherence Tomography** 10μ or 10^{-5}m **Serial sections** 1μ or 10^{-6}m
Cell 200 cell types	**Conservation equations** Passive flux equations Carrier mediated transport Electroneutrality constraints	**Confocal microscopy** 1μ or 10^{-6}m **2-photon microscopy** 100nm or 10^{-7}m
Organelle Cell membrane, mitochondria, nucleus endoplasmic reticulum, ribosomes, Golgi apparatus, centrioles Lysosomes, peroxisomes.	Continuum models Stochastic models	**Electron tomography** 1nm or 10^{-9}m
Cell function Membrane receptors Membrane ion channels Signaling pathways Metabolic pathways Transport Motility Maintenance Cell cycle		
Proteins, carbohyrates & lipids	Molecular dynamics Quantum mechanics	**Xray diffraction** 1A or 10^{-10}m
Post-translational modifications **Protein folding** **Translation** **Post-transcriptional modifications** **Gene regulation** **Transcription**	Markov models Boolean network models	
Genes 19,000 genes		

Figure 4a. Human systems [Kriete et al. 2005, p. 385]

www.manaraa.com

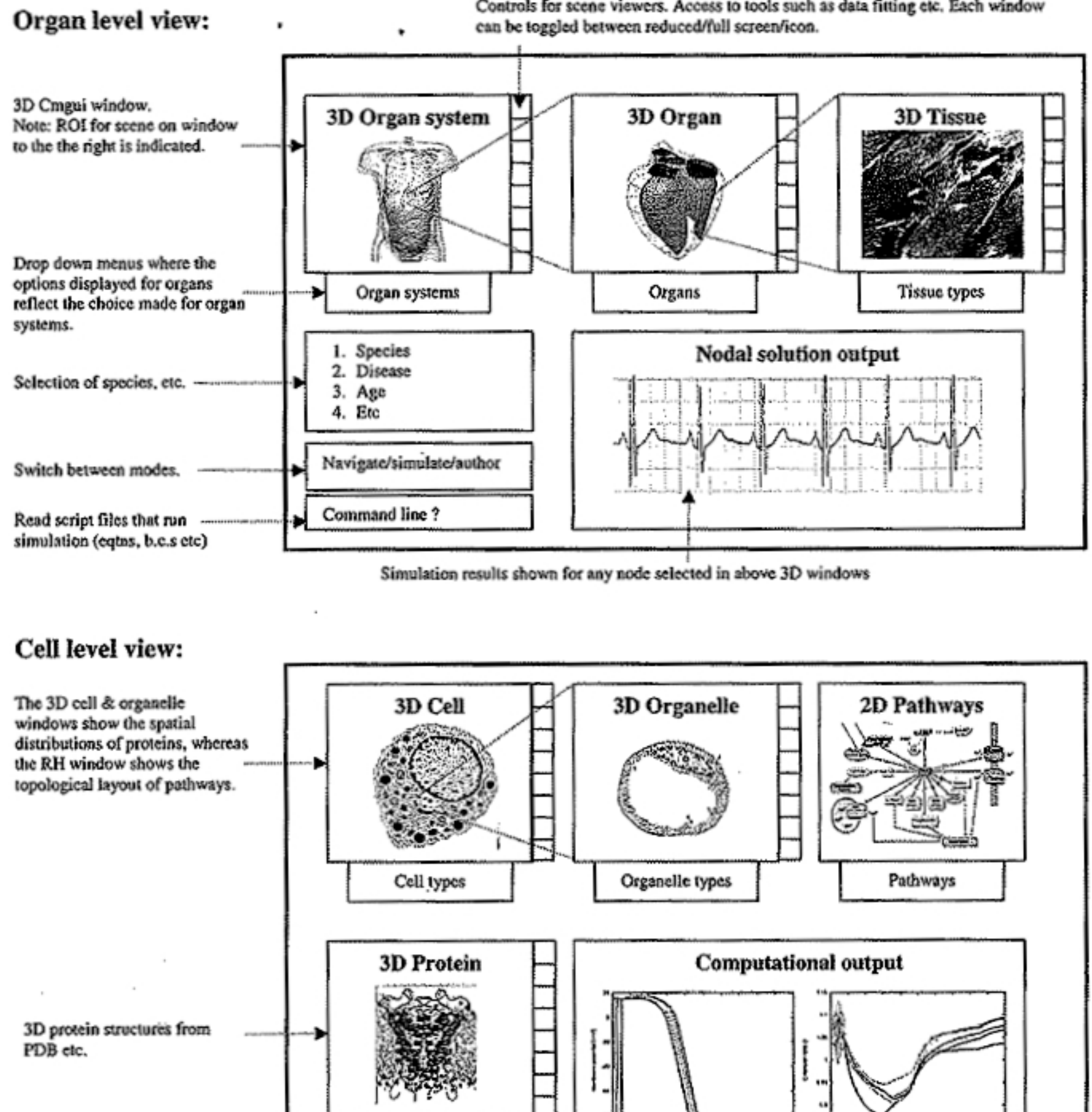


Figure 4b. Human systems [Kriete et al. 2005, p. 391]

المنارة للاستشارات

www.manaraa.com

Other domains are utilizing the tools from the Semantic Web to enable a disparate research team to collaborate across different software tools and research methods. One such research field is computational systems biology, which aims to model the complete human body at all scales from genes to cells to tissues to organs [Kriete et al. 2005] (Figure 4). The modeling of the human body at this level is needed to better understand the physiological function of the healthy and the diseased body. The level of integration required to enable the numerical coupling of these systems requires that the data being shared at each scale provide contextual information to enable the model receiving the information to understand how to interpret the information. Computational systems biology is leveraging ontologies to enable collaboration and exchange of experimental and numerical bioinformatics data to increase the dissemination of results and the longevity of the data [Kriete et al. 2005].

2.7 Virtual Worlds

To create the more efficient and inclusive engineering environment discussed in this paper, the work that has already been done surrounding virtual worlds must be leveraged. Virtual worlds are becoming a popular medium for learning, training, gaming, and many other activities. Popular virtual worlds include Second Life, World of Warcraft, SimNation, and many others. These environments have developed into profitable businesses and continue to intrigue a broad and diverse audience. Extending these virtual worlds to help solve problems in business and the defense industry has become a popular research area. One way the military currently uses these environments, for example, is for force-on-force distributed training.

المنارة للاستشارات
www.manaraa.com

Virtual worlds are defined as including "synthetic sensory information that leads to perceptions of environments and their contents as if they were not synthetic" [Blascovich et al. 2002, p. 105]. Some current research areas include developing narrative in interactive worlds [Young et al. 2003] and defining simulation and experiments in virtual worlds [Winsberg 2003]. This body of work will further the development of a framework that is capable of handling large amounts information for working with large and ultra-large systems. In addition, this work will aid in creating an environment where users are "inside an environment of pure information that [they] can see, hear, and touch" [Bricken 1990, p. 1].

In the software toolkit Croquet, objects are utilized to collaborate across a wide area network [Smith et al. 2003]. Croquet objects can be viewed on multiple computers within the same world to collaborate on anything from documents to games. In each of these instances, the form of the object that the domain being used defines and requires informs the thinking and discovery process through which the user is drawn, enabling acquisition of a point in a game, knowledge about a new subject, understanding of a virtual counterpart in a virtual world, or something else. Engineering objects represent the same goal for the engineer.

2.8 Object Definitions

In previous discussions surrounding object-oriented programming languages, object-oriented applications, and object-oriented implementations, the term "object" is often left undefined. However, there are some instances in which the term is defined as it pertains to object-oriented languages, applications, and implementations [Eckert et al. 2003, Foucault 1994]. These instances define an object as something that:

المنارة للاستشارات
www.manaraa.com

1. Has form [Foucault 1994]
2. Contains artifacts and is utilized for discourse [Eckert et al. 2003]
3. In the instance of object-oriented programming, includes the interfaces and methods necessary for interacting with the data it contains
4. In terms of the ONYX, DOME, BDA, is utilized as a data container for the physical object that it represents and may contain some interfaces for other objects to interact with it

When software objects were first implemented by Nygaard and Dahl [Dahl et al. 1966], their purpose was to create a more intuitive connection between the real world and the program being created. This is the predominant theme in many discussions of objects [Heim 1997, Horváth et al. 2004, Pidd 1992, Rothenberg 1986]. Objects allow the individuals interacting with a system to more easily adapt to what the developer is trying to convey. Also, from a programming standpoint, objects enable the programmer to easily create a program with characteristics that more generally resemble the problem being simulated or solved, which is what Kay [Goldstein 1980, Kay 1993, Metz 2001, Shoch 1979] and Nygaard [Dahl et al. 1966] proposed. The definition of the objects used in this research is derived from [Luch et al. 1996] and [Eckert et al. 2003]:

> An entity is just something with a non-empty set of attributes that is typically used as a template for more sophisticated components. An object is an entity with the added constraint that it has a non-empty set of capabilities. Similarly, an agent is an object with a non-empty set of goals, and an autonomous agent is an agent with a non-empty set of motivations [Luck et al. 1996, p. 52].

www.manaraa.com

The definition of agents provides a broad explanation of how computer scientists think about working with objects in terms of computer simulations that model human activity and software agents that are constructed to mimic human behavior. The field of social sciences focuses more on the physical realm of how humans interact with objects:

> We use a wide definition of the term "object" to encompass all sorts of physical and electronic artifacts that can convey meaning in interpersonal communication, but have an existence beyond a single act of communication [Eckert et al. 2003, p. 145].

These definitions provide a low-level illustration of what we, as humans, interact with and how we interact on a daily basis. The objects in these instances are broadly defined and provide a starting place for the discussion surrounding what the term "object" means and how engineering objects build on work from other domains that utilize that term.

In this thesis, objects have a physical counterpart and a correlation from the physical world to the virtual world. The objects encompass a number of physical and digital artifacts that can convey meaning. This ability to convey meaning provides a basis for the ability to construct virtual systems. The object will be able to:

- define their own status
- define their method of operation
- define their method of interaction with other objects
- sense and act on the environment in which the object is situated

Users constructing a simulation should be able to seamlessly assemble parts as in real life, enabling a narrative to be constructed and ending with the production of a detailed part [Dörner 2002], [Skov 2002]. Engineering objects should primarily be able to manage

المنارة للاستشارات
www.manaraa.com

complexity [Sharpe et al. 2000] and enable the dynamic creation and addition of information in the decision-making environment. The objects that will be used to manage complexity are different from programmatic objects in object-oriented programming and are derived from objects as described by the French philosopher Michel Foucault, who says that objects are "the extension of which all natural beings are constituted – an extension that may be affected by four variables. And by four variables only: the form of the elements, the quantity of those elements, the manner in which they are distributed in space in relation to each other, and the relative magnitude of each element" [Foucault 1994, p. 134].

2.9 Characteristics that objects must inherently have

In this research, objects must have inherent abilities that allow them to adapt to surroundings and distinguish themselves from other objects coexisting in the same environment. In discussing this requirement and the methods used to achieve it, many current research areas will be drawn on. The discussion of objects will begin with the work of French philosopher Michel Foucault.

Foucault examined our methods of interacting with our surroundings to gain an understanding of how our surroundings inform us. To handle this level of complexity in information and systems, a method is needed that enables parallels to be drawn between how information and interaction are handled in the physical world and how they are handled in the virtual world. Gaining information about an object in the physical world is typically straightforward. Information about the weight, for example, does not need to be acquired through a third-party interface. This information is easily gained by picking the object up or attempting to pick it up. Holding an object also allows a human to investigate

المنارة للاستشارات
www.manaraa.com

the material or materials from which the object is constructed. The object can also be dropped, which provides information about its mechanical characteristics. Two objects can be picked up to understand how they might interact with each other, although interaction that is not human-driven can also occur between two objects. For example, two objects can attach to each other without direct human interaction.

There are many ways to test an object's properties to gain information about it. In each of these simple interactions, information about an object's temperature, material mechanics, and weight are easily acquired. That is, the information that can be obtained from an object is dictated by the method of the direct interaction with the object. If this simple means of gaining information about objects in the physical world is compared to current methods to gain information about virtual objects, a much different result is experienced. An engineer may work for days to acquire information about a pump's material mechanics properties, fluid performance parameters, spatial information, or many other properties that are easily obtained in the physical world. To overcome these restrictions, virtual objects are proposed that have the same characteristics as physical objects in the sense that any information that can be gained from interacting with a physical object is also available through a single interface—the virtual object—in the virtual world. These objects will have the ability to self-recognize, adapt, and exchange information without user input. One disadvantage of objects in the physical world is that it is often impossible to make a temporary change and then return the object to its original state. This limitation is not present in the virtual world. Many current computational intelligence technologies will be used to allow objects to operate in a self-organizing and

المنارة للاستشارات
www.manaraa.com

self-describing manner so their interactions are enhanced. These functionalities permit virtual objects to behave very similarly to their physical counterparts.

www.manaraa.com

Chapter 3: Towards an advanced engineering framework

Virtual engineering is the act of using technology and information in such a way that all stakeholders can actively participate and understand what the issues are in a system under design. The types of problems that need to be addressed include multi-scale problems, complex systems problems, and ultra-large systems problems. As our abilities to measure, build, and bring arguments together at many scales increase, tools are needed that enable us to design and understand the outcomes of these systems. For example, the tools we develop should enable us to model from molecule to cell to organism or from part to subcomponent to machine. An engineer should be able to approach an engineering problem much like an artist approaches a painting. The painter focuses on how the paint is applied to the canvas in concert with the other colors and shades on the canvas, but not on how the paint is created, contained, or transported. The painter focuses on the multi-scale problem of how individual microscale strokes of the paint work together to create the whole mesoscale painting. This is the same process that design engineers need to go through in creating a complex system. A design engineer similarly needs to be able to focus on how the components of a system work together optimally to solve a problem and not be concerned with the manufacturing process of the part or how it is modeled. A design engineer should be focused on the multi-scale problem of how individual components work together to create a system greater than the sum of its parts.

المنارة للاستشارات
www.manaraa.com

What is needed is a computational framework that enables the design engineer to creatively address problems related to existing complex systems and to create more complex engineered systems. Specifically, these software frameworks should enable design engineers to take a higher-level approach to interacting with information because the way computers are currently used does not enable problems to be addressed any differently than they were 50-60 years ago. This can be seen in the way the physics of engineering problems are examined. Today, the Navier-Stokes equations (CFD) are still used for analysis, but more grid can be used and more detailed solutions can be addressed because computers have more memory and processing power. While more computational problems can be addressed because of this increased processing power, the manner in which design engineers interact with all of these analysis tools and analysis data has not evolved over this same time period. To enhance the way computers are used, software must be created that applies an improved method of processing and interacting with information.

The virtual engineering process embodies activities that other disciplines assume are present in daily activities. To further the artist analogy, it is not the artist's job to develop the tools for painting (e.g., manufacturing the paintbrush). Information about manufacturing paintbrushes is assumed to be easily accessible as well as inherently available, not to mention unnecessary for the actual process of painting. Although an artist does need to worry at some point about choosing the appropriate paint brush, during the process of painting, the painter need only think about how the paint is applied to the canvas. Similarly, an engineer should be able to work with objects in a virtual space without thinking about detailed development of the analysis and modeling tools, even

المنارة للاستشارات
www.manaraa.com

though at some level in the engineering process that information is important. S/he should be able, for instance, to grasp a virtual part in a pump and alter it and only have to think about the consequences of such a move to the rest of the system in which the pump resides. Much like the artist, engineers must also be able to move across scales within a system and understand how the parts within the system will interact with each other without being concerned with the underlying tools (e.g., process simulation, CFD, FEA, CAD) being utilized to create the virtual systems.

One area that focuses on many of the same aspects of virtual engineering is called Think, Play, Do. A description of its components follows:

- Think – Innovation Technology (IvT) (e.g., modeling, simulation, virtual reality) liberates creative people from mundane tasks, enabling them to experiment more freely and widely, producing a variety of options
- Play – IvT enables people to design, prototype, and test more cheaply and effectively and to delay choices about investment until market and technology patterns become clear
- Do – The extent of digital integration with other kinds of technology gives innovators confidence in their ability to successfully transform new ideas and designs into products and services

Taken from "Think, Play, Do" Doddgson et al. 2005, p. 4-5

Think, Play, Do is based on the idea that the tools needed in today's business environment demand access to a broad range of data from many project stakeholders. This level of access is necessary to create a decision-making environment. Doddgson et al. also note the demand that engineers place on numerical models at all levels of product detail. Design

المنارة للاستشارات
www.manaraa.com

engineers and project stakeholders demand complete virtual access to product acceptance models all the way to product maintenance. This access is needed in trying to improve product reliability and reduce product cost. Often, design engineers have a wealth of experiential information that enables them to see patterns and places for improvement without having to see a physical prototype. In this work environment, the more information that is available virtually and accessible through familiar product representations, the more the design engineers can improve the overall product without a physical prototype. These observations will be utilized as requirements in the development of the advanced engineering software framework described in this research.

3.1 Advanced Engineering Software Frameworks

As discussed in Chapter 2, current software frameworks succeed in solving many different engineering problems and questions in regards to meeting today's product development and delivery schedules. These frameworks and algorithms enable engineers to more efficiently answer questions, make decisions about specific problems, and address specific areas within engineering disciplines. Over the past few decades, a significant amount of work has been completed on the construction of software frameworks to solve engineering problems and on interfaces to connect disparate software packages. These tools have played important roles in creating environments where engineers are better enabled to solve problems and create solutions that would have otherwise remain hidden. These frameworks have three main limitations:

- They are monolithic; that is, they provide limited capability within their own interfaces and modules. They are not extensible to new applications.

المنارة للاستشارات

www.manaraa.com

- They can only integrate a limited number of models, based on strong typed data interfaces requiring extensive conversion of external data formats.
- They only provide limited data access by external software tools and limited interfaces for external execution control.

These limitations must be overcome by creating a new framework based on the new way of handling engineering models and information. Specifically, the computer needs to handle the middleware tasks of information integration, extension of existing models to new applications, and detail development. Today, these middleware tasks are handled by the engineer.

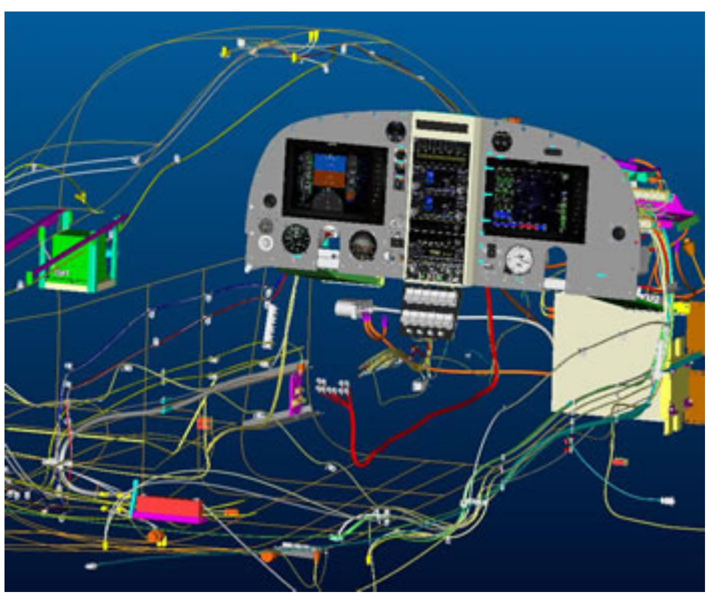
To enable computers to perform this middleware task, a software framework must support:

- An object-oriented approach to information management
- Incorporation of emergent behavior methods
- A bottom-up information semantic dataflow

This requires the creation of a "wiring layer" that provides the interface by which other software platforms can coexist to share information, which becomes important in the development of engineering informatics tools to address the limitations described above. Just like in the physical sense with airplane avionics boxes, the "wiring layer" enables a diverse set of software tools to connect to each other and transfer information without user input (Figure 5). A popular discussion topic is:

المنارة للاستشارات

www.manaraa.com



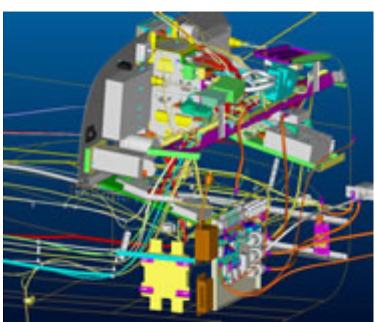


Figure 5. Avionics wiring layer in a plane [Evektor 2008]

المنارة للاستشارات
www.manaraa.com

> … the role of metadata and semantic technologies to help integrate the various information sources with each other and with applications. I think that one of the key reasons that large commercial applications are so inflexible and difficult to modify is that the data access of the applications is "wired in" - connected to a specific data base. While the applications and databases have been separated since the advent of relational databases more than twenty years ago, modifying the application to access a different data source requires serious application changes and testing.
>
> The answer to this dilemma is to provide a sophisticated layer of metadata between the applications and information sources to act essentially as a shared integration or "wiring" layer. Moreover, the richer you make the semantic model embedded in the metadata layer, the more this shared integration layer becomes a kind of "common understanding" among all the various components being integrated, which makes the overall system more adaptable and dynamic. That is, different integration decisions will be made in real time, depending on the overall environment. [Wladawsky-Berger 2006]

The "wiring layer" is possible for engineering tools as long as the software connected to the wiring layer adheres to an agreed-upon communication protocol. Each software framework is able to use any data structures for communication but must have an interface that is able to communicate with the wiring layer. Within a software framework, this wiring layer becomes the exposed software interface that enables the metadata for domain-specific software tools to be shared. The interface for this wiring layer is important to the development of solutions for engineering informatics, but frameworks that can

المنارة للاستشارات
www.manaraa.com

exploit this interface become increasingly powerful in the tools and experiences the engineer is given access to.

In this research, an advanced engineering framework will be developed that utilizes engineering objects and VE-Suite to create a wiring layer for engineering information. The key components of an advanced engineering framework are:

- Transparent interfaces
- Object-oriented characteristics (i.e. modularity, hierarchy, and abstraction)
- Enabling emergent behavior

One of the key components of this development will be implementing an object-oriented approach to information management to enable the investigation and utilization of the subsequent engineering objects that are created. This implementation will result in engineering objects that will enable the engineer to focus on engineering rather than on information integration. The engineering objects, when implemented with each component in a product being developed, will create environments where virtualized systems and parts can be analyzed and produced with fewer costs being devoted to the design and development phase of the realization process.

3.2 Objects

The main difference between the objects described in this work and those that have been defined and implemented previously is that the objects described here provide a mechanism for relationships with other objects through multi-scale numerical relationships that describe physical phenomena that are not possible in other object-oriented approaches or engineering frameworks. The importance of this difference will emerge over the next few chapters.

المنارة للاستشارات
www.manaraa.com

One of the key characteristics required by engineering objects is the ability to encapsulate the information for a specific component in a simulation. This encapsulation provides the framework for moving a decision about a particular object forward. The encapsulation in an engineering object enables a user to drill down into the object, determining what information is needed and what can be discarded. This is different from many presently used engineering processes, in which this information is often hidden or disconnected and the user must dig for each piece of information across different software packages, resulting in time being spent on non-problem-solving tasks.

These objects carry with them context and meaning and the ability to be modified by the user. The context and meaning that they carry is the meta-data they contain and the information about any sub-objects that they contain. These characteristics build on the functionality of programmatic object-oriented principles in that virtual objects are modular, easily reused, extensible, polymorphic, able to support complex objects (i.e., objects can make up other objects), and can be loosely or tightly coupled to other objects. One key difference is the ability to change representations of itself at run time through the manipulation of the information that the object contains. Most of all, an engineering object must have the ability to self-discover and adapt to other objects that may need to exchange information with that particular instance of the object. The information that is exchanged with other objects must be able to be managed internal to an engineering object without outside assistance from the user.

Engineering objects will help manage complexity because they manage information in an object-oriented method in that information is grouped based on its physical counterpart. This design is different from other engineering frameworks where information

المنارة للاستشارات
www.manaraa.com

for one component may be stored in disparate software packages, requiring the user to gather the information. Within engineering objects, even if information is stored within disparate software packages, the user interface into the object is through a single engineering object interface. In addition, the user can decide at what level of immersion he or she wishes to interact with the engineering object.

Part of the inherent nature of engineering objects is that they can be comprised of other objects, much like physical objects can be comprised of multiple sub-objects. Foucault notes that objects in nature are described as follows: "Each visibly distinct part of a plant or an animal is thus describable in so far as four series of values are applicable to it. These four values affecting, and determining, any given element or organ are what botanists term structure." [Foucault 1994, p. 134] The structure that is derived from the description of objects enables humans to understand complex systems. The structure, as Foucault notes, enables us "to describe certain fairly complex forms on the basis of their very visible resemblance to the human body, which serves as a sort of reservoir for models of visibility, and acts as a spontaneous link between what one can see and what one can say" (p. 135). While engineering objects may not be used to describe the human body, the human body can be used as a parallel system to demonstrate how engineering objects are constructed and illustrate what is necessary for software to enable users to communicate and understand complex systems such as the body.

The interfaces to engineering objects are constructed to enable the structure of the information that the engineering object contains to not be degraded when passing through the interface. Foucault notes, "By limiting and filtering the visible, structure enables it to be transcribed into language. It permits the visibility of the animal or plant to pass over in

المنارة للاستشارات
www.manaraa.com

its entirety into the discourse that receives it" [Foucault 1994, p. 135]. If the structure of an object is degraded beyond what the user is asking for, the description of the object necessary for discourse is unavailable. The comprehensive structure of an engineering object must be available if necessary to enable understanding to be gained from the object. In object-oriented programming languages, fixed interfaces (i.e., functions) are created to access an object's specific variables, but in the case of engineering objects, the interfaces will be constructed to be flexible to adapt to the information describing the physical entity so that the structure of the information is not degraded.

3.3 Object Interactions

To create connections between objects, tools must be utilized that enable objects to self-describe themselves to the world and to understand information presented to them. Information interactions include human-to-human, human-to-object, human-to-model, model-to-model, and model-to-object. Some types of information interaction (e.g., human-to-human) have been well defined in the literature, providing a foundation on which to base engineering informatics. Literature about some interaction types has been available for as long as 40 years [Foucault 1994]. These interactions have a significant place in engineering in that they help provide not only a basis for how engineers should interact with information, but also indicate what information must be automatically made available to enable appropriate interactions to occur without direct user interaction. When creating a virtual component, the user should not need to consider the solver or solvers that are employed, but should be able to construct the part as if in real life.

When objects are constructed and connected into a network that enables the end author to interact with and explore various options for connectivity and interrelationships,

المنارة للاستشارات
www.manaraa.com

the resulting network resembles the web. These particular networks are called scale-free networks [Barabasi 2003]. A typical characteristic of these networks is that there are a few major hubs or master objects that have sub-objects and information sources feeding into the master objects. With an understanding of the resulting network created by multiple objects, characterizing classes of engineering objects becomes possible.

The classes of objects used in engineering range from humans within the design process to sensors that feed information in one direction. Classes of objects are then grouped into five basic subcategories that are binned by object characteristics based on an object's interaction or lack of interaction with its surroundings. Each bin holds a group/class of information that will enable other models to detect how to interpret and use the information provided by the other objects. These bins (Figure 6) are classified as follows:

- Class 1 – One-way information objects
- Class 2 – Two-way information objects
- Class 3 – Two-way interactive objects
- Class 4 – Instructive objects
- Class 5 – Knowledgeable objects

المنارة للاستشارات
www.manaraa.com

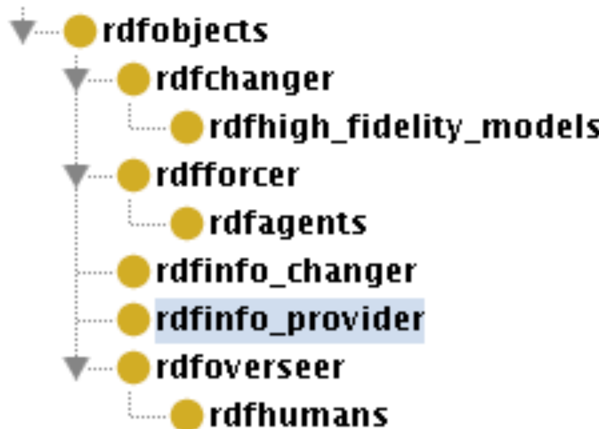


Figure 6. Class of engineering objects

www.manaraa.com

The bins also dictate what information will be published about each class of objects so that other objects can understand where and how they fit into the engineering process. These bins are related by two main factors. The first factor is related to how an object interior to an environment can affect the environment. These objects can be broken into three sub-classes of objects: models that provide input, models that can provide input but also can receive some input from the user, and two-way interactive models. The second class of objects is agent-type objects, which can be broken into two subclasses: cleanup agents, or dumb agents that are told what to do; and super agents, which rank close to or the same as humans. We assume that humans are the top knowledge form in the hierarchy of these objects.

With these categories in place, the grouping and handling of information can be automated because assumptions can be made about how each object interacts with the world, the type of information it contains, and the manner in which the object can manipulate the world and the information that is provided to it. These classes enable the objects, as described by Foucault, to adapt as the underlying objects change. However, the core interfaces do not need to change.

The objects in the research described here are being developed to provide a mechanism that enables relationships with other objects through numerical relationships of physical phenomena that are not possible in other object-oriented approaches. As stated earlier, this research defines an object as encompassing all sorts of physical and digital artifacts that can convey meaning in interpersonal communication, providing the ability to construct virtual systems, and possessing these characteristics:

المنارة للاستشارات
www.manaraa.com

- defining its own status, method of operation, and method of interaction with other objects
- sensing and acting on the environment in which it is situated over time
- responding directly to the environment

Engineering problems are often defined by a series of constraints that are dictated by the environment, management, marketing, or a whole host of intents and expectations. These constraints imply the level of information fidelity required within the engineering process and are often either lost or overemphasized. Each domain has a set of rules (e.g., gravity) that dictate these constraints as well as what may or may not exist in products within that domain. The rules also define the characteristics of the world in which the product will be developed.

Object integration in the engineering environment will occur through the exchange of information at similar fidelities, enabling objects to interact with each other and humans to interact with objects. The level of interaction needed to move a specific decision forward drives the level of fidelity required for the engineering object. Objects' characteristics are determined primarily by the decisions that must be made.

The objects can be classified based on the information they contain and the raw sources for this information, which will dictate the capabilities the object has in the VE-Suite environment. These information sources can range from sensors, radio frequency identification (RFID) tags, high-fidelity numerical models, spreadsheets, and many other sources. Each of these is capable at some level of interacting with its environment. Each entity may only provide one-way information, but some may be two-way coupled to understand their surroundings and act independently of the user investigating the product.

المنارة للاستشارات
www.manaraa.com

3.3.1 Emergent Behavior

In the real world, many phenomena occur without external intervention (e.g., ants building an ant hill, flocks of birds, termite hill construction, the growth of a coral reef, traffic patterns, the stock market). These events occur through the use of communication through the environment in which the entity resides. For example, in traffic, cues that a driver receives from signs and other cars' signals influence how he or she drives. These signals and signs provide the driver with information about what to expect and how to operate their car.

For this discussion, self-organizing will be defined as a process that an open system returns to an organized state spontaneously after surroundings change. [Bak 1996]. Open systems in this case refer to the fact that the software tools can accept input from external programs and users. Characteristics of self-organizing objects are:

- ability to tell what they need to connect to
- ability to tell what type of information they can accept
- ability to tell where they need to run from within the hierarchy of information available to the object

Self-describing is defined here as the ability for a virtual object to provide information about itself through its own interfaces, revealing the representations that allow the user to understand the object in every context as in the physical world, as in nature when ants use the environment to communicate indirectly with each other, enabling the colony to accomplish a task as a whole. Similarly, constructing large systems of engineering objects without user intervention requires that many of the tasks regarding identification about the engineering object and its capabilities must be handled without

المنارة للاستشارات
www.manaraa.com

external intervention. Characteristics of a self-describing object that can be derived from this definition are:

- ability to tell other objects about its internal characteristics
- ability to define input/output variables that are accessible for a given request
- ability to define fidelity and other vendor and meta-data, which comes primarily from the ontology

Self-operation is an object's ability to know what model to run to provide the appropriate information to a requesting object and when to run that model. Self-operating also implies the ability to connect the appropriate models and fidelities of models given the question being asked of the object. If a lower-fidelity model is run, the higher-fidelity model may not have to be run because a change to a lower-fidelity model may not have an impact on higher-fidelity models. Conversely, if a higher-fidelity model is run, the lower-fidelity models will likely have to be rerun. Self-operation enables self-organization and self-description.

Characteristics of a self-operating object include:

- ability to optimize itself
- ability to inverse engineer itself
- ability to tell the virtual environment what needs to be run

3.3.3 Object-Oriented Principles

One of the areas of weakness in current engineering software frameworks is the inability to generically construct interfaces to adequately enable the structure and representation of an engineering object to be shared with the rest of the software framework. In this research, representation is the data structure for a particular aspect of an

المنارة للاستشارات
www.manaraa.com

engineering object. Specifically, representation is a formalization of point of view or perspective. For example, in a graphical perspective, the representation of an engineering object will primarily be its CAD data. The CAD data has a specific data structure, is different from the numerical results, and also has a different graphical representation from the CAD data. As in object-oriented programming, methods must be available to interact with the underlying information. These methods, in a practical manner, are functions. From a higher level, these functions are variation operators. The variation operators in engineering objects are used to drive exploration of a problem space. These operators are the exposed tools that will enable users to modify CAD data and numerical simulation parameters, enable optimization algorithms to automatically search the problem design space, and change the underlying inputs of a particular object.

Much like our brains hierarchically represent our experiences [George et al. 2004, George et al. 2005, Hawkins et al. 2006, Hawkins 2004], engineers should also create a hierarchy of information. The brain operates on information farther away from the sensor (the neuron in this case) (Figure 7), enabling it to accomplish incredibly complex tasks as information is broken up into manageable pieces. In addition, the brain uses invariant representations to store information about the world, permitting it to store an incredible amount of information in a very small space. In the case of the brain, invariant means that the information that the brain stores, whether from seeing, hearing, or touch, is stored in the same format to enable different sections of the brain to operate on the same information. More significantly, the brain can perform tasks using general information because it remembers patterns rather than explicit information. These patterns dictate how we interact with the world and permit us to apply patterns to a broad range of problems.

المنارة للاستشارات
www.manaraa.com

This storage mechanism permits the same portions of the brain to share the load of problem solving independent of the problem domain. For example, if the eyes need help solving a problem, the portion of the brain that handles information from the hands can be used and vice-versa. By the time information from the eyes and hands reaches their respective portions of the brain, the information has been translated to an invariant format and has been relegated to the portion of the brain trained to handle the information.

Creating software frameworks that have the ability to exchange information from diverse problem domains with the same level of abstraction as the brain requires the use of ontologies and other tools created for the semantic web (i.e., XML and XSL). The ontologies created from these implications are very general and highly pattern-oriented, not detail-oriented. The engineer needs to work at a high level of abstraction with the information much like is done with CAD packages today. The engineer provides dimensions and key geometric features but does not generate any of the curve equations for the computer. While the objects discussed in this research will contain the ability to perform specific tasks much like our brains are able to perform tasks on specific sets of information, the ability to share information across modules within an object will be possible in much the same way that programming languages enable data share.

المنارة للاستشارات
www.manaraa.com

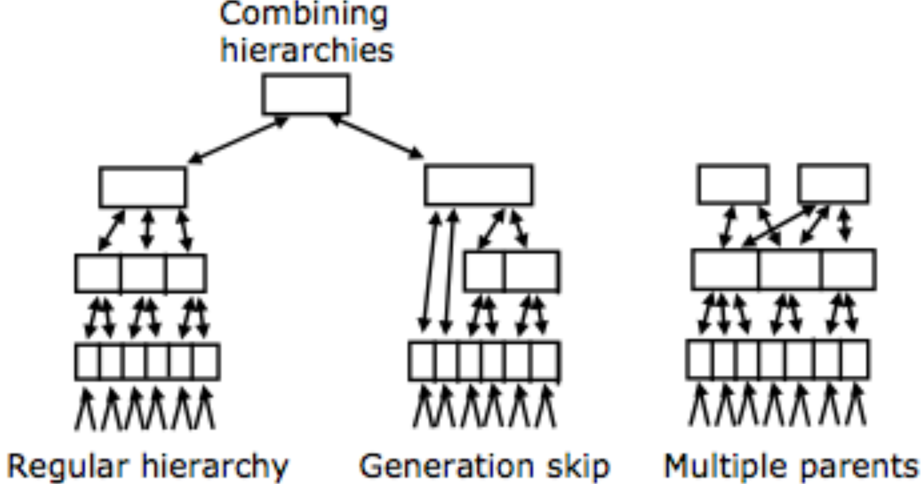
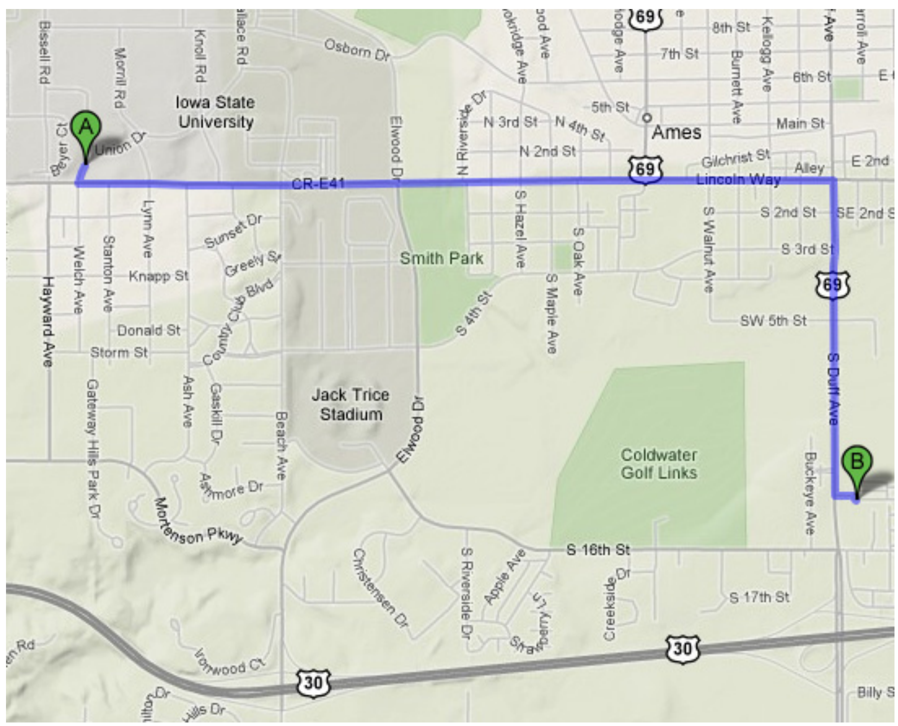


Figure 7. Hierarchical representation of the brain [Hawkins et al. 2006, p. 6]

www.manaraa.com



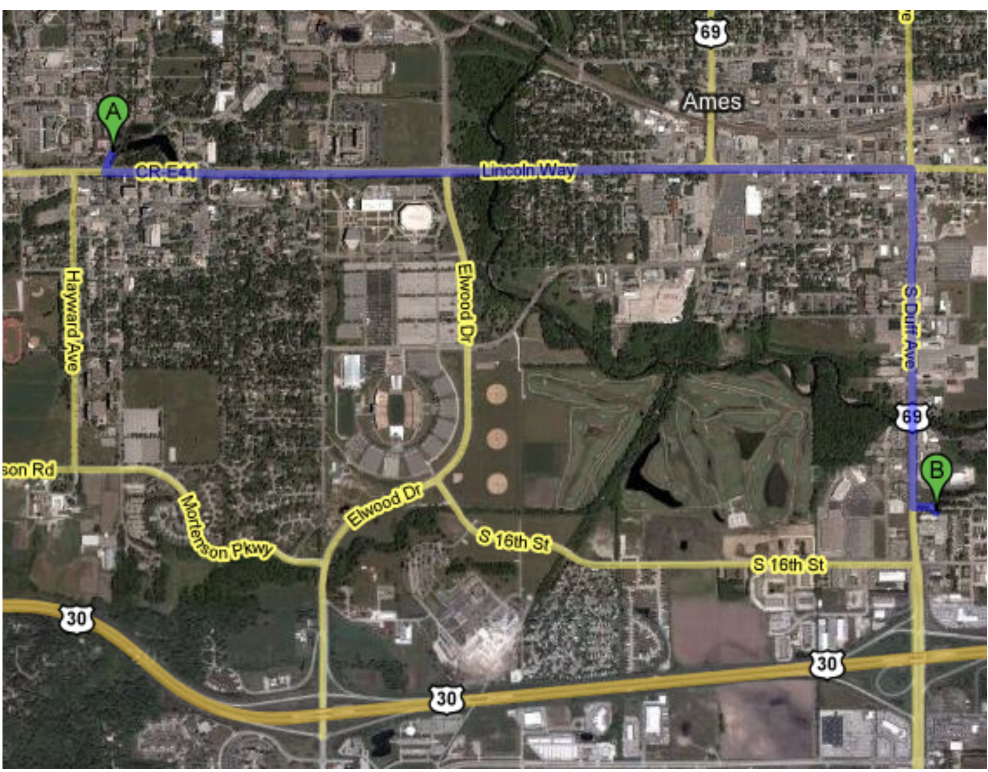


Figure 8. Map directions from Iowa State University to Hickory Park Restaurant, Ames, IA [Google 2008]

المنارة للاستشارات

www.manaraa.com

The end result of these implementations will be software that enables the engineer to see the engineering domain much like online maps provide (Figure 8) driving routes: basic maps show what direction to drive and which turns to take. Maps are available that have various layers showing elevation changes (e.g., relief maps), previous roads, previous building locations, zoning information, or any other geographical information about the city. These maps can show changes that were not visible with basic, non-layered maps. In the same way, engineering software must enable engineers to see whatever layer of information they desire at any fidelity. For example, if warranty information is being viewed for a product and a specific component is frequently breaking, the underlying physics models for this component must be accessible with a simple action from the user. The engineer in this case must be able to drill down from warranty information (e.g., the number of times this component broke), to the CAD representation, to the FEA analysis that was performed by the original design team to better determine the problem behind the warranty recalls.

3.5 Summary

Once implemented, engineering objects as described here will enable the user to more easily traverse from a simplified information state to a complex information state, which is necessary to gain a true understanding of the information [Davis 1999]. The environment created by objects provides a mechanism for engineers, artists, and individuals of many backgrounds to enter a mode of discourse that enables participants to interact with other participants and to understand what they are trying to communicate. In addition, these objects typically contain artifacts that enable participants to recall events or meaningful points of interest surrounding the objects. This is the same level of realism that

المنارة للاستشارات
www.manaraa.com

must be present for the engineering objects being described and implemented in this research. In order for these engineering objects to be utilized, a software framework must be implemented that enables objects to communicate without intervention or direction by the user, just like objects interact in the physical world.

www.manaraa.com

Chapter 4: Implementation of the proposed advanced engineering framework

Creating an advanced engineering framework based on engineering objects requires the following three tasks to be implemented:

- Transparent interfaces—a transparent interface results in data independent methods being exposed to the user to enable data from any domain to be passed through the interface. The goal of the interfaces developed here is to avoid strong typed methods that are attached to a specific problem domain.
- Implementation of object-oriented principles—to enable virtualized systems to be created that avoid the problems that Booch and Ross et al. outlined, the methods that enable the objects to be created for this engineering framework will include modularity, hierarchy, abstraction, and design patterns to be utilized with engineering objects. These qualities will be exhibited in the engineering objects constructed here and will be supported by the engineering framework. Through the use of transparent interfaces, modularity, hierarchy, abstraction, and design patterns can be implicit in terms of the capability that the framework can support.
- Emergent behavior—the engineering framework will enable emergent behavior in two ways. First, the structure of the information that is received by the computational units and by the core engines will provide key reference data so that UIs can be constructed, three-dimensional graphical representations can be

المنارة للاستشارات
www.manaraa.com

constructed, and computational units can gain information about what is upstream or downstream of them without user intervention. Second, any computational unit will be able to query the rest of the virtual environment for data if the respective unit requires other inputs to perform its tasks. This querying capability also occurs without user input and enables the computational unit to exhibit some autonomous behavior.

The core components of VE-Suite require several changes to support these tasks. These needs will be met through the extension of the current VE-Open CORBA interface, implementation of an XML Schema and respective API, and extension of VE-Xplorer to support the display of engineering objects in a virtual world. Other changes will be made to VE-Conductor and VE-CE. All of the changes outlined in this chapter are a result of the research performed for this thesis. The implementation of each component was shared with other organizations such as NETL, REI, and other graduate students in the Simulation, Modeling, and Decision Sciences Program.

4.1 Transparent Interfaces

To enable information to be accessible to the core VE-Suite engines and the engineering objects contained within the virtual world, transparent interfaces are needed that are independent of the problem domain to which the interfaces are being applied. These interfaces must enable data from any domain to be accessible throughout the engineering framework and allow the full fidelity of the data to be accessible wherever the user requests it. Rather than pushing data to the user, a query-based model will be used for these interfaces. A query-based transparent interface puts all of the control of the information in the hands of the user, the computational units, and the plugins in the

المنارة للاستشارات
www.manaraa.com

engineering framework. A query-based system is how we interact with the objects around us. To find out how much an object weighs, we must pick it up; we cannot tell by simply looking at it. Implementing this query-based model requires changing the CORBA IDL interface (VE-Open) for back-end computational units to support a query-style interface, enabling a command-driven unit interface engine that receives commands through a user-constructed query interface based on user requests. The unit parses the command, compares it to a set of available commands that are supported in the unit, and carries out the required tasks. Each of these steps is completed without user intervention, resulting in autonomous and emergent behavior by the computational units. In addition, only the information requested by the user will be transferred, resulting in several smaller data structures being transmitted and reducing the network burden.

4.1.1 Implementation of Transparent Interfaces

To implement transparent interfaces, VE-Conductor will be updated to run in two editing modes: offline and online. In the offline mode, the user is responsible for more of the manipulation of the VE-Suite software engines. In the online mode, the VE-Suite engineering framework manages much of the background work for querying and changing input parameters. The only programmatic difference in VE-Conductor between the online and offline modes is the point at which the SetNetwork and SetParam calls are made.

In the online mode, the user connects to the computational engine once VE-Suite is started. This tightly couples the VE-Conductor, VE-CE, and the computational unit. In the online mode, the user can query the VE-CE and the computational unit to bring the embedded network in a computational unit to VE-Conductor. When a VE-Conductor plugin on the design canvas is double-clicked, VE-Conductor queries the module for

المنارة للاستشارات
www.manaraa.com

parameters and specific parameter properties. When a new module is included on the design canvas, a subsequent SetID call is made immediately through the VE-CE to the computational unit to make a new instance of the object in the computational unit. This enhancement is an important step in the construction of the engineering framework. This feature is a key component in being able to scale the engineering framework to support hundreds of sources of information by supporting modularity. By enabling a single computational instance to manage multiple instances of an object in the virtual world, a smaller memory and management load is put on the engineering framework. In this implementation, only the inputs and results are stored for each instance of the object in the computational unit. Also, when a module is removed from the VE-Conductor design canvas, a CORBA call immediately removes the respective instance from the computational unit. The computational unit is still available if the engineer decides the object is necessary in the virtual world. When the engineer decides that inputs need to be changed for a specific object, the SetParam CORBA call is made to set the input parameters on that unit. Again, only the object that is being modified by the engineer is affected. Modularity in this case does not require that all the object's inputs be set again, just the object's inputs that are being requested by the engineer.

To implement this new functionality within VE-Conductor, the following functions are modified in the current VE-Open IDL:

string Query(in string commands)

This query method takes the command's parameters requested by the engineer. VE-CE passes this call directly to the respective computational unit

المنارة للاستشارات

www.manaraa.com

and responds directly to VE-Conductor, through VE-CE with the response from the computational unit (Figure 9).

void SetNetwork(in string network)

This function's action depends on VE-Conductor's mode (i.e., online or offline). If VE-Conductor is operating in the offline mode, the network string contains the whole network's information, including all the modules' input parameters. The computational engine, however, does not store these parameters; it only parses the network portion of the DOMDocument to extract the module list and link information. This is done to enable scaling within VE-Suite to support ultra-large systems. By only requiring the network information to be stored in the VE-CE, the memory footprint of VE-CE remains small even with ultra-large systems. VE-CE then calls the individual module's SetID and SetParams to pass on the respective part of the network string for the specific computational unit to parse and store the information.

المنارة للاستشارات
www.manaraa.com

```xml
<?xml version="1.0" encoding="UTF-16" standalone="no" ?>
<commands name="Commands" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="veshader.xsd">

  <vecommand commandName="getInputModuleParamList">
    <parameter dataName="ModuleName" id="932658b0-40ff-df48-8732-a7a423958ff2">
      <dataValue type="xs:string">Data.Blocks.CVAP</dataValue>
    </parameter>
  </vecommand>

</commands>
```

Figure 9. VE-Open query command from VE-Conductor

www.manaraa.com

In addition to the functions modified above, these three new calls in the VE-Open IDL support the VE-Conductor online editing mode;

void SetID(in string moduleName, in long id)

void DeleteModuleInstance(in string moduleName, in long module_id)

void SetParams(in string moduleName, in long module_id, in string param)

In the online mode, SetNetwork only passes the top-level system information needed to describe the virtual world. Each module's inputs are passed separately through VE-CE into the unit using the SetParams call. SetID and DeleteModuleInstance calls are used for the computational unit to manage its instance list when the user adds or deletes multiple instances of the same module on the VE-Conductor design canvas.

4.1.2 Summary

Transparent data interfaces are the first component in the process of enabling object-oriented methods and emergent behavior in the engineering framework. The interfaces described above enable the engineering framework to be domain-independent through the use of string data types that are shared throughout VE-Suite. While the string data types have a processing overhead cost, this is weighed against the domain independence that is gained by using strings. In this research, the processing overhead was not found to be an inhibiting factor when working with these interfaces. The main performance lag is due to a serial threading model for CORBA ORBs in each of the core VE-Suite engines. In the future, this will be changed to a different threading model to improve the overall performance of the VE-Suite engineering framework.

المنارة للاستشارات
www.manaraa.com

4.2 Object-Oriented Principles

Three of the main tools created for the Semantic Web are used to create a contextualized engineering environment. These tools include XML (the primary tool used by VE-Suite) and XML Schema [W3C 2007], XSL [W3C 2008], and OWL [Herman 2007]. Integrating these tools into an application that drives a virtual environment changes the environment from being purely picture-based to being information-rich, with many avenues for the engineer to explore. The XML schema implemented in VE-Suite provides the primary mechanism for data transfer within the VE-Suite framework. XSLT [W3C 1999] is used to process the XML documents generated by VE-Suite to create web pages that are W3C compliant, enabling future software clients using VE-Open to easily access information pertaining to a component by querying a web page, rather than querying multiple sources for a complete description of the component. The information available through such a portal will include high-fidelity information such as CFD data and text-based information describing virtual components (e.g., a pump or a turbine). These software clients will implement libraries that are capable of interpreting the XML data being streamed so that engineers can easily interact with the information, rather than having to build custom code for every engineering problem examined.

4.2.1 Modularity, Hierarchy, and Abstraction

As discussed in Chapter 3, the engineering objects described in this research must satisfy many requirements, including the ability to handle multiple representations and the ability to handle data stored in a format that enables broad use of the information among many objects. The source of this information must come through an invariant representation, such as an XML schema. The invariant representation in this case is a

المنارة للاستشارات

www.manaraa.com

formal definition that does not change and is indifferent to the problem domain it is applied. A schema provides the foundation for creating the necessary data structures so that the virtual objects discussed here can exchange information, present queries, and understand responses to the rest of the environment, as well as interact with agents that may explore the environment or interact with the objects. The objects must have the capability to store any information in data structures that adhere to the schema defined here to enable modular and abstract objects to be constructed. This schema is much like the invariant matrix used in proper orthogonal decomposition [Kirby 2001, Meer 1998]. In this case, the input is the data from the objects. The solution, after having passed through the invariant representation, is the DOMDocument (Appendix B). This transform takes place through the VE-Open libraries. These documents represent a finite number of snapshots from the source invariant representation and provide the basis on which the objects are constructed and manipulated. The VE-Open schema developed here is a broad schema developed to handle a diverse set of problem domains. This schema must also address the three main representations that the proposed objects require to provide the engineer with the full context of the physical object: graphical, numerical, and the user control. These representations (i.e., graphical, numerical, and the user control) provide the user with a complete set of interfaces with which to interact with the virtual object The power of this schema, as well as the challenge surrounding it, is that it does not limit the user in the development of objects. Rather, new objects can be introduced in a natural manner by supporting hierarchical objects (e.g., objects constructed of other objects). As noted earlier, the schema remains constant across all problems while each DOMDocument is the specific representation for a particular problem under investigation. This implementation feature is

www.manaraa.com

important as it enables VE-Suite to be constructed around the same transparent data interfaces no matter what problem domain is being investigated.

4.2.2 Ontologies

Ontologies are used to provide the mechanism by which sources of information can be classified as well as show the connection, hierarchy, and pedigree of information sources. The classification enables VE-Suite plugins and computational units to understand the full context of information that is received from queries. For example, when a computational unit queries an upstream or downstream component, it does not know any contextual information about the data it is receiving. The computational unit does not know the order accuracy of the solver, the convergence criteria of the solver, or the methods used to generate the information from the neighboring computational unit. This information is necessary to provide error approximation on the information being presented and to provide other uncertainty merits to the user. The ontology results in formal definition so that each of the VE-Suite plugins can provide contextual information to the rest of the virtual world. An initial ontology implementation within VE-Suite follows:

```
<rdfs:Class rdf:about="&rdf_;objects"
            rdfs:comment="most generic term for an entity in the ves world"
            rdfs:label="objects">
           <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>
```

This element becomes the basis for other types of objects within the VE-Suite domain. Creating a subclass from which to derive other objects enables the software interpreting these streams to easily derive structure from the implied nature of the XML syntax. This structural information would not have been as easily accessible with other languages and

www.manaraa.com

other markup implementations. An example of objects that extends the base object class follows:

```
<rdfs:Class rdf:about="&rdf_;info_provider"
           rdfs:comment="one way information out"
           rdfs:label="info_provider">
          <rdfs:subClassOf rdf:resource="&rdf_;objects"/>
</rdfs:Class>
```

This element describes an object such as a sensor.

```
<rdfs:Class rdf:about="&rdf_;overseer"
           rdfs:label="overseer">
          <rdfs:comment>can affect change on any portion of the world as well as
                                   investigate any other object in the world</rdfs:comment>
          <rdfs:subClassOf rdf:resource="&rdf_;objects"/>
</rdfs:Class>
```

In addition to the implied structure and relation to other objects that the ontology provides, embedding contextual notes into each respective object through an rdfs:comment is relatively easy. This element describes objects such as software agents that may work on the engineer's behalf.

```
<rdfs:Class rdf:about="&rdf_;humans"
           rdfs:label="humans">
          <rdfs:comment>humans are completely able to change and observe large
                                   scale environments</rdfs:comment>
          <rdfs:subClassOf rdf:resource="&rdf_;overseer"/>
</rdfs:Class>
```

The human object would describe an engineer and may provide information about what position they hold within an organization to determine what security privileges should be granted to the user or how to display information. This user information can also be used to configure a virtual environment based on stored preferences about particular classes of individuals.

Each of these elements provides an initial framework by which information can be classified within VE-Suite's virtual engineering environment. These elements are a broad

المنارة للاستشارات
www.manaraa.com

description that must be distilled in such a manner that the software can transfer information. To enable this, an XML schema has been created.

4.2.3 XML Schema

XML schemas provide the basic structure by which information can be transferred within the VE-Suite engineering framework. While the ontology provides the broad framework that computers use to classify information sources without human input, the schema provides the means by which the data can be packaged to hold the information provided by a particular source. For example, the ontology defines an object that can be a human or an information provider. These objects, when broken down into an XML document, would be composed of veDataValuePairs and other veXMLObjects described below. An example of such a document will be illustrated below, but first the basic XML elements that compose the description of an object will be described.

The schema is composed of a few key XML element types. The first type is the veXMLObject element:

```
<xs:complexType name="veXMLObject">
    <xs:attribute name="objectType" type="xs:string" use="optional" />
    <xs:attribute name="id" type="xs:ID" use="optional" />
</xs:complexType>
```

This element type is the basis for all other elements within the VE-Open schema, enabling any other element type within the schema to be embedded or referenced in a generalized manner. This enables abstraction, hierarchy, and modularity to be embedded in the schema and is the enabling factor for these qualities to be present in the objects that the XML schema describes. Although a formality, this element type enables the logic to be complete when embedding and referencing derived veXMLObjects in other element types. The

www.manaraa.com

functionality that veXMLObject enables will be illustrated below in veCommand. The veCommand is the element type that is passed in the Query functions described earlier.

The second element type is the veDataValuePair:

```
<xs:complexType name="veDataValuePair">
  <xs:complexContent>
    <xs:extension base="veXMLObject">
      <xs:sequence>
        <xs:element name="dataName" type="xs:string" maxOccurs="1" minOccurs="1" />
        <xs:choice maxOccurs="1" minOccurs="1">
          <xs:element name="dataValue" type="xs:anyType" />
          <xs:element name="genericObject" type="veXMLObject" />
        </xs:choice>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The veDataValuePair type holds a descriptive name about the data it contains as well as a veXMLObject or raw xs:anyType. This flexibility enables veDataValuePair to be a generic container element that holds any form of data being processed by a particular object. Note that a veDataValuePair is a complete extension of a veXMLObject. This extension permits a veDataValuePair to be embedded within another veDataValuePair.

The third element type is veCommand:

```
<xs:complexType name="vecommand">
  <xs:complexContent>
    <xs:extension base="veXMLObject">
      <xs:sequence>
        <xs:element name="command" type="xs:string" />
        <xs:element name="parameter" type="veDataValuePair" minOccurs="0" maxOccurs="unbounded"
/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

This element type contains a descriptive name for the command in addition to an xs:sequence of veDataValuePairs. The command is constructed to enable any object to

www.manaraa.com

request or send a series of veDataValuePairs with information about the potential application of the data contained within. Because a veDataValuePair can contain any veXMLObject that is derived for the VE-Open XML schema, a veCommand can be used as the overall container to transmit information about objects and the attributes used to describe them. This information is transferred in the Query methods and the SetNetwork functions.

The previous three elements described (i.e., veXMLObject, veDataValuePair, veCommand) are the core building blocks of the VE-Open XML schema. Each of the following elements described will use the key elements in the construction of the descriptors for an object. veParameterBlock is a general component that contains information about general information sources within VE-Suite:

```
<xs:complexType name="veParameterBlock">
    <xs:complexContent>
      <xs:extension base="veXMLObject">
        <xs:sequence>
          <xs:element name="blockID" type="xs:unsignedInt" maxOccurs="1" minOccurs="1" />
          <xs:element name="blockName" type="xs:string" />
          <xs:element name="transform" type="veTransform" minOccurs="0" maxOccurs="1" />
          <xs:element name="properties" type="veDataValuePair" minOccurs="0" maxOccurs="unbounded"
/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

An example of a parameter block would be a reference to a VTK dataset. The property element is configured to maintain a list of custom elements for describing a particular information source. This list of elements may also contain a list of hardware specifications for a sensor array or for a CFD solver configuration.

CADNode describes the geometrical representations that are stored for a particular object.

www.manaraa.com

```
<xs:complexType name="CADNode">
  <xs:complexContent>
    <xs:extension base="veXMLObject">
      <xs:sequence>
        <xs:element name="parent" type="CADAssembly" maxOccurs="1" minOccurs="0" />
        <xs:element name="transform" type="veTransform" minOccurs="1" maxOccurs="1" />
        <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1" default="Assembly" />
        <xs:element name="type" type="xs:string" />
        <xs:element name="attribute" type="CADAttribute" maxOccurs="unbounded" minOccurs="0" />
        <xs:element name="activeAttributeName" type="xs:string" />
        <xs:element name="animation" type="CADNodeAnimation" />
      </xs:sequence>
      <xs:attribute name="visiblility" type="xs:boolean" />
      <xs:attribute name="physics" type="xs:boolean" />
      <xs:attribute name="opacity" type="xs:double" use="optional" default="1.0" />
      <xs:attribute name="makeTransparentOnVis" type="xs:boolean" default="true" />
    </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

The CADNode contains two unique features. First, the CADNode does not maintain its own geometrical information, but references a file that contains this information. Second, the element can contain information about how to apply high-fidelity lighting capabilities. These are stored in the attribute element. This element contains a CADAttribute, which maintains a GLSL program embedded in the CADAttribute.

The following veXMLObjects will be described to provide context for the XSLT example that follows. These elements are used to construct the connectivity between virtual objects that are modeled in a system. The first element examined is a vePoint:

```
<xs:complexType name="vePoint">
  <xs:complexContent>
    <xs:extension base="veXMLObject">
      <xs:attribute name="xLocation" type="xs:unsignedInt" use="required"/>
      <xs:attribute name="yLocation" type="xs:unsignedInt" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A vePoint is primarily used by the software within VE-Suite that renders graphical representations of the network schematic for the system under review. vePoint is composed of two unsigned integers representing the X and Y locations of the point. Data types for a

www.manaraa.com

point are unsigned integers so that graphical widgets libraries can easily render the point location. Graphical widgets libraries typically work in whole numbers rather than decimal values. The second element utilizes vePoint and is a veLink:

```
<xs:complexType name="veLink">
  <xs:complexContent>
    <xs:extension base="veXMLObject">
      <xs:sequence>
        <xs:element name="fromModule" type="veDataValuePair"/>
        <xs:element name="toModule" type="veDataValuePair"/>
        <xs:element name="fromPort" type="xs:unsignedInt"/>
        <xs:element name="toPort" type="xs:unsignedInt"/>
        <xs:element maxOccurs="unbounded" minOccurs="2" name="linkPoints" type="vePoint"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute use="optional" type="xs:string" name="type"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

A veLink is composed of the necessary components to link one system component to another. The descriptors for the two modules that the link couples are fully described in addition to the necessary information to draw the link. This choice was made so that, upon obtaining the link, the software would not only be able to describe the information in the link, but would also be able to draw it.

The third element for a network description in VE-Suite is the veNetwork:

```
<xs:complexType name="veNetwork">
  <xs:complexContent>
    <xs:extension base="veXMLObject">
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="link" type="veLink"/>
        <xs:element maxOccurs="6" minOccurs="6" name="conductorState" type="veDataValuePair"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="tag" type="veTag"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

It should be noted that the veNetwork element is relatively simple, but builds on the previous two elements for full description. A series of links composes veNetwork and

www.manaraa.com

provides information about how the network should be rendered by VE-Suite's rendering software. veNetwork is essentially a graph composed of edges (e.g., veLinks) and vertices (e.g., veModels). The representation of veNetwork follows closely on that defined by the DOT [Graphviz 2008(b)] language utilized by GraphViz [Graphviz 2008]. While the DOT language is not utilized internally by VE-Suite, this task remains as future work to leverage the DOT language in addition to the use of the Boost Graph Language [Seik et al. 2001]. These tools enable VE-Suite to use graph decomposition algorithms and detection algorithms to determine disconnected and feedback sections of graphs.

As noted previously, the veModel represents the nodes on the graph. The veModel builds on all of the previous elements and has the main responsibility for containing the inputs, outputs, CAD, and raw stream data for a particular model representation. The veModel is the data container for an object (Appendix B). In reference to the classification of objects, the veModel contains the raw data that would tell other objects about itself. In addition to containing the object's raw representational data, the veModel can also contain a veSystem, which will be described later. The purpose of this embedded element is to provide the user with the ability to:

- Create a hierarchical assembly of complex objects
- Embed a third-party solver into a broader simulation

This capability provides one of the main components that enable the core VE-Suite software framework to support a broad range of problem domains.

```
<xs:complexType name="veModel">
  <xs:complexContent>
    <xs:extension base="veXMLObject">
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="ports" type="vePort"/>
```

المنارة للاستشارات
www.manaraa.com

```
        <xs:element maxOccurs="1" minOccurs="1" name="iconLocation" type="vePoint"/>
        <xs:element maxOccurs="1" minOccurs="0" name="icon" type="xs:string"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="results" type="vecommand"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="inputs" type="vecommand"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="informationPackets"
         type="veParameterBlock"/>
        <xs:element name="geometry" type="CADNode"/>
        <xs:element maxOccurs="1" minOccurs="0" name="modelAttributes" type="vecommand"/>
        <xs:element maxOccurs="1" minOccurs="0" name="modelSubSytem" type="veSystem"/>
      </xs:sequence>
      <xs:attribute name="vendorID" type="xs:string" use="required"/>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The key components in the veModel element are the veParameterBlock, CADNode, vecommand, and veSystem elements. These elements provide the necessary information for each core software engine in VE-Suite to produce the proper representation for the object. For example:

- If an object does not have CAD data, then nothing is rendered for the object.
- If the object does not have outputs, then other objects will not be able to gather data from it.

The attribute element within the veModel contains the classification data for other objects to determine how to handle data from a particular object. Currently, the classification data is limited and further implementation is left for future research.

The veSystem element is the overall element that links the disparate veModel and veNetwork elements. It is also the main element that is saved when writing out a ves file (i.e., the DOMDocument storing all of the objects) from VE-Suite. In addition to establishing a relationship between veNetwork and veModel, it also enables systems to be embedded within models. This element provides the capability to construct complex engineering objects within VE-Suite.

www.manaraa.com

```
<xs:complexType name="veSystem">
   <xs:complexContent>
     <xs:extension base="veXMLObject">
       <xs:sequence>
         <xs:element type="veModel" maxOccurs="unbounded" minOccurs="1" name="model">
</xs:element>
         <xs:element type="veNetwork" minOccurs="1" maxOccurs="1" name="network"> </xs:element>
       </xs:sequence>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>
```

The veSystem element also provides the foundation to link multiple third-party solvers together. For example, when integrating an Aspen Plus flowsheet with another solver, the Aspen Plus solver and the other solver each looks like a single system to the VE-CE. Within each of the systems may reside complex subsystems (Figure 10), which are handled by their respective solvers. Any subsystem corresponds to a single computational unit connected to the VE-CE, which does not mean that subsystems cannot be broken in terms of information transfer across subsystem boundaries.

4.3 Emergent Behavior

With the transparent interfaces and object-oriented principles within the VE-Open XML Schema and IDL, the core VE-Suite software engines can be changed to utilize this capability. The changes implemented enable the software engines to manage more of the middleware tasks that were previously handled by humans. Some of these tasks are: querying model inputs, providing modeling results, executing model simulations, performing post-processing tasks such as meshing, and transforming data for post-processing or model import. These tools enable the software engines to facilitate emergent behavior in the computational units and graphical environment.

المنارة للاستشارات
www.manaraa.com

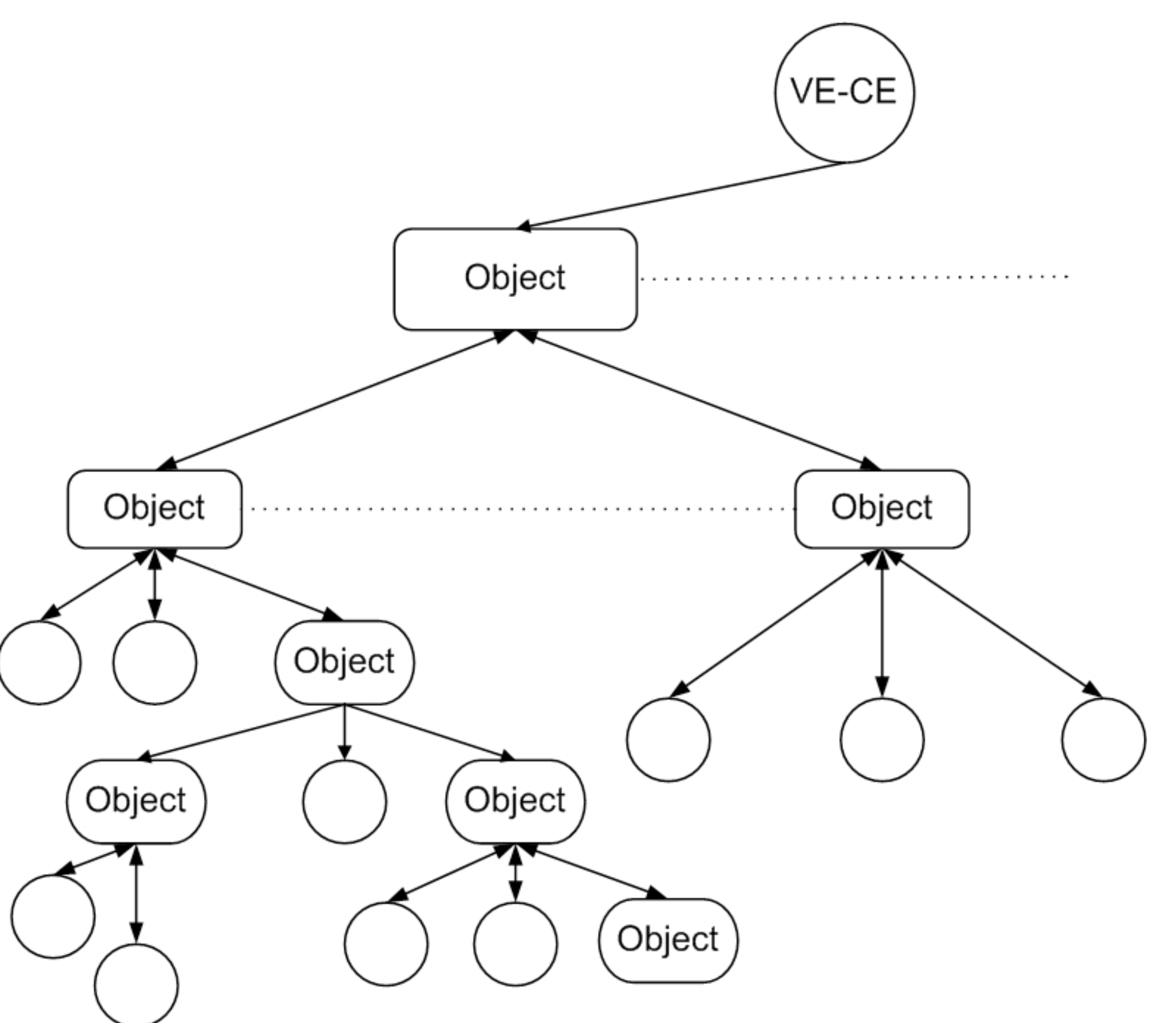


Figure 10. VE-Open subsystem example

المنارة للاستشارات
www.manaraa.com

With a system and network schematic fully realized in XML, it is possible to distribute this schematic information to many other platforms including web browsers, which will be illustrated with XSLT. XSLT is a broad-based general tool that can be used to transform XML data into a multitude of different formats, such as a web page. Below is a small snippet of the XSLT script that takes the VE-Open XML document and transforms the network diagram, which is described by a veNetwork, into a webpage that allows the user to see model-specific information from any location in the world. The model-specific data in this case is a series of veDataValuePairs that are populated with information that describes a particular engineering system under investigation.

```
<xsl:template match="linkPoints">

<xsl:variable name="x1">
        <xsl:value-of select="xLocation"/>
</xsl:variable>

<xsl:variable name="x2">
    <xsl:choose>
                <xsl:when test="boolean(following-sibling::linkPoints/xLocation)">
                        <xsl:value-of select="following-sibling::linkPoints/xLocation"/>
                </xsl:when>
                <xsl:when test="not(following-sibling::linkPoints/xLocation)">
                        <xsl:value-of select="xLocation"/>
                </xsl:when>
    </xsl:choose>
</xsl:variable>

…

<xsl:variable name="xPos">
    <xsl:choose>
     <xsl:when test="$xValue = 0">1</xsl:when>
     <xsl:when test="$xValue &lt; 0"><xsl:value-of select="$xValue * -1"/></xsl:when>
     <xsl:when test="not(($xValue = 0)and($xValue &lt; 0))">
                        <xsl:value-of select="$xValue"/>
                </xsl:when>
    </xsl:choose>
```

www.manaraa.com

```
</xsl:variable>

<xsl:variable name="yPos">
      <xsl:choose>
        <xsl:when test="$yValue = 0">1</xsl:when>
        <xsl:when test="$yValue &lt; 0"><xsl:value-of select="$yValue * -1"/></xsl:when>
        <xsl:when test="not(($yValue = 0)and($yValue &lt; 0))">
                              <xsl:value-of select="$yValue"/>
                    </xsl:when>
      </xsl:choose>
</xsl:variable>
```

Note that the script above traverses into the veNetwork element to find the raw vePoint data needed to render the network schematic. This script provides an avenue to present high-fidelity information that enables the user to interact with a complex system's data rather than with a multitude of different tools to gather the necessary information about a specific component. In other, more complex portions of the script, XSLT is used to traverse into the veNetwork element to provide basic information about the system's components. The Semantic Web tools implemented here enable VE-Suite to leverage current technology to provide unique capability in the engineering framework without creating new tools to disseminate and display information. If the transparent interfaces and object-oriented principles were not implemented with industry standard tools (i.e., Semantic Web tools), new tools would have to be created to parse and interrogate the data within VE-Suite.

www.manaraa.com

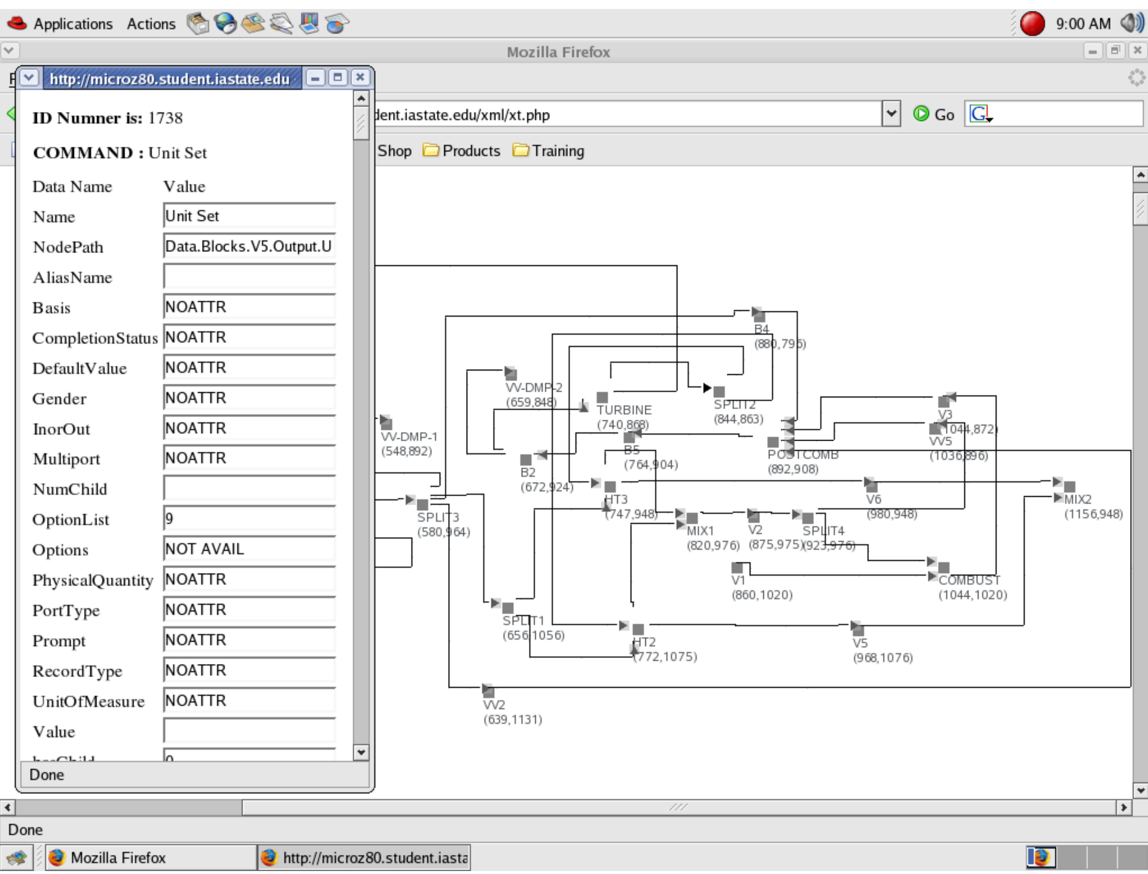


Figure 11. VE-Conductor input UI

www.manaraa.com

4.3.1 VE-Conductor

The changes in VE-Conductor enable real-time information retrieval and queries from the computational units connected to VE-CE as noted in Section 4.1. Because of these changes, the user can query a unit for subsystem information from a third-party embedded network solver. The user can query for input and result parameters from any computational unit attached to VE-CE. The results and input data are provided in a browser-like user interface (Figure 11) to handle display and editing for query-enabled units. A developer can override this base functionality with a specific plugin to handle the respective query-enabled unit. This capability will be illustrated later in this document. As noted previously, the unit-specific data is all accessed in real time by the user. This enables the user to edit and interact with the system under investigation in the three-dimensional environment created through VE-Suite while simultaneously interacting with a computational unit to make low-level changes to the flowsheet. This workflow is possible through the implementation of the query interfaces in VE-Suite.

4.3.2 VE-CE

The changes to VE-CE have turned it into a data proxy that is responsible for scheduling the execution of various units and the transfer of information and queries between units. This enables VE-CE to be run on a low-powered gateway computer, even when the network data is large and must be passed through the VE-CE interfaces. This design is beneficial because it enables the computational units and VE-Conductor to be run anywhere on the Internet and to interact transparently through a firewall. In addition, it enables VE-CE to promote emergent behavior within the computational units by proxying the data without encumbering the user with those requests. When operating with a process

المنارة للاستشارات
www.manaraa.com

simulator as one of the units in the VE-Suite framework, VE-CE passes commands from the user through to the respective unit. The unit is then responsible for sending the information on to the respective software package.

As revised, VE-CE will not store unit input parameters as it did before; rather, VE-CE only parses the top-level system. Subsystem elements are assumed to be subsystems that will be managed by their respective units. This design enables VE-CE to scale as the subnetworks within a simulation expand. However, there is still not a direct link between VE-Conductor and the computational unit. VE-CE is the proxy for all calls.

4.3.3 Computational Unit

The changes to the computational unit support a command-driven software interface through the implementation of unit wrappers to accept an XML-formatted command through the query interface:

string Query(in string commands)

The computational unit parses the XML command sent from the VE-CE and extracts the command element to determine what is needed by the engineer. For each predefined command, a command handler is implemented to perform the specified action. Following is a list of current predefined commands supported by computational units. This list will expand as needed in the future.

- "getNetwork" retrieves the flowsheet information from a third-party solver so VE-Suite can draw the network (Figure 12) and enable the user to query individual unit operations for results information
- "getModuleParamList" returns the list of available parameters for a given unit operation

www.manaraa.com

- Once the user has chosen a specific parameter, the properties for that variable are queried via the “getParamProperties” command and displayed to the engineer

These commands and methods for accessing data within computational units have shown to scale from flowsheets with anywhere from 10 to 200+ unit operations (Figure 12). The following is a list of the detailed commands described above.

Command: getNetwork

Parameter: none

Return String: the XML network, including module name, identification, and interconnection links.

Note: This command is only applicable to a unit that actually embeds a network in itself.

Sample Command XML:

```
<Command>
        <vecommand>
        <command> getNetwork </command>
        </vecommand>
</Command>
```

Command: getModuleParamList

Parameter: moduleName

Return String: a list of parameter names for that module

Sample Command XML:

```
<Command>
        <vecommand>
        <command> getModuleParamList </command>
        <parameter>
                <dataName>moduleName</dataName>
```

www.manaraa.com

```
                <dataValueString>Gasifier</dataValueString>
        </parameter>
    </vecommand>
</Command>
```

Command: getParamProperties

Parameter: moduleName

Parameter: moduleId

Parameter: paramName

Return String: a list of the names and values of the parameter's properties

Note: It is possible to have multiple instance of the same unit in a single flowsheet network. Those modules are identified with module IDs. With IDs, the handler would know which instance's properties to query.

Sample Command XML:

```
<Command>
    <vecommand>
    <command> getParamProperties </command>
    <parameter>
                <dataName>moduleName</dataName>
                <dataValueString>Gasifier</dataValueString>
        </parameter>
    <parameter>
                <dataName>moduleId</dataName>
                <dataValueInt>102</dataValueInt>
        </parameter>

    <parameter>
                <dataName>paramName</dataName>
                <dataValueString>Temperature</dataValueString>
```

www.manaraa.com

```
            </parameter>
        </vecommand>
    </Command>
```

Because a single unit's data (e.g., the Aspen Unit) can be large and VE-CE should be lightweight, the state information such as input variables' values are held in the unit itself. Each unit needs to maintain a list of its instances along with its parameters and values and add or remove instances as needed. So one new IDL would be added:

```
void DeleteModuleInstance(in long module_id)
```

Calling this function will delete the instance along with the data structure that has the ID that is passed into the function.

The SetParams function needs to be modified so the unit knows which instance the input parameters belong to:

```
void SetParams(in long module_id, in string param)
```

The SetId function would have new actions in addition to setting the unit's ID. Because each ID would identify a certain instance of a certain module, the action would include searching the list of existing instances and allocating memory for the instance's parameters if it is not already on the list. A new SetCurID will be introduced to make a certain instance active as the current instance:

```
void SetID(in long id)

void SetCurID(in long id)
```

The GetId function would also change to return this unit's list of module IDs. GetCurId would be added to return the current running instance's ID. Because multiple instance data would now be stored in the unit and the Calc function can run on only one instance at a

www.manaraa.com

time, the GetCurId would return the ID of the instance that holds the set of parameters that the current or next StartCalc function would operate on.

ArrayLong GetID()

Long GetCurID()

An abstraction layer between the raw CORBA interface and code and the user's code will be added to the units. The new abstract layer will handle some of the query work and implement basic default functionality so that the user only has to override needed functions. Essentially, this is similar to the utility classes for the other core engines that enable the VE-Open code to be hidden. This abstraction layer will be described in the examples following this chapter.

Like the inputs, the result and stream was previously stored in VE-CE. In this new design, the results will also be stored in the unit itself. A GetResult Call will be added to the unit so VE-CE can gather results data. Similarly, the stream result will also be saved in the unit itself. In addition to the GetResult call, a GetStream call will also be added. The downstream unit will call GetStream(import) on VE-CE. Because VE-CE will have the network information, it will know which upstream module connected on which port. It will subsequently call that module's GetStream(modId, PortId) and return the result.

4.3.4 VE-Xplorer

The final set of changes required within the core components of VE-Suite are related to VE-Xplorer. Generic and schematic networks [Huang et al. 1993] are commonly utilized within today's engineering environments to enable engineers to understand connectivity in systems and to provide ways to show complex networks. These networks

www.manaraa.com

serve a specific need within the engineering process in the development of a product from birth to death (Figure 13).

As shown in Figure 13, the generic network provides a place for the engineer to begin thinking about the problem. This network purely illustrates global components and basic relationships and is not intended at this point to provide high-fidelity information to the user. As the design process moves forward, this network morphs into the schematic network, which provides more detailed information but, at this point, still does not necessarily provide geometrical or production- or manufacturing-level information. This is where tools such as Aspen Plus can improve the engineering process. Aspen Plus, for example, can add chemical processing information such as mass flow rates, operating temperatures, and other stream information associated with a chemical processing plant to the plant network diagrams. These development tools are critical in interacting with large systems no matter what domain or discipline they address. In VE-Suite, the generic network can be constructed within VE-Conductor. There are multiple ways to look at the network under investigation in VE-Suite, including:

- A two-dimensional schematic in Conductor (Figure 12)
- A three-dimensional schematic in Xplorer (Figure 14)
- A three-dimensional geometric view in VE-Xplorer (Figure 15)

المنارة للاستشارات

www.manaraa.com

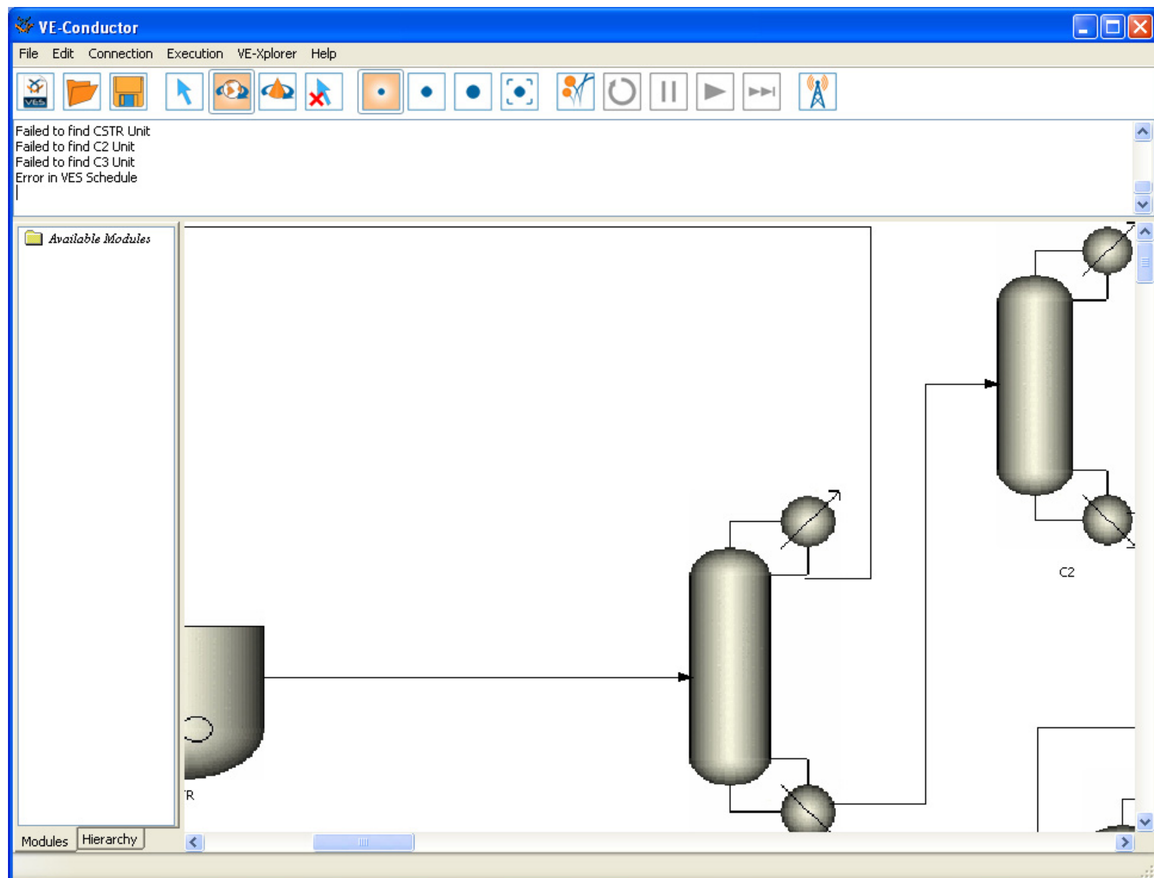


Figure 12. VE-Conductor network diagram

www.manaraa.com

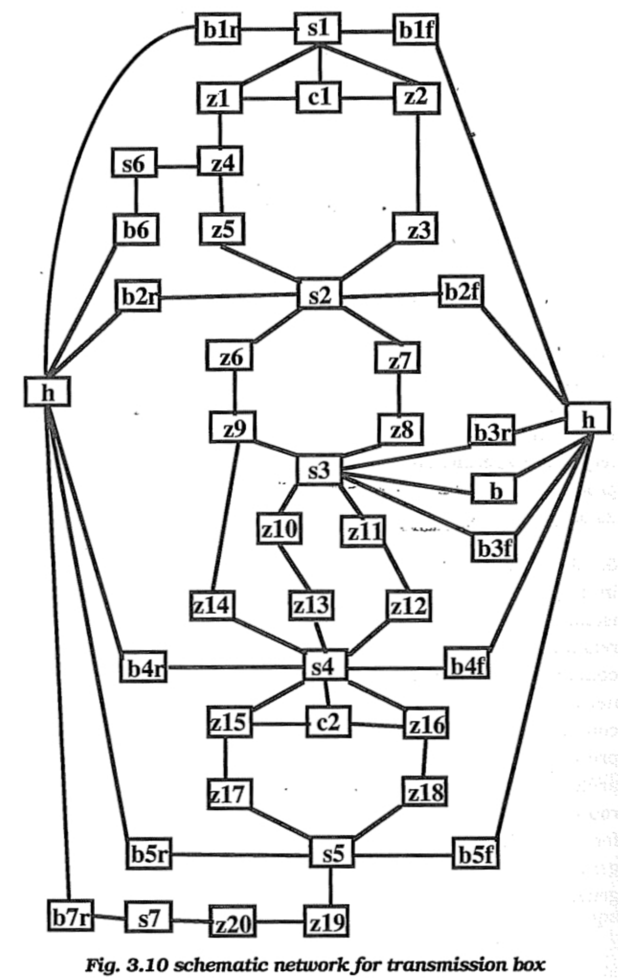


Figure 13a. Network schematic examples [Huang et al. 1993, p. 64]

المنارة للاستشارات
www.manaraa.com

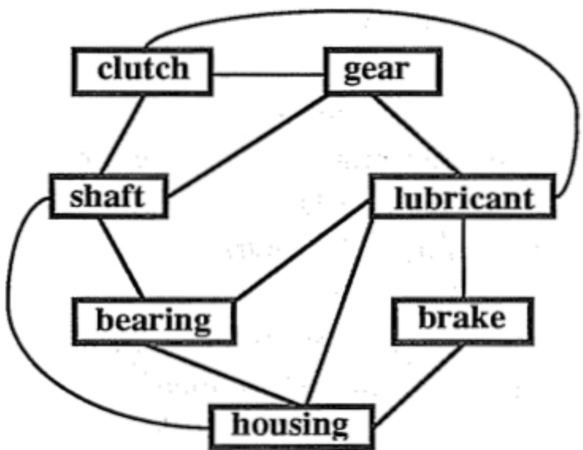


Fig. 3.11 generic network for transmission boxes

Figure 13b. Network schematic examples [Huang et al. 1993, p. 65]

المنارة للاستشارات
www.manaraa.com

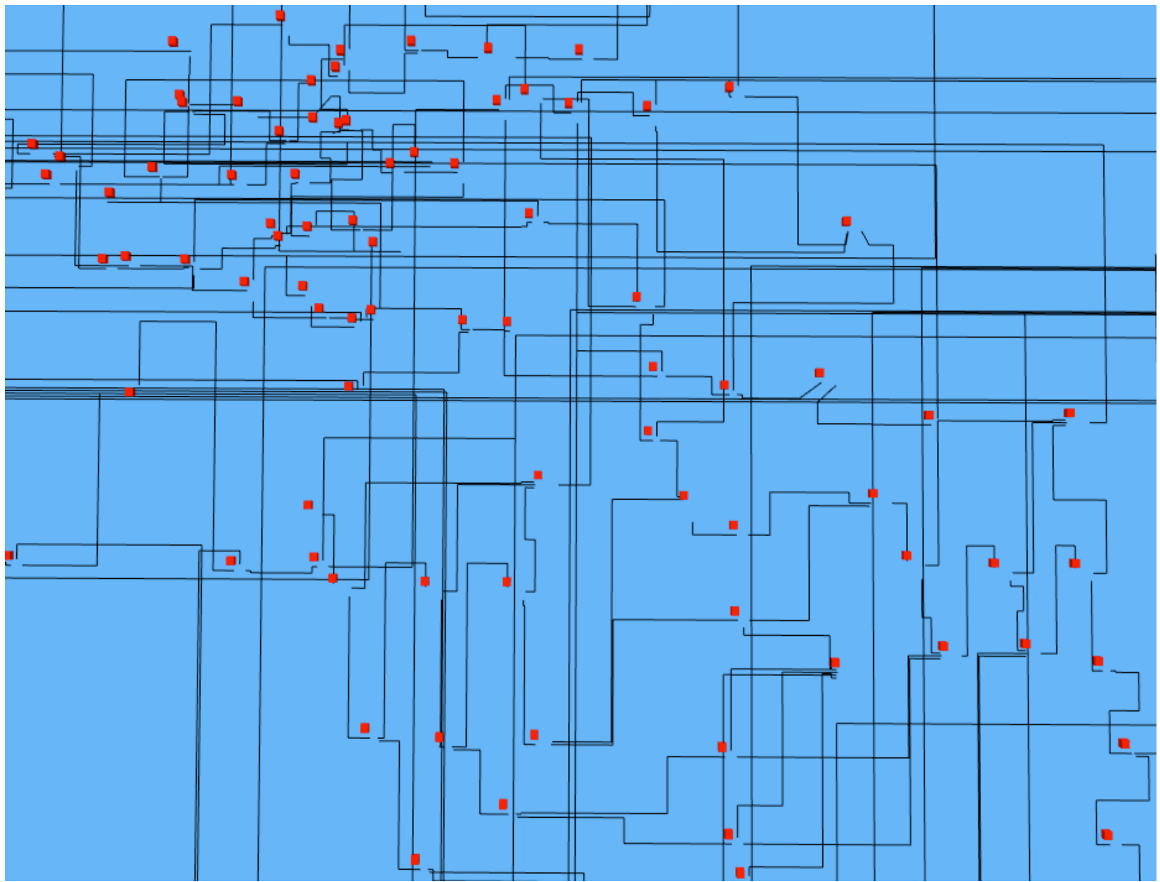


Figure 14. VE-Xplorer schematic view

www.manaraa.com

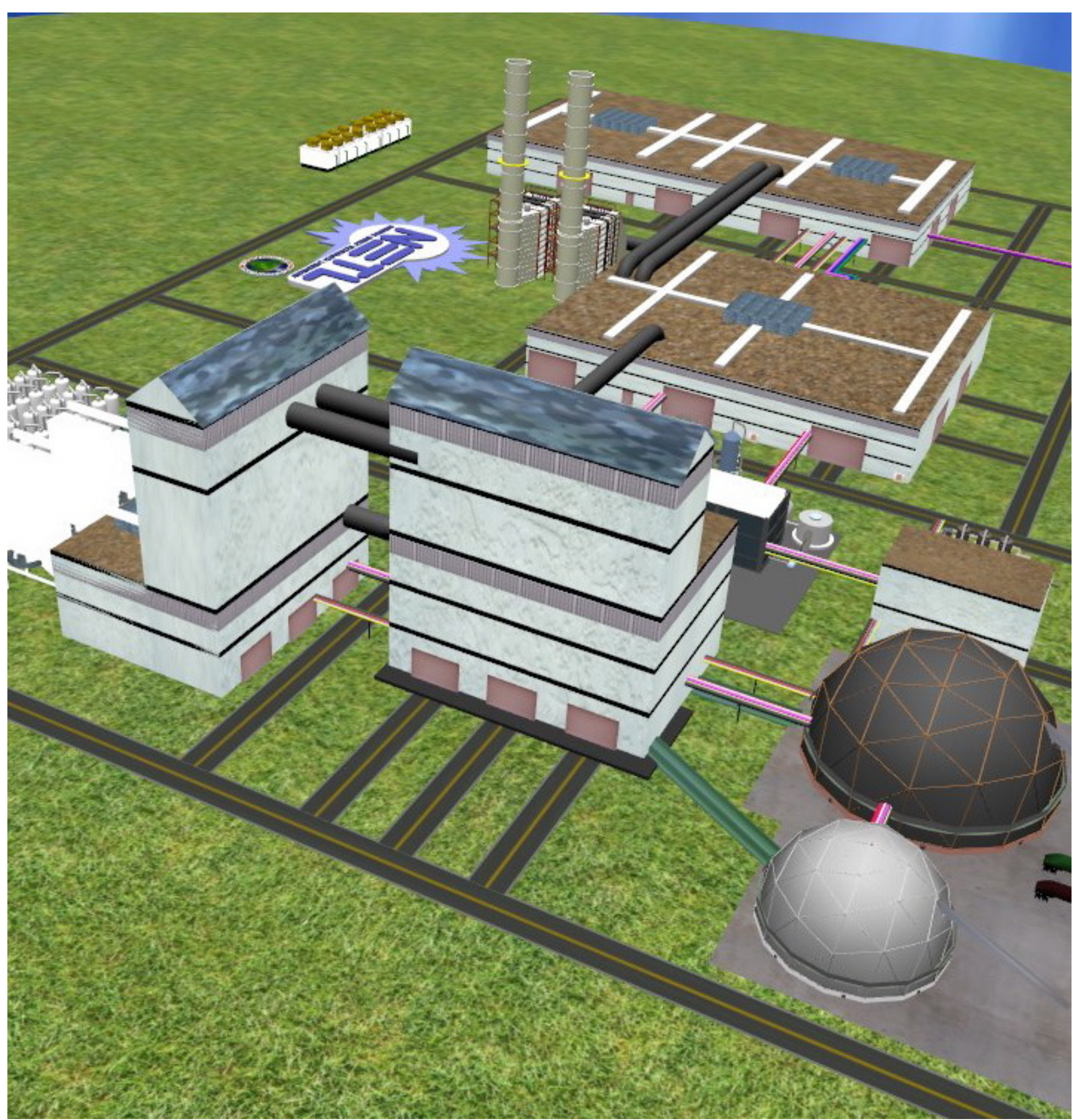


Figure 15. VE-Xplorer geometric view

المنارة للاستشارات
www.manaraa.com

In each of these views, the user should be able to move objects and select menu items. Lines are used to show objects as being connected and as having some relationship, but the type of relationship is not shown unless the user wants it to be. If the user wants to specify the information flow, arrowheads appear on the line ends. This information is then used by the computational engine to determine where and when data is needed. Again, the network shows the basic relationship of components to each other. Aspen Plus queries this network itself. This network can then be used in VE-Conductor to add further information such as CAD models, three-dimensional graphics representation, or other information the user wishes to store with the network. Additionally, if there are more external information sources for the network under investigation, the information can be added to a particular node of the network. This is possible through each engine of VE-Suite, utilizing VE-Open in terms of its internal data format.

The functionality added is a two-dimensional network diagram rendered with three-dimensional objects, enabling users to view the two-dimensional network displayed in VE-Conductor within the three-dimensional VE-Xplorer environment. This addition is the first step toward being able to interact with the network within the same user environment. For example, the user will be able to right-click on the three-dimensional objects and bring up menus within the three-dimensional environment, whether on the desktop or in a three-dimensional virtual reality device. Again, this functionality will be added in the future and will implement the ability to parse the XML representation of a network:


```
<network>
        <link id="8c176b27-8cf8-1541-a7c4-9752ab8b666e" name="1" type="0">
          <fromModule dataName="B2" id="8176f0ec-88b6-0246-ba4c-038a98f27c3f">
            <dataValue type="xs:integer">454</dataValue>
          </fromModule>
          <toModule dataName="B1" id="f3ebd3d5-477c-3542-a9fa-45626e576f64">
```

www.manaraa.com

```
      <dataValue type="xs:integer">252</dataValue>
    </toModule>
    <fromPort type="xs:integer">1</fromPort>
    <toPort type="xs:integer">0</toPort>
    <linkPoints xLocation="41" yLocation="90"/>
  </link>
  <link id="da3a13c7-75de-f848-854a-09805fbef47e" name="2" type="0">
    <fromModule dataName="B1" id="e2504a80-37e8-8647-ab93-abd6162826a9">
      <dataValue type="xs:integer">252</dataValue>
    </fromModule>
    <toModule dataName="B3" id="094d6ec9-9f8e-f549-8ff6-c8b17c72a50a">
      <dataValue type="xs:integer">527</dataValue>
    </toModule>
    <fromPort type="xs:integer">3</fromPort>
    <toPort type="xs:integer">2</toPort>
    <linkPoints xLocation="351" yLocation="47"/>
  </link>
  <conductorState dataName="m_xUserScale" id="88650542-6f71-6d44-8358-1b66684a4112a">
    <dataValue type="xs:double">1</dataValue>
  </conductorState>
  <conductorState dataName="m_yUserScale" id="4a5c290f-76f8-cb48-8e6c-455dd3ab86b3">
    <dataValue type="xs:double">1</dataValue>
  </conductorState>
  <conductorState dataName="nPixX" id="8c94fa3d-42d6-3544-b04b-6da0e70304a4">
    <dataValue type="xs:integer">20</dataValue>
  </conductorState>
  <conductorState dataName="nPixY" id="6c3c55f8-422a-d743-a278-b8fd0f6c46f8">
    <dataValue type="xs:integer">20</dataValue>
  </conductorState>
  <conductorState dataName="nUnitX" id="b12778d3-58a9-ac46-a0b0-a4cc81f0aa8d">
    <dataValue type="xs:integer">200</dataValue>
  </conductorState>
  <conductorState dataName="nUnitY" id="3142e447-7125-c142-8b6d-1fa4dd79d7e0">
    <dataValue type="xs:integer">200</dataValue>
  </conductorState>
</network>
```


and render the three-dimensional graphics equivalent (Figure 14). With this capability in place, the user will have the ability to use the default box three-dimensional icon to render in the network or to render custom CAD representations. This representation can be placed with the proper directory location for the application being completed by the user. For example, the user can create a custom three-dimensional icon of a gasifier and have it be rendered in place of the default box. The benefit of this functionality to the end user is the ability to move from a two-dimensional schematic to a comprehensive three-dimensional

www.manaraa.com

physical representative model with a step between the two extremes to provide the user with a conceptual layout, enabling him or her to make the jump from the network view to the three-dimensional model view. In addition, coupling between components enables the user to understand the connectivity between subsystems in a large system analysis such as a power plant (Figure 16).

Integrating CAD tools is also a key research effort that will be undertaken in the near future. Integrating these types of tools (e.g., OpenCASCADE [Open CASCADE 2008]) will allow engineers to change details in the component's current graphical representation and then send the new geometrical data to the respective numerical model and see the updated results in the visual environment. This roundtrip design process will allow the design loop to be closed and permit the engineer to focus on system design instead of transferring data from one engineering package to another.

A feature is currently being developed that will allow VE-Suite to interact with initial graphics exchange system (IGES) files and render the associated geometry. This will allow VE-Suite to address a number of current engineering operations, including:

- Easy computer-aided design (CAD) loading capabilities
- Interactive CAD changes
- Interactive analysis with finite element analysis (FEA), computational fluid dynamics (CFD), and any other numerical tools requiring grid generation

المنارة للاستشارات

www.manaraa.com

Figure 16. Investigating a virtual power plant

المنارة للاستشارات
www.manaraa.com

First, many CAD converter tools use IGES as an intermediate format for translation. For CAD software packages, IGES is one of the most well-supported export/import methods. For example, Pro/E uses IGES as its main export format because of its capability to store raw NURBS data in file. Other software tools, such as PolyTrans, suggest translating IGES files from Pro/E rather than using the raw Pro/E files, due to the unchanging nature of IGES files and the universal support of IGES. In VE-Suite, IGES files can now be imported via a library called OCC. Once the files are imported, the NURBS data must be extracted and rendered. With the IGES data in OCC, the NURBS data is easily extracted and passed to the rendering library. The rendering library, which is referred to as VE-NURBS, is contained within VE-Xplorer in VE-Suite. VE-NURBS currently only supports OpenSceneGraph for rendering but can be extended to other scene graphs such as OpenSG or raw OpenGL. The VE-NURBS library currently only supports B-spline types of NURBS data but is being extended to support NURBS data more robustly.

Once the ability to support reading and rendering geometric data through the IGES file format is accomplished, the next step of functionality within VE-Suite is the ability to interact with the geometric data that is imported into the library. The VE-NURBS library contains the capability to render the control points for a specific surface. The user is then able to interact with the control points through a wand or mouse and to move the points in space. The VE-NURBS library then redefines the surface without having to regenerate it, allowing for a more interactive exercise. Once the user has finished modifying the surface, the surface data can be saved in the IGES file format through the OCC library. The new IGES file can then be loaded back into Pro/E and utilized for other engineering operations.

المنارة للاستشارات

www.manaraa.com

Next, with the functionality to read IGES files and the ability to interact with the surfaces described by the IGES file, it becomes feasible to interact with other numerical computer aided engineering (CAE) tools and grid-dependent tools such as CFD and FEA software packages. CFD tools such as StarCD can import IGES files and mesh the resulting surface without user input. These pro-surface tools also have the ability to repair the surface to make it easier for StarCD to mesh. Once the surface has been repaired and meshed in pro-surface, StarCD .cel and .vrt files can be exported, and those files can then be imported into StarCD pro-am. Pro-am has the ability to take a surface mesh and generate a polyhedral volume mesh without user input. With the volume mesh complete, the model parameters can be defined and the model can be run.

With these three new capabilities, VE-Suite can complete the engineering loop from conceptual surface modeling to high-fidelity analysis to surface modeling. Certainly, as with any new software features, these features will need to be thoroughly tested and utilized in everyday cases, but the foundation has been laid to provide an environment in which all tools utilized by an engineer can be integrated into one environment for use throughout the engineering process.

VES files with all the current state information about a design can be saved, enabling the system to evolve over time. Just as an engineer would save various revisions to a CAD/CAE model, he or she can save various revisions to the virtual simulation constructed in the VE-Suite common user environment. This also enables the design to be tracked as it evolves through the design process.

المنارة للاستشارات
www.manaraa.com

4.4 Structure for VE-Suite Application Directory

The final component implemented for this research is a formalization of the directory structure utilized by VE-Suite applications. This structure enables a compact process for storing the data necessary for opening and looking at applications in VE-Suite and is comparable to application bundles in Mac OS 10.X [Apple Computers, Inc. 2005]. The directory structure enables future versioning enhancements to be explored, but also simplifies the data access within VE-Suite to enable data access without user intervention. With the proposed directory structure (Figure 17), each of the core VE-Suite engines can implicitly access any piece of information requested by the user.

المنارة للاستشارات

www.manaraa.com

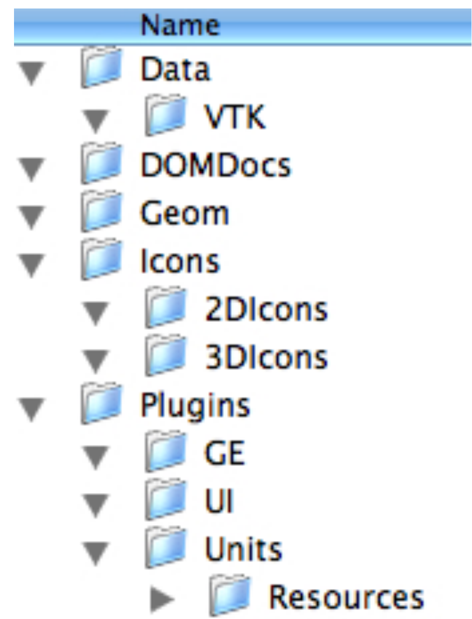


Figure 17. Sample VE-Suite application directory structure

www.manaraa.com

4.5 Summary

Two case studies will be examined to illustrate the new capabilities described above within VE-Suite. Each case uses a different set of functionality within the VE-Suite toolkit and will provide a means to better understand the object-centered method. The following chapter will discuss the use of these implementations with two applications that build on the new work discussed in the implementation chapter:

- The integration of VE-Suite and Aspen Plus, which will highlight the capabilities of the online mode in VE-Conductor and the new systems support
- The construction of tools that have the potential to reduce the complexity that the product engineer must manage when leveraging the new predictive modeling tools in the product development process

www.manaraa.com

Chapter 5: Large and Ultra-Large System Integration

In power plant design, access to a broad range of information is necessary to make informed decisions that impact plant performance, cost, and risk. Many information sources are available in today's engineering environment, from spreadsheet-based models to process models to CFD models. Each of these models provides valuable information for the decision-making process as well as a different and unique perspective on the power plant's design characteristics (Figure 16). Providing stakeholders with accurate, reliable, and complete information is an important characteristic of today's engineering tools. Coupling process simulation modeling with an information framework, which will provide stakeholders with process simulation modeling information in conjunction with three-dimensional CAD geometry, will be examined in this chapter. Presenting process simulation information in this format will help the engineer contextualize abstract simulation information.

The integration of two software frameworks, APECS (with Aspen Plus) and VE-Suite, will be examined in this chapter. In addition, this research highlights the capability to work with flowsheets containing hundreds of unit operations. This coupling will support automatic and manual mapping of pre-configured flowsheet interconnectivity to VE-Suite, automatic and manual configuration of Aspen Plus parameters for access in VE-Suite, and basic runtime control of APECS co-simulations from VE-Suite, all via VE-Conductor.

www.manaraa.com

This is a collaborative effort between Fluent, Ames Lab, Reaction Engineering International (REI), and the National Energy Technology Laboratory (NETL). The specific components completed for this research are the participation in the design of the CASI library, design and implementation of the VE-AspenUnit, and the modification of the CASI library to support some on-demand feature requirements to support real-time interaction with large systems of models.

One of the key elements of functionality required to couple VE-Suite and Aspen Plus is a wrapper, or abstraction, library for Aspen Plus. The function of the library is to provide a high-level C++ interface to the Aspen Plus software. In addition to a simplified interface, the library encapsulates the details of Aspen Plus interfacing in the library itself. While doing this, the library also maintains an external interface to keep from breaking the existing library client codes as well as to provide additional robustness enhancements. Key features of the library include:

- Implementation of C++ as a class library
- Hidden details of AspenTech's automation interface implementation (AspenTech's automation interface is undergoing rapid changes)
- Ease of use from non-managed C++
- Portability to other platforms (wrapper code)
- Simplified development of automation code for Aspen Plus

Both APECS/Aspen Plus and VE-Suite will be utilized to produce an immersive and interactive environment where these advanced power generation facilities can be

المنارة للاستشارات
www.manaraa.com

created. These two toolsets bring unique capabilities to the engineering environment that enable more efficient power plants to be constructed.

5.1 APECS

NETL and its R&D collaboration partners are developing APECS [Zitney] as a commercial software tool that combines process simulation with high-fidelity equipment models based on CFD. APECS enables engineers to better understand and optimize power plant performance with respect to coupled fluid flow, heat and mass transfer, and chemical reactions.

The APECS integration framework (Figure 18) uses the process industry-standard CAPE-OPEN [Pons 2003] software interfaces to provide plug-and-play interoperability between process simulation and equipment models. The hierarchy of equipment models ranges from high-fidelity CFD models to custom engineering models to fast reduced-order models (ROMs). At NETL, system analysts typically use APECS to run power plant co-simulations coupling the CAPE-OPEN-compliant steady-state process simulator, Aspen Plus, with CAPE-OPEN-compliant CFD models based on Fluent.

The APECS system reduces the time and effort required to couple CFD-based equipment models into plant-wide Aspen Plus simulations. Today, design engineers can use APECS to integrate CFD models into a process simulation in a matter of an hour or two by using the CAPE-OPEN software interfaces and a number of systematic and timesaving features, including easy-to-use configuration wizards and an equipment model database.

المنارة للاستشارات
www.manaraa.com

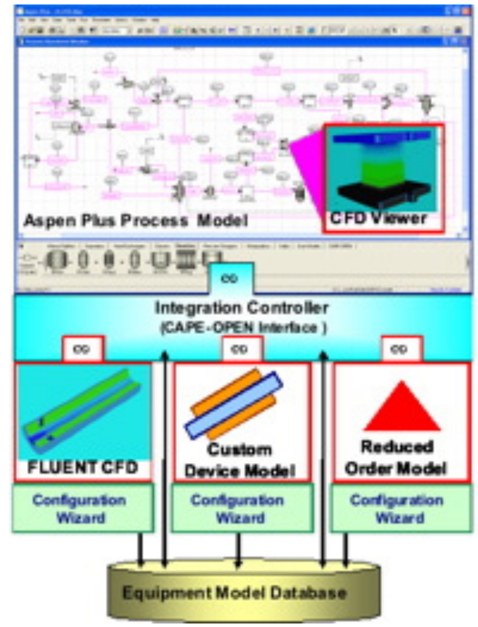


Figure 18. APECS software architecture

www.manaraa.com

To improve co-simulation turnaround time, APECS provides options on both ends of the performance spectrum, including the use of fast ROMs and parallel execution of the CFD models on high-performance computers. ROMs are a class of equipment models that are based on pre-computed CFD solutions over a range of parameter values, but are much faster than CFD models. For example, the APECS system currently provides for automatically generating and using a ROM based on multiple linear regressions to demonstrate the concept.

The APECS system also provides a wide variety of analysis tools for optimizing overall power plant performance. Design specifications are used to calculate operating conditions or equipment parameters to meet specified performance targets. Case studies are used to run multiple simulations with different input for comparison and study. Sensitivity analysis shows how process performance varies with changes to selected equipment specifications and operating conditions. Optimization is used to maximize an objective function, including plant efficiency, energy production, and process economics. For process optimization in the face of multiple and sometimes conflicting objectives, APECS offers stochastic modeling and multi-objective optimization capabilities developed to comply with the CO software standard.

In terms of this research, APECS represents an example of being able to integrate a closed source solver through the transparent interfaces. It provides unique capability that would otherwise be inaccessible to other components that are connected in the VE-Suite engineering framework.

المنارة للاستشارات

www.manaraa.com

5.2 Aspen Plus

Aspen Plus [AspenTech 2008] from Aspen Technology is a commercial, steady-state process modeling tool for steady-state simulation, design, performance monitoring, and optimization. The process simulation capabilities of Aspen Plus enable engineers to predict the behavior of a process using basic engineering relationships such as mass and energy balances, phase and chemical equilibrium, and reaction kinetics. Aspen Plus contains data, physical properties, unit operation models, built-in defaults, reports, and a wide variety of analysis tools including equation-oriented modeling, case studies, sensitivity analysis, and optimization.

For modeling coal-fired power generation systems, Aspen Plus offers solids handling capabilities important for combustion and gasification modeling; comprehensive physical properties, thermodynamics, phase and chemical equilibrium relations, and reaction kinetics for gas cleanup modeling; and an extensive library of heat exchange and rotating equipment models for simulating combined cycles.

Aspen Plus also offers an open environment to easily incorporate proprietary in-house or third-party technology. These may be created using Microsoft Excel®, FORTRAN, or Aspen Custom Modeler®. In addition, Aspen Plus supports the process industry standard, CAPE-OPEN.

5.3 CASI

The key motivation for creating the C/C++ Aspen Simulator Interface (CASI) library is to encapsulate the details of communicating with Aspen Plus (Figure 19). In this work, the Aspen Plus automation server is used to provide access to simulation data and control the execution of the simulator from VE-Suite. The Aspen Plus API is an ActiveX™

المنارة للاستشارات
www.manaraa.com

Automation Server. The ActiveX™ technology enables an external Windows™ application to interact with Aspen Plus through a programming interface using a language such as Microsoft's Visual Basic™. The server exposes objects through the COM object model.

AspenTech is planning to implement a number of changes to the automation interface that fundamentally alter how software must be written to utilize the interface. CASI limits the software modifications required to support future AspenTech changes to the CASI library. Thus, user code (including VE-Suite) does not require modification.

المنارة للاستشارات
www.manaraa.com

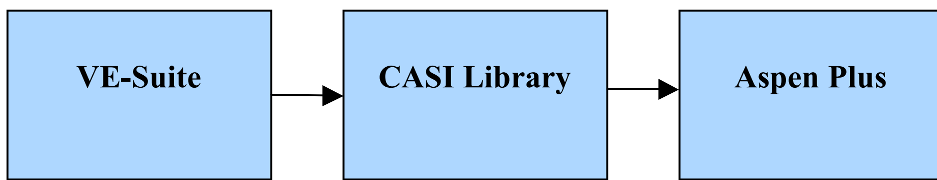


Figure 19. CASI software abstraction

www.manaraa.com

5.3.1 Object-oriented architecture

As noted previously, the CASI library has been implemented in object-oriented C++. This object orientation fits naturally with the Aspen Plus model of documents, unit operation blocks, process streams, and variables. The following sections provide additional details about the library.

The CASI library consists of three main C++ classes:

- *class Variable* – This is an abstraction for an Aspen Plus variable and has member functions for obtaining data associated with the variable. This class is derived from the CASIObj class because of the functional overlap between block, streams, and variables.
- *class CASIObj* : public Variable – This is an abstraction for both blocks and streams. From the standpoint of the class interface, both blocks and streams can be effectively represented by the same abstraction. Member functions include methods to obtain port information, chemical component information, and block inputs and outputs.
- *class CASIDocument* – This is an abstraction for the entire Aspen Plus flowsheet. Methods of this class allow the developer to load flowsheets, connect to the Aspen Plus automation engine, and obtain detailed information about the current active flowsheet.

5.4 VE-AspenUnit

The main component of the Aspen Plus integration with VE-Suite is the unit application referred to as the VE-AspenUnit. The VE-AspenUnit does the majority of the work required to access Aspen Plus functionality within VE-Suite. The rest of the VE-

المنارة للاستشارات
www.manaraa.com

Suite framework utilizes core functionalities present within each of its core engines in conjunction with the additions described in the implementation chapter. This design is chosen to enable the end user to utilize Aspen Plus and VE-Suite with minimal work required to integrate other software unit operations. The overall goal of this work is to show the capability to integrate an external third party closed software package and have it self-describe itself to the rest of the VE-Suite framework.

The research component here is to demonstrate that the VE-Open implementation discussed in the previous chapter is a viable solution to support mapping a power plant object described by Aspen Plus into VE-Suite. This example illustrates the capability to interact with hundreds of unit operations in real time within VE-Suite. For any block or stream, the respective results, inputs, and stream data is available to the engineer in real time. This is facilitated through the query-based interfaces described in Section 4.1. The VE-AspenUnit processes the VE-Open data generated from the CASI library. This design enables the VE-AspenUnit to broker requests between VE-Suite and Aspen Plus. In addition, graphics components can be overlaid on the unit operations that are queried from Aspen Plus. When VE-Suite is running with Aspen Plus, there is a one-to-one mapping of unit operations to graphics entities. This enables the engineer to associate CAD on a per-object basis in the environment. This is possible through the use of the Aspen Plus hierarchy blocks. Typically, there are unit operations on an Aspen Plus flow sheet that do not necessarily correspond to a physical object. These unit operations are utilized in Aspen Plus for the purpose of creating the best possible fidelity simulation of the physical system under review. The hierarchy blocks typically then represent a physical object. This enables

المنارة للاستشارات
www.manaraa.com

the user to drill down from the power plant object level in VE-Suite, to the systems level in the plant, to the sub-systems level, and then down to the part level (Figure 20).

The implementation changes discussed in the previous chapter enable the VE-AspenUnit to provide the user with easy access to any Aspen Plus flowsheet without having to edit code. Utilizing this functionality in VE-Suite requires the user to go through seven steps:

1. Launch VE-Suite (Figure 21)
2. Launch the VE-AspenUnit (Figure 22)
3. Open the flowsheet of interest (Figure 23)
4. Review input parameters (Figure 24)
5. Review results (Figure 25)
6. Review stream data (Figure 26)

Modifications to the core VE-Suite engines make these steps possible, but they can also be utilized by any third-party solver, enabling self-description of solvers to exist within the VE-Suite framework. In addition, this application highlights the capability to work with systems of systems within the VE-Suite framework (Figures 20, 14, 15).

المنارة للاستشارات

www.manaraa.com

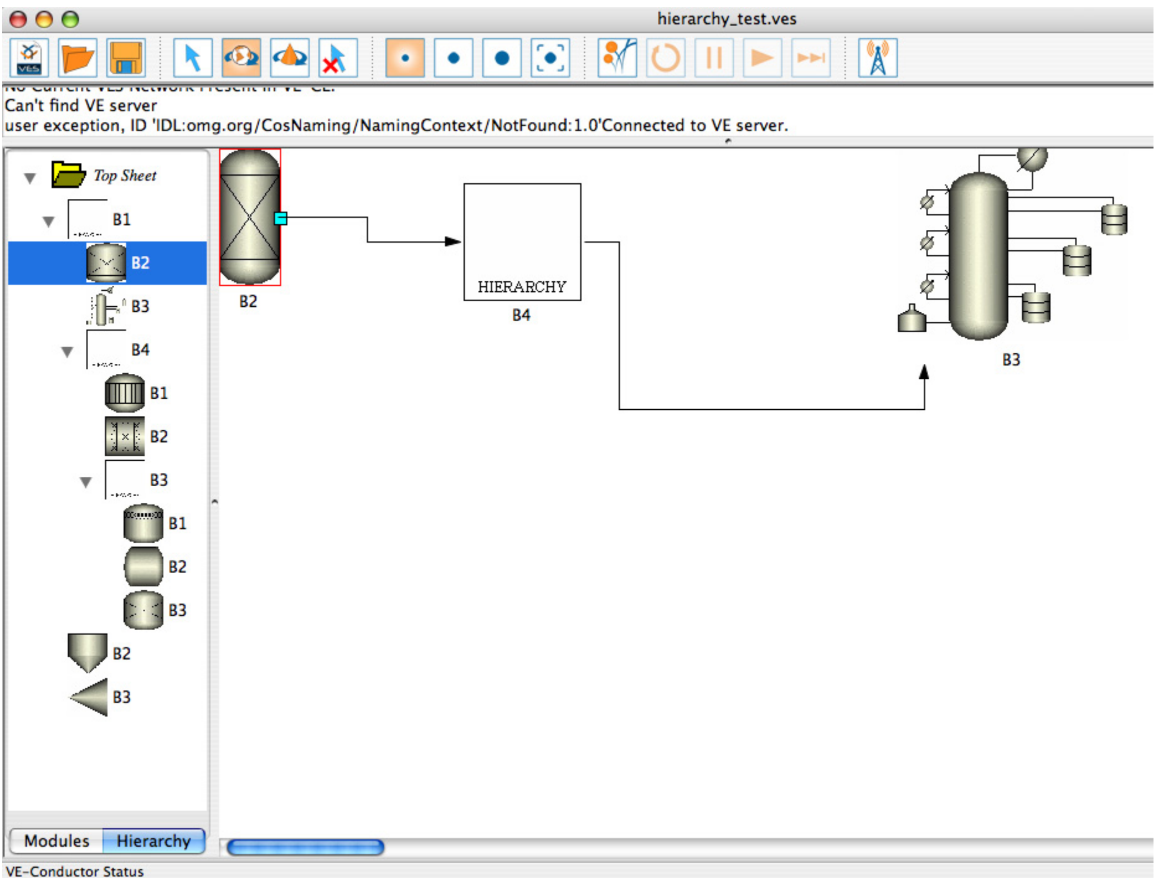


Figure 20. VE-Conductor hierarchy view

www.manaraa.com

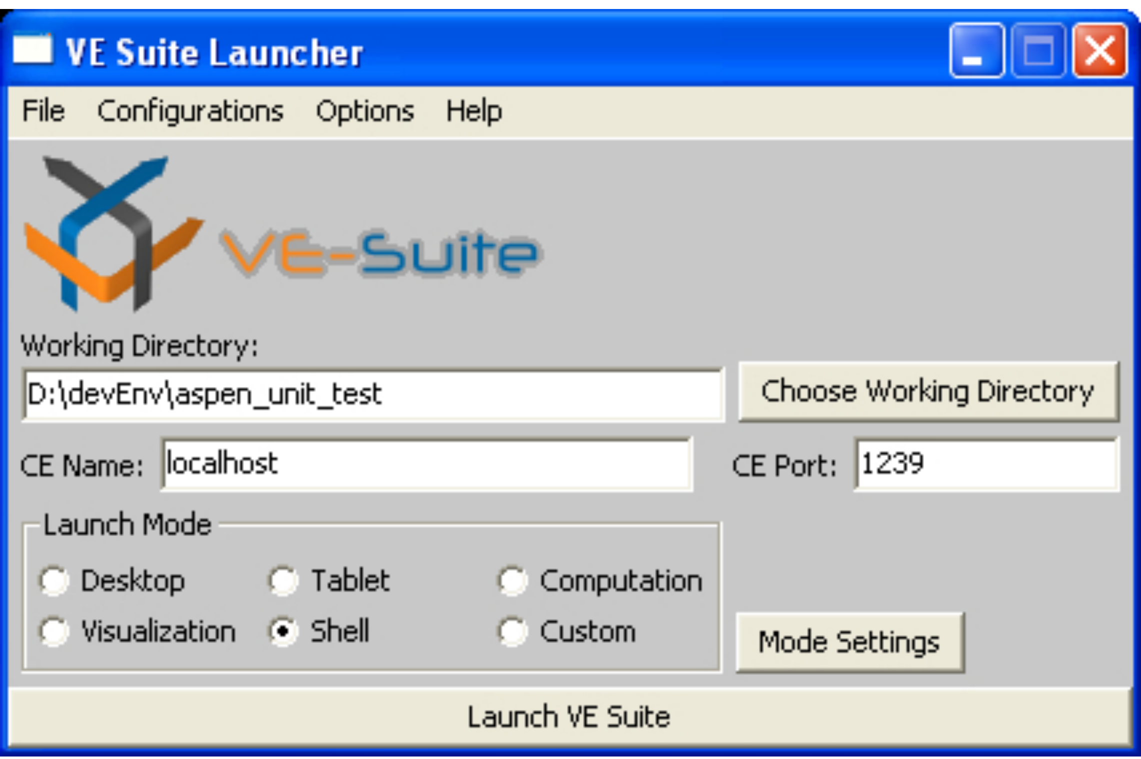


Figure 21. Launching VE-Suite

المنارة للاستشارات

www.manaraa.com

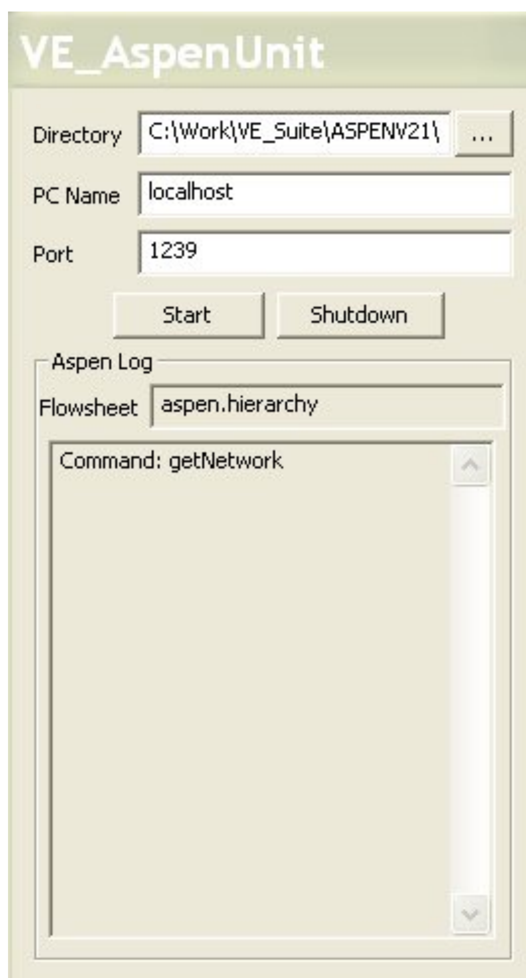


Figure 22. Launch the VE-AspenUnit

المنارة للاستشارات
www.manaraa.com

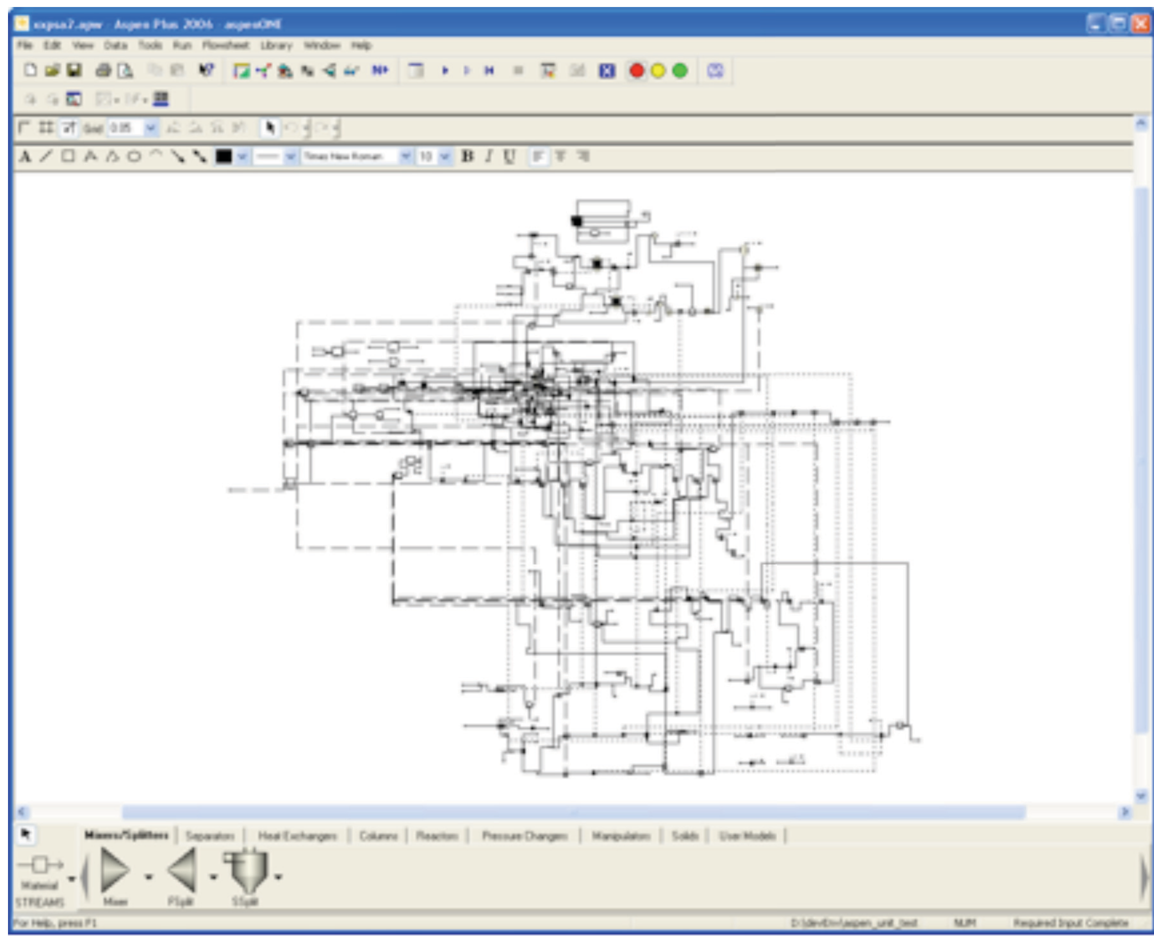


Figure 23. Opening the flowsheet of interest

المنارة للاستشارات
www.manaraa.com

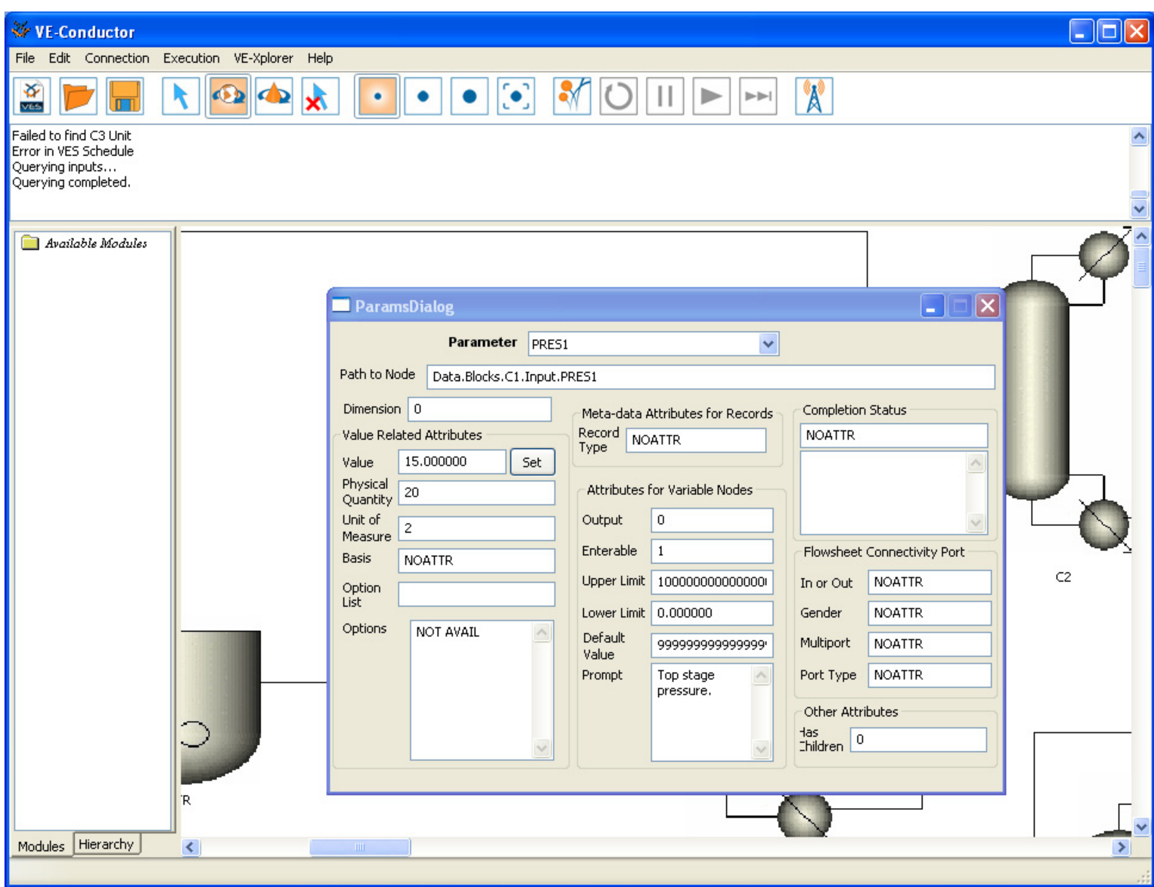


Figure 24. Review input parameters

المنارة للاستشارات
www.manaraa.com

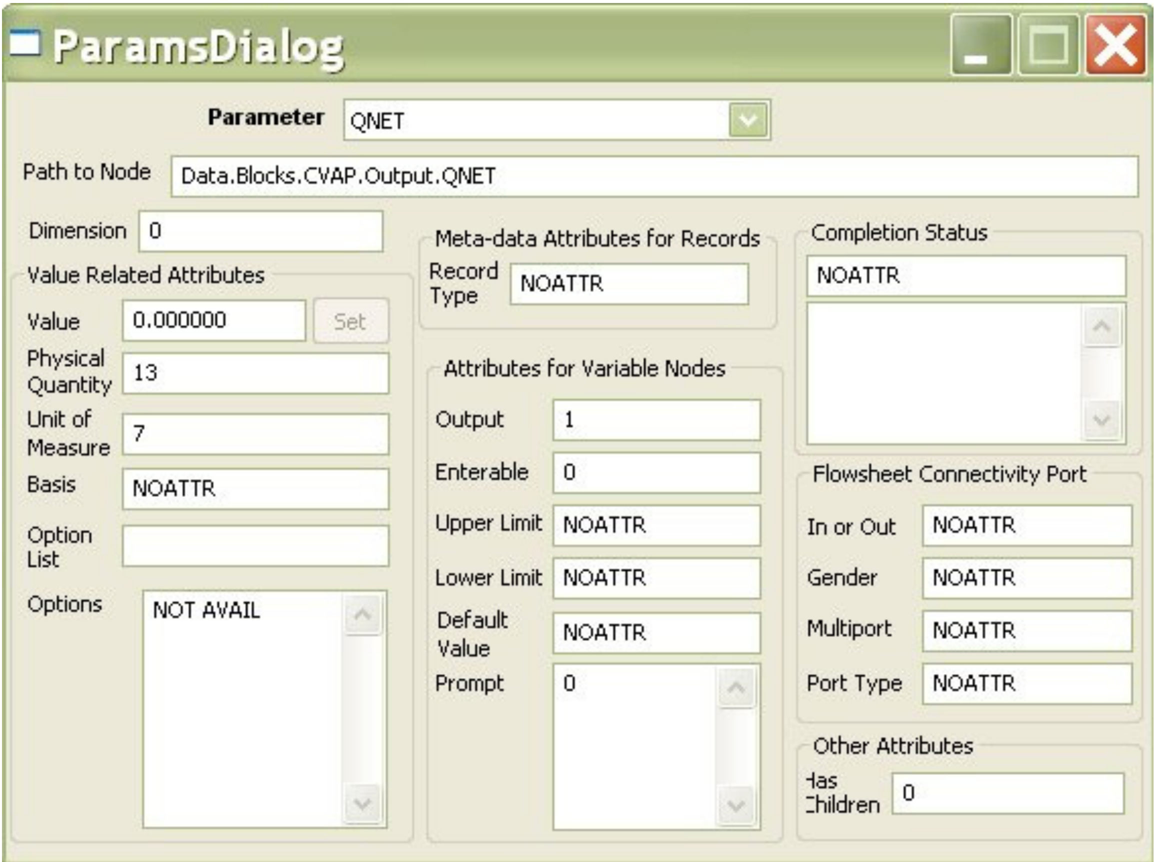


Figure 25. Review results parameters

المنارة للاستشارات
www.manaraa.com

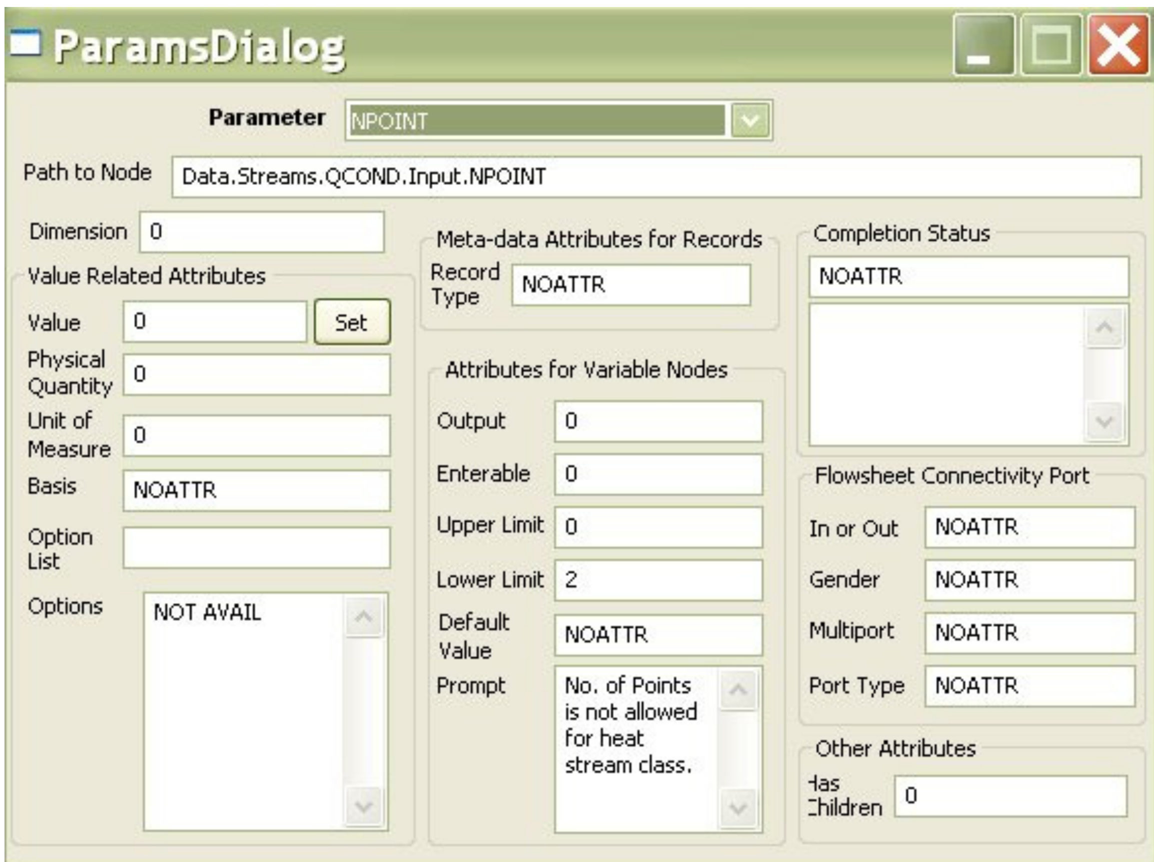


Figure 26. Review stream parameters

المنارة للاستشارات
www.manaraa.com

This research shows the integration of Aspen Plus and VE-Suite through the development of the CASI library and the VE-AspenUnit. The engineer's ability to interact with large systems of unit operations within VE-Suite has also been illustrated. The method of integrating VE-Suite and Aspen Plus also illustrates the use of objects in configuring the decision-making environment by the engineer. This is possible by enabling the engineer to overlay CAD, CFD, or FEA data on any Aspen Plus unit operation within VE-Conductor and have the data available within VE-Xplorer. This integration example also shows that VE-Open is capable of supporting large amounts of information from third-party solvers and simulators. VE-Open provides mechanisms for data to be stored in a modular manner and referenced hierarchically. These characteristics enable the real-time performance seen in this integration. The integration of Aspen Plus and VE-Suite enables more information to be accessible to stakeholders in creating advanced power generation facilities in the next decade. This toolset enables process simulation data to be presented in a format that is accessible to a broad audience.

المنارة للاستشارات

www.manaraa.com

Chapter 6: Engineering Mass Products

The second application created with VE-Suite in this research is focused on a cotton picker (Figure 27). The cotton picker [Arndt 2007] picks cotton without breaking the cottonseed in a cotton boll. This is accomplished through a sophisticated mechanical picking system, which will not be discussed here, and a pneumatic cotton conveying system. In this case, the air system is the subsystem that will be investigated on the picker platform. The cotton conveying system has three main components (Figure 28):

- The squirrel cage fan supplies air to the system.
- The manifold redirects air from the fan to three transport duct systems. The important characteristic of the manifold is to efficiently redirect the high-speed air from the fan to the transports ducts in a small space envelope.
- The transport duct system is composed of three sub-components: the lower duct, the nozzle, and the upper duct.

المنارة للاستشارات
www.manaraa.com

Figure 27. A cotton picker in the field

المنارة للاستشارات
www.manaraa.com

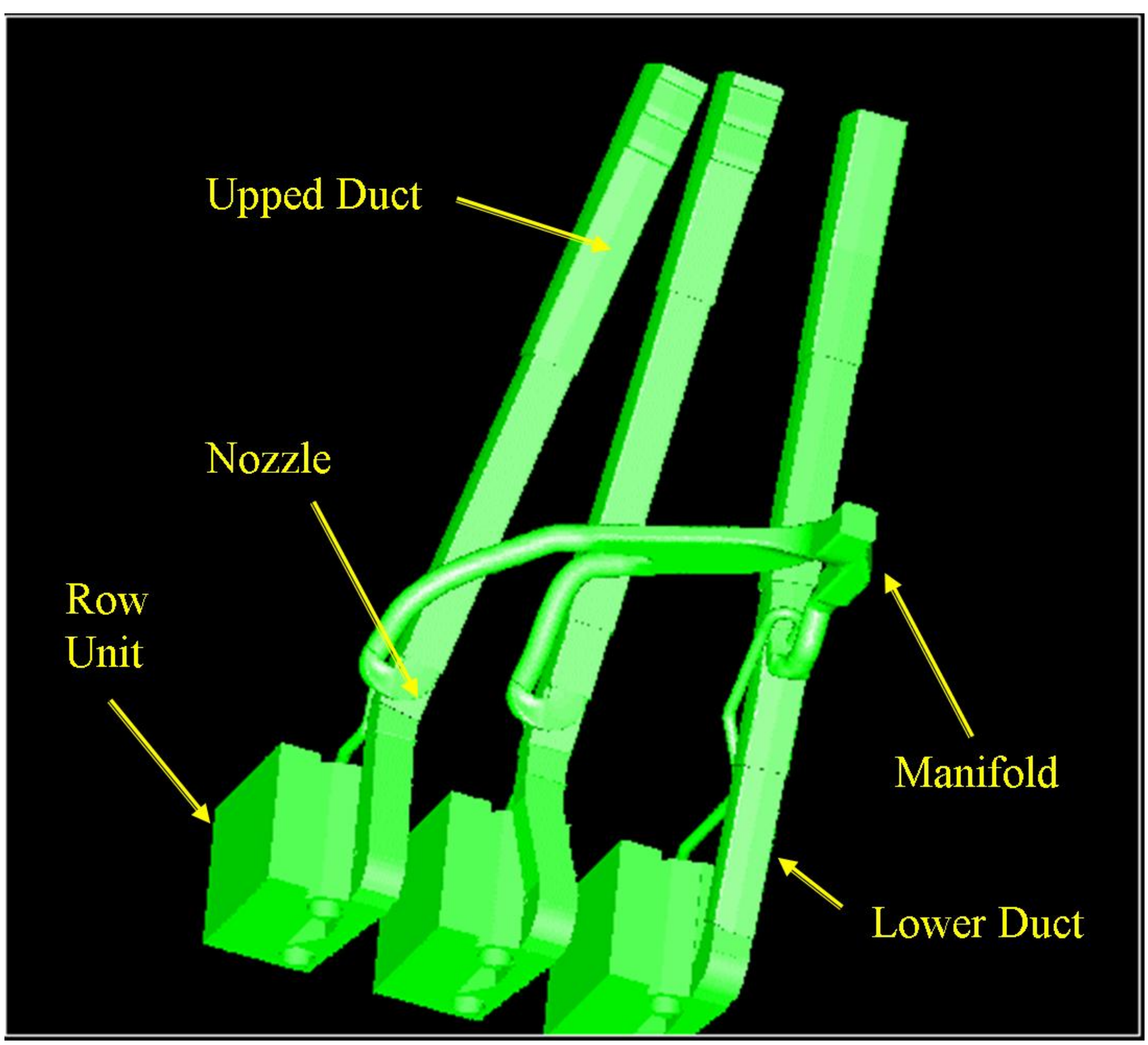


Figure 28. Cotton picker air system

www.manaraa.com

The goal of the design changes is to reduce the amount of energy required to drive the air system. To do this, each component, with the exception of the fan, will be modeled with CFD to better understand the airflow characteristics. The models are constructed to enable answering specific questions regarding power consumption. This case will illustrate the ability to design a subsystem of a complex product within the revised VE-Suite toolkit as well as the initial ability to pass high-fidelity boundary information from one discrete model to another. There are several steps in the engineering design process that benefit from the functionality that virtual engineering provides. During each phase of the six-step engineering life-cycle process [Blanchard et al. 1998], it is necessary to not only have seamless access to the necessary decision-making information created in each step, but to also have access to the information used in the previous steps of the design process. This enables each stakeholder during the life-cycle process to know immediately how decisions impact previous decisions and outcomes.

Currently, when a product such as the cotton picker is designed, each engineer on the picker design team stakes a claim on a part of the cotton picker platform to work on new components. For example, the air system will have certain space claims throughout the picker that may or may not contain the end solution or desired solution for the picker air system because the engineer has no idea where to begin looking for good designs within the space constraints.

Design suggestions are based primarily on past knowledge of the air system and not necessarily on a complete understanding of how the air system works. Once each team of engineers for various parts of the picker has staked their claim (e.g., frame, air system, engine, cab, etc.), more detailed work is done to try to understand how these system-of-

المنارة للاستشارات
www.manaraa.com

systems can coexist and interact on the same farm implement. It should also be remembered that engineers on the picker design team are not necessarily experts in the field for which they are required to design components. For example, an engineer may understand the basics of an engine and have the technical ability to find the necessary information to understand how an engine works, but he or she may struggle with how best to integrate the engine into the picker platform and how best to describe to the vendor the constraints on the type of engine he or she needs to place on the picker platform.

Designing the cotton picker platform should be a seamless process that enables engineers, marketers, and senior leadership to interact to make joint decisions to produce a product that will meet economic goals as well as performance and mechanical specifications. The process that the team goes through from proposal, to funding, to preliminary design, to production should be integrated and retrievable at any point in time. The current roadblock to the seamless occurrence of this process is primarily a lack of readily available information for the engineer and design team regarding specialized information such as the air system characteristics. This is mostly because the current design paradigm does not easily permit engineers and managers to ask questions without having to deal with the complex models and software packages (e.g., CFD) needed to answer those questions. In most cases, this interface is controlled by a human analyst who filters out the information they think is unnecessary. Much in the same way that computer numerical control machines took the place of humans running lathes, tools are needed that enable computers to control some of the analysis process during the design process.

As defined above, information must be exchanged between models and the engineer at multiple levels. The top level of this exchange would be the pure boundary

www.manaraa.com

condition information being shared between models, which would be noted as the explicit information. The implicit information would be the CFD information that can be gained if required by the engineer. For the most part, the engineer does not need to know the type of CFD package being used within the object, or the details of the CFD model. The engineer needs the errors and uncertainty associated with the model, and needs to understand the model results. Because the object has been preconfigured to answer specific questions, the engineer does not have to worry about asking a question that is answered with an invalid response, but can explore to find areas of interest.

The work described below is a product of this research except for the creation of the VE-NURBS library. The VE-NURBS library was completed as part of this research with the additional help of another graduate student in the Simulation, Modeling, and Decision Sciences Program.

6.1 Cotton picker models

The models that will be utilized in this problem will span the fluids modeling fidelities from algebraic expression-for-loss models to Navier-Stokes models. An inviscid flow model, which will span the previously mentioned models, will be run through a commercial solver. In each component of the cotton picker, these three models will be utilized as source of information (Figure 29). At some stages of the design process, the engineer only requires a low level of fidelity to make a decision, in which case the loss model or inviscid flow model would be useful. At other points in the design process, the higher-fidelity models would be required to adequately make a decision. The models will provide the necessary information to enable engineers to better understand the picker's physical characteristics. Each model will also have a specific error or uncertainty

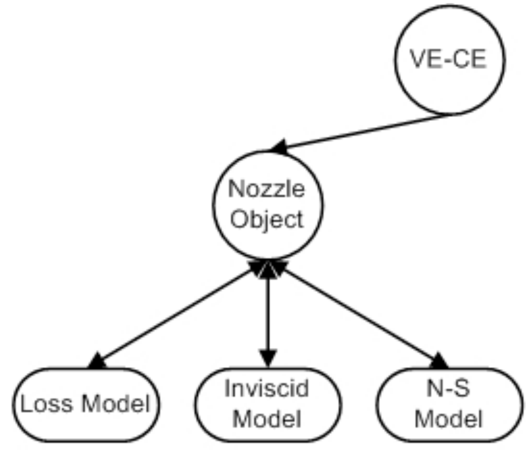
المنارة للاستشارات
www.manaraa.com

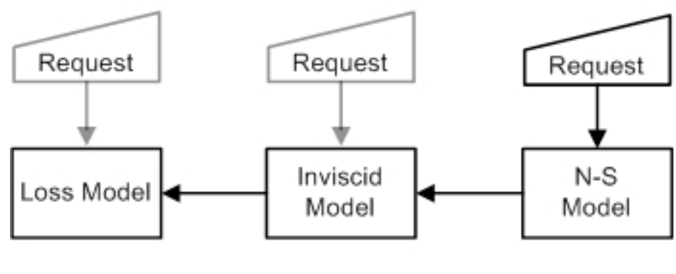
associated with it to enable the engineer to choose what level of information is needed given the time allowed for a decision.

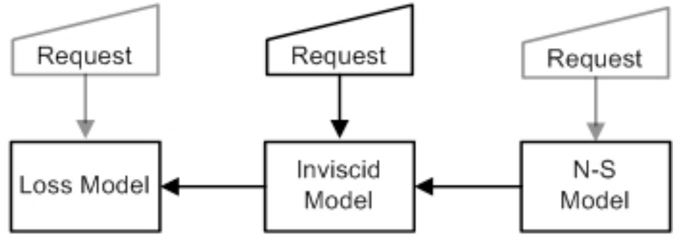
In addition, each of these models will be linked through a base object. In the future, the base object will enable the three sub-models discussed above to run the appropriate model based on the current area of investigation, in addition to the level of fidelity desired by the engineer. This process hides much of the redundancy in running and using models in the engineering process from the engineer. In the future, the models will be able to detect required updates. For example, if the Navier-Stokes model changes, all the lower-fidelity models should update accordingly so that they have the most recent data on which to base their calculations.

المنارة للاستشارات

www.manaraa.com







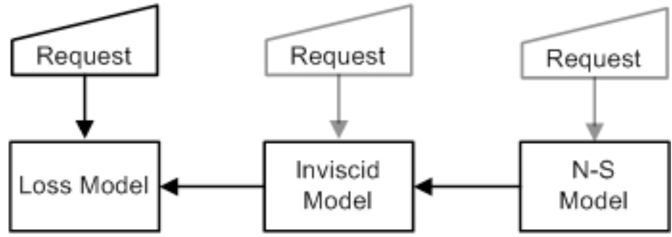


Figure 29. Numerical models for the cotton picker air system

www.manaraa.com

6.2 Utilizing the VE-NURBS tools for interactive CFD

The new VE-Suite tools discussed in the implementation section make it possible to take a volume mesh in a commercial CFD mesher and create a NURBS surface for importing into VE-Suite's NURBS tools. These tools enable manipulation of the NURBS surface from within the VE-Xplorer environment and to the ability to export the changed surface in IGES format. The basic steps to take advantage of hiding model interface complexity are:

- Create a surface in a CFD package using splines and patches, keeping track of the cell numbers for each batch. For example, if a patch is 60 cells by 20 cells and starts with cell 1, that patch contains cells 1 through 1200. This information is necessary in the next step.
- Once the surface is created in the CFD package, make sure that each of the patches is defined properly. Once the surface and patches have been checked, export the .cel and .vrt files for the resulting surface and create a NURBS file. A utility included with VE-Suite will take a .cel and .vrt file as input to create the NURBS surface.
- A utility in VE-Suite translates the file created above into an IGES file. Once the data created in the CFD package is in IGES format, all the functionality described in the implementation chapter is accessible to the engineer on the desktop.
- Create local coordinate systems on all of the boundary surfaces in the CFD package so that the boundary conditions at runtime can be defined without user input. In addition, all the model parameters need to be noted so that models created in the

المنارة للاستشارات
www.manaraa.com

interactive design phase can be run properly. This information should be stored in a formatted file for access by a VE-Suite unit.

- With the above files and data in place, the loop utilized within VE-Suite looks like this:
  - Preprocess the CFD model to generate the initial IGES file
  - Store boundary and model information for access by a VE-Suite unit
  - Load the initial IGES file into VE-Suite
  - Change the IGES file and save
  - Read the IGES file into the unit, remesh, and run
  - Send the data back to VE-Xplorer for review
  - Repeat until finished

With the above process in place (Figure 30), any numerical solver can be plugged into VE-Suite and utilized in an interactive design manner.

6.3 VE-Suite software plugins

Each of the plugins utilized for the cotton picker application is built on the standard plugins contained within VE-Suite. Utilizing these plugins eliminates the need for coding in the cotton picker application above and beyond the extensions described in the implementation chapter and the units that will be described below.

6.3.1 Graphical plugins

The graphical plugin is composed of the default capability within VE-Suite in addition to the capability to interactively transform the surface to enable an engineer to continuously design a component rather than using the discrete and linear engineering

www.manaraa.com

process described previously. Each graphical plugin for each component in the air system within the cotton picker will have a respective graphical plugin.

www.manaraa.com

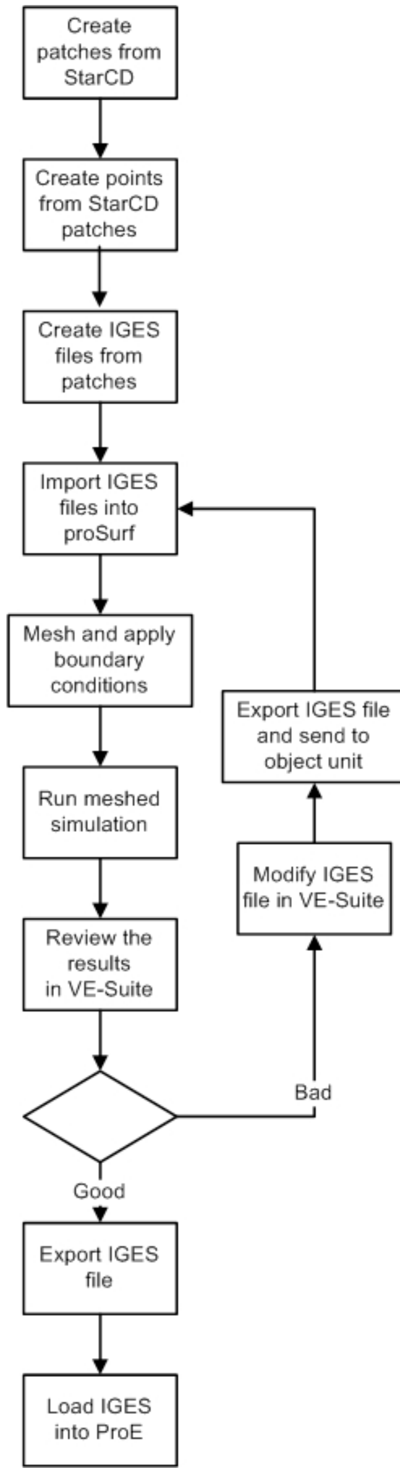


Figure 30. Interactive CAD process diagram for VE-Suite

المنارة للاستشارات

www.manaraa.com

6.3.2 UI plugins

The UI plugins in VE-Conductor utilize the standard plugin distributed with VE-Suite. The goal in developing this plugin is to enable the user to query the unit for the inputs that it provides the user to manipulate. This functionality is the first step toward a self-describing engineering object. In the case of the cotton picker, the only code that needs to be written by the user is the unit, which means that less of a burden is placed on the engineer in developing a virtual engineering environment. The unit will provide the inputs for the plugin.

6.3.3 Units

The software utilized to encapsulate these models requires an extension to the current VE-Suite software architecture. These software tools enable the initial implementation of models that will adapt to their surrounding models and enable the software tools to manage the information transfer for the user. This software extension primarily occurs in the computational unit interface of VE-Suite, which is located within the VE-CE software engine. To enable an object to be complex (i.e., composed of other sub-models), two new interfaces were added to VE-Open: VEObject and the InfoSource. The InfoSource represents a raw source of information such as the loss model, Navier-Stokes model, or inviscid flow model, in the case of the cotton picker. An InfoSource is not restricted to the implementation of a numerical model but can be extended to sensors, experimental data, or any other source of information that must be integrated into a product design environment. The VEObject is an extension to the base unit interface, but allows the registration of InfoSources to the VEObject, thus enabling a hierarchy of InfoSources to be constructed and a web of information created for that particular VEObject. The Web

المنارة للاستشارات
www.manaraa.com

can then be locally managed by the VEObject and can manage the operation of the various sub-InfoSources for the user so that information can be run and queried without user interaction or direction.

The interfaces for InfoSource and VEObject are implemented as follows:

```
//////////////////////////////////////////////////////////////////////////////
  interface InfoSource
  { // This is the interface for working with a hierarchy of models under one
    // unit operation. This is a beta interface.

     //This is for querying the status of the module
     string GetStatusMessage() raises(Error::EUnknown);

     //This is to Set the Module up
     void SetParams(in string param) raises(Error::EUnknown);

     //This is to get info source results - can be and sort of data
     string GetResults() raises(Error::EUnknown);

     //This is to Set the ID
     void SetID(in long id) raises(Error::EUnknown);

     //This is to Get the ID
     long GetID() raises(Error::EUnknown);

     //This is to Set the name
     void SetName(in string name) raises(Error::EUnknown);

     //This is to Get the name
     string GetName() raises(Error::EUnknown);
  };
  //////////////////////////////////////////////////////////////////////////////
  interface VEObject : Unit
  { // This is the interface for working with a VEObject. It inherits from Unit.

     //This is to disconnect the Unit to the Executive
     void UnRegisterInfoSource(in string InfoSourceName) raises(Error::EUnknown);

     //This is to Register a Unit to the Executive, flag=0 is normal module, flag=1 will be the global module
     void RegisterInfoSource(in string InfoSourceName,
                                        in Body::InfoSource infoSourceIn, in long flag) raises(Error::EUnknown);
   };
```

The additions to VEObject are necessary so that the VEObject knows what sub-InfoSources are connected to it and should be considered when accessing information about that object. This new function addition to the unit interface allows the InfoSources to be executed as follows:

```
# run the nozzle 1
export TAO_MACHINE=ids7
export TAO_PORT=1239
```

www.manaraa.com

```
NozzleUnitApp -ORBInitRef NameService=corbaloc:iiop:$TAO_MACHINE:$TAO_PORT/NameService -VESUnitName
NozzleObjectRow1 &
sleep 3
LossModelApp -ORBInitRef NameService=corbaloc:iiop:$TAO_MACHINE:$TAO_PORT/NameService -VESObjectName
NozzleObjectRow1 -VESInfoUnitName NozzleLossModelRow1 &
```

TAO_MACHINE and TAO_PORT are the port numbers and machine where the naming server runs for The ACE ORB (TAO). The command line flag VESUnitName enables the unit wrapper code to be the same for multiple objects. For example, the code that is written for the nozzle object can be utilized for the upper and lower ducts because the generic object code only has to broker the information flow from each InfoSource to the user and the computational engine. This brokering of information operates on the same command structure that is discussed with the Aspen Plus integration. As with the changes implemented in the computational engine to enable self-description of large simulation software such as Aspen Plus, the same techniques can be implemented in individual objects to enable the code to be extensible. When each nozzle registers with the TAO naming service, it registers a name that enables the InfoSources to look up the respective object that it is associated with. In the future, this lookup and connection with a VEObject may occur without having to specify a particular VEObject to connect to as the networks grow to include hundreds or thousands of InfoSources and objects. When this occurs, the users running these virtual simulations will probably be unable to know all the names of the VEObjects to connect to.

6.4 Engineer's Experience

With the tools implemented above, the engineer has the capability to construct a complex system from a bottom up approach. As the engineer drags the components of the air system onto the VE-Conductor design canvas, he or she is also constructing the

www.manaraa.com

network used by VE-CE to determine the execution without user input. All the engineer is doing is connecting the components of the air system together just like he or she would do with the physical components. This information can then be saved out in the DOMDocument format. This data can be saved at various intervals to enable model state information to be retrieved at later dates to gain insight into why various engineering decisions were made.

www.manaraa.com

Chapter 7: Results & Conclusions

The goal of this research is to outline the necessary requirements and components for an advanced engineering framework to enable a bottom-up design approach in the engineering process through the use object-oriented methods. These requirements and components enable the construction of engineering objects that change the engineering design experience. As noted in previous chapters, the engineer does not have to be concerned with the underlying numerical models or the details of the implementation of the models. The engineer just has to construct the system and decide what modifications must be considered.

The implementation and example applications in this paper illustrate that engineering objects can be used to characterize information management in the engineering design process. This enables engineers to work with large systems generated from secondary applications. In addition, the ideas and software implemented here change how models may be segregated to improve the engineering workflow by providing a new way to characterize information. The changes implemented within VE-Suite have enabled it to become another tool within the engineering design process.

This dissertation has laid out the initial requirements for methods to address the demands of the large amounts of information available in today's engineering decision-making process. Many potential areas of research must still be explored to understand

المنارة للاستشارات

www.manaraa.com

engineering informatics requirements, including the use of engineering objects. Objects must enable computers to augment the human capabilities of integrating information, understanding relationships between different sets of information, and providing contextual information that may aid in providing further insight into a problem.

In this initial research, there were no signs that the VE-Open implementation would not support interacting with ultra-large systems. The example problems illustrated the benefits of enabling query and on-demand interface specification and data structures. This type of method enables the user to query as much information as necessary and to provide real-time control of a complex simulator such as Aspen Plus. In addition, this interface is not limited to integrating VE-Suite with Aspen Plus. The thin-layered CASI library provides an example of how to convert data from a closed-source solver to the broader VE-Suite framework. In addition, it illustrates the capability to interface with systems-of-systems from within VE-Suite. These two example applications provide a brief look at how new tools utilizing semantic and meta-data-based tools can benefit the engineering process by providing intelligent applications to the engineer's desktop. These tools are not developed and researched to stay in the scientific academic community, but will be delivered to the desktop of the engineer so that a new engineering workflow can be created.

Utilizing the model of the scale-free networks has been shown to enable the VE-Suite unit operations that connect to VE-CE to grow without restriction. These scale-free networks provide the capability to handle information queries and lookups within subcomponents of systems. For example, in the cotton picker example, the specific solvers (i.e., loss model, inviscid flow model, or Navier-Stokes model) can all update and

المنارة للاستشارات
www.manaraa.com

communicate simultaneously without having to contact the VE-CE because the InfoSource does not have to contact the VE-CE to obtain the necessary information for the respective models. This model has been shown to provide a localized control schema that enables the VEObject to handle appropriate requests as needed by the local VEObject unit operation.

This research proposes applications of Semantic Web technologies to software packages used by the engineering community. The same tools that enable information integration and contextualization on the Internet could also enable integration of engineering tools and specifications, allowing the product development cycle to be completed in an unprecedented manner. Semantic Web tools that will be used to contextualize the engineering environment are XML and XML Schema, XSL, and OWL. Engineering information will be disseminated via web pages that will allow users from around the world to see model-specific information. Ontologies will also be used to classify information and to show the connection and hierarchy of information sources so that connections between entities in VE-Suite are clear.

The object-centered method aims to address many of the issues facing the current engineering design process by enabling the engineer to focus on engineering and not on information integration. To illustrate the proposed capability of the object-centered method, an initial implementation of the XML schema has been described. The schema is currently in active use within VE-Suite as the core communication and data transfer mechanism. VE-Suite proposes to enable a broad range of problems to be addressed across many disciplines for the complete lifecycle of a product or system. This will enable engineers to focus on using the information provided by engineering models and other diverse information models to make decisions in the product realization process. The initial

المنارة للاستشارات
www.manaraa.com

interface specification, VE-Open, will enable engineers to address these multi-disciplinary issues and to collaborate at a level that enables information to flow from one design team to another. Implementing the object-centered method will enable the problems experienced when collaborating within large design teams to become less intrusive in the engineering decision-making process. The object-centered method, when implemented across each step of the product realization process, will create environments where virtualized systems and parts can be analyzed and produced with far fewer costs devoted to the design and development phase of the realization process.

This work has presented a foundation on which to build efforts to change the engineering process. This foundation has included:

- Development of engineering objects
- Development of an initial advanced engineering framework
- Implementation of the VE-Open XML schema and CORBA IDL interface
- Support of third-party numerical solvers containing large systems of unit operations
- Support of segregated numerical models for product sub-systems
- Implementation of methods to construct systems at run-time within VE-Suite
- A self-describing interface specification for third-party solvers

These additions will enable future work to be completed in the areas of drag-and-drop numerical integration, creation of narrative environments, and agent-based engineering support algorithms. In addition, different approaches to the problem of integration of large systems of models and solvers can be investigated. One new approach to be explored is a bottom-up method of handling the integration and distribution of solver information. This will enable the user to be unrestricted in the number of models that can be integrated, as

المنارة للاستشارات
www.manaraa.com

illustrated with the cotton picker example for running models. These new research areas show promise in being able to investigate problems across modeling scales, fidelities, and in investigating as-built problems that exist for large systems.

www.manaraa.com

Appendix A: VE-Suite Description

VE-Suite [Bryden et al. 2004] is intended to be used in the engineering process, whether for business model investigation or training. It is used in a diverse set of engineering applications to allow engineers and other project stakeholders to gain insight into complex engineering problems. VE-Suite's extensible software design enables users to incorporate component models and corresponding two-dimensional and three-dimensional graphical representations to create new plug-and-play framework components. By design, the framework components can be distributed across computational resources to make the most efficient use of resources.

In nearly all aspects of the engineering process—design, manufacturing, and maintenance—the tools employed at each phase rely on virtual models (e.g., software tools) to reduce cost and shorten development time. This results in a wider variety of software tools being used across a wide range of vendors and engineering firms. In this environment, engineers are required to manually move information from one software package to another. Thus, the process does not support real-time, collaborative design in which the engineer establishes the dynamic thinking process needed to obtain an intuitive feel for the performance of a product. It also does not permit the real-time exploration of questions raised by other engineers, designers, or managers. This working arrangement significantly lessens the number of alternatives that can be investigated, limits the essential

المنارة للاستشارات
www.manaraa.com

creative design process, and discourages "what if" questions that can lead to breakthroughs in design. As a result, the engineer has to shift his or her focus from engineering to manual information integration. To allow engineers to focus on engineering, a new workflow and paradigm is needed. This workflow is described within a new enabling technology called virtual engineering and is implemented via VE-Suite. Using VE-Suite to implement virtual engineering reduces the design cycle time to allow new technologies to reach production and operation more quickly than previously possible. Engineering tools and information need to be integrated throughout each engineering project. That is, information from the design phase needs to be available to design and manufacturing contractors without manual reentry or other hassles. Currently, for a variety of reasons (e.g., budgetary constraints and inter-company politics), no commercial software package can integrate information from the complete product design team, from economists and numerical modelers to design and manufacturing firms. VE-Suite addresses this constraint by creating a tool that has open interfaces and allows other commercial and open-source packages to exchange data in a comprehensive design environment. In this environment, all the data and tools necessary to make a particular engineering decision are available to the stakeholder trying to move the engineering process forward.

When creating tools to enable engineers to use engineering analysis to make more informed decisions, it is necessary to take into account the broad range of analysis that may be used in the engineering product realization process. This review process reveals a broad range of problems. Some current products will require the following types of models:

المنارة للاستشارات

www.manaraa.com

- Graphical
- Requirements
- Budgets
- Physics
- Simulation results
- Input/output data and data structures
- Finite element
- Numerical

These models highlight the breadth of the information that must be handled by an engineering decision-making framework. The framework should enable engineers to access the proper fidelity of information when needed throughout the engineering process.

The engineer's ability to plug any model and source of information into this virtual engineering framework is its primary design goal. Without the ability to plug and play with models, the engineer becomes bogged down in coupling software rather than creating or solving complex engineering problems. The software framework must promote changing the way complex systems are engineered rather than trying to integrate the tools that are already in the mix. The framework described here will leverage the areas of research described previously to create a framework that will allow a modular development process to occur in addition to being flexible enough to fit into many different design processes. By allowing information to be extracted and added whenever the user desires, the framework can be adapted to many different design methodologies. Modularity must be supported to create fundamental modules that the engineer can work with. These modules, by definition, also carry with them a set of rules that dictate the construction and operation of the

المنارة للاستشارات
www.manaraa.com

modules. Modularity permits the engineer using VE-Suite to connect specific components and representations of components together to create the desired system. This enables the engineer to focus on the outcome of the system rather than the components of the system, allowing him or her to add more capability to the system under design and preventing problems as the system grows and evolves over time.

This section provides an overview of VE-Suite's software design and implementation. VE-Suite contains four software engines: VE-Open, VE-Conductor, VE-CE, and VE-Xplorer. The first VE-Suite engine described, VE-Open, is the proposed communication standard that will allow VE-Suite's software engines and objects to be integrated. The key elements of the VE-Suite framework design are the user interface, computational engine, visualization engine, and component models. Note that the various software elements all exist as independent CORBA [Object Management Group, Inc. 2008] components with standardized Interface Definition Language (IDL) implementations defined within the proposed standard, VE-Open. The use of component architecture design techniques has numerous advantages for this application, including platform independence, location transparency, and reuse of component models [Verbaeck 2004].

Model integration and communication: VE-Open

The VE-Open design builds on an open architecture approach to integrating information as well as on the neutral format described earlier. VE-Open utilizes both integration formats by specifying a schema for information to adhere to and leveraging other schemas such as COLLADA [Arnaud et al. 2006], which has taken a useful approach

المنارة للاستشارات
www.manaraa.com

to creating an extensible specification built on XML and XML Schema. COLLADA focuses heavily on games and on the physics and polygonal data representation issues surrounding games. Therefore, it ignores many CAD-related issues, which benefits COLLADA significantly because many side issues fall outside the project scope or need to be left for other projects. In addition, this tight focus can benefit other tools such as VE-Open by providing a solid method to reference schema data within VE-Open. This approach enables VE-Open to remain lightweight while still utilizing work from other projects and specifications. The component models described below have access to this information, which enables more physical attributes to be accessed by the engineering objects.

Component models are mathematical representations of individual virtual objects that are used by the framework to construct an overall simulation. The key to making the simulation framework extensible is to provide a mechanism by which component models can easily be integrated without extensive software development. To address this need, the relatively modern idea of component architecture design has been adopted. CORBA is used along with a standard model interface definition, which is implemented as an IDL and referred to as VE-Open [VESuite.org 2008], to create componentized computational models. These models can be used interchangeably with any framework that supports the standardized IDL, are location transparent (run on any network accessible machine), platform independent (Linux, Windows, etc.), and programming language flexible, and can be distributed in binary form.

The interface to the CORBA-based component models is designed to allow the models to be autonomous, accepting inputs and stream data from the computational

المنارة للاستشارات
www.manaraa.com

engine, running the encapsulated model, and generating outputs and modified stream data. It is important to note that the CORBA interface between the computational engine and the component models is the standardized model interface supported by the framework for model integration. The interface defined for VE-Suite, VE-Open, is analogous to that of the CAPE-Open specification used by chemical process simulation tools. VE-Open is also analogous to the Distributed Interactive Simulation (DIS) [Distributed Interactive Simulation 1999] specification utilized in military applications to share war game simulation information across distributed compute resources with multiple clients. The VE-Open model interface has a number of unique characteristics:

- Simplicity: The functions that are implemented are general and can be adapted to a wide variety of simulation environments.
- Generalization: The new interface removes the specificity of any discipline and provides generic structure for data types and software engine structure.
- Enhanced data passing: The new interface provides facilities for passing data beyond the level of simple scalars to downstream models.

In addition to specifying the communication standard for how core engines and component models will communicate, the VE-Open specification also includes an XML schema that defines how commands and data arrays can be constructed and passed to the various parts of a virtual object.

The XML schema that is contained within VE-Open defines how simple data arrays and other key data structures used within VE-Suite should be constructed. This portion of VE-Open is a key component in enabling the data that is used by the mathematical representation of the virtual object to be easily understood by the three-

المنارة للاستشارات
www.manaraa.com

dimensional graphical representation of the object. The XML schema does not only allow the computational engine to gain information about the proposed simulation; any other component within the VE-Suite framework can also gain information about the system under design.

Graphical user interface: VE-Conductor

The graphical user interface (UI) is implemented with the following software design goals: multi-platform support, detachability, location transparency, extensibility, and unified control. The UI is the controller that allows the engineer to interrogate the virtual design environment. It also makes use of platform-independent libraries to enable the software to run on a wide range of computer hardware and operating systems ranging from Unix workstations to Pocket PCs and PDAs. After reviewing a number of different UI libraries, WxWidgets [WxWidgets 2008] was chosen for use in VE-Conductor. A list of available modules is maintained in a tree structure on the left side of the window, while the main canvas area shows the current simulation network.

The UI exists independently from the computational engine as a separate CORBA component. This functionality allows the UI to be attached and detached from an active simulation on any compatible computer on the simulation network. For example, a user could build and start a simulation, detach from the computational engine or visualization engine, go to a different location, re-attach to the simulation, and regain monitoring and control functions. This detachable UI is where the user can create a plant configuration, set model inputs, start and stop simulation execution, and view simulation results. Once a client-server connection is made, the engine is able to send results, messages, updates, and

المنارة للاستشارات
www.manaraa.com

communications from other attached UIs in real time. Users can connect or disconnect at will to configure, modify, or monitor the simulation of a given plant configuration.

To accomplish this functionality, a CORBA IDL interface between the UI and the computational engine was defined and the UI was designed to communicate via CORBA to both the computational engine and the graphical engine. The CORBA interface provides all the necessary communication mechanisms between these components. The communication link is bidirectional, handling items such as model parameters passed to the computational engine and receiving items such as execution status and results from the computational engine. This specification allows the UI to provide unified control for all user interaction, ensuring that the user is not burdened with moving among different UIs to perform operations. There is a single UI with the ability to monitor and control the virtual design environment. The interface specification is open source, so it is possible for other research groups to implement a proprietary UI that adheres to the specification and communication protocol.

Another advantage of this design is the ability for multiple UIs to be attached to the same computational engine, allowing multiple users to monitor a simulation from different locations. The UI also has the ability to connect to the graphical environment and control what graphical representations are shown for high-fidelity data (e.g., contour planes, vector planes, streamlines, iso-surfaces) or for low-fidelity data (e.g., gauges showing scalar information about plant performance, costing data, or emissions data). The connection between the UI and the visualization engine is similar to the connection between the UI and the computational engine. This communication link is also bidirectional and is used to direct what is shown within the virtual design environment.

المنارة للاستشارات
www.manaraa.com

Another important consideration for the UI design is extensibility. The UI is able to dynamically discover, identify, and load UI elements for new component models. This capability keeps the level of difficulty involved in integrating new component models to a minimum because it eliminates the need for modifications to the core interface when new models are added. The dynamic discover-and-load capability is accomplished by loading user-developed module UIs from dynamic link libraries (DLL in Windows) or shared libraries (shared object library in Linux/Unix). A plugin C++ base class defining this UI-module interface is provided to all module developers. Developers can inherit from this class to create their own module UIs and then compile the resulting code into a DLL/shared library. The UI framework's plug loader code will recognize the new module and bring that into its user-module library. By this mechanism, the core UI can plug in the third-party module-specific UI directly from binaries. This mechanism allows users to develop custom input and results interfaces. One of the benefits of this design is that it allows the core VE-Suite engines to focus on handling information flow and not on the development of UIs.

Computational Engine: VE-CE

The computational engine (VE-CE) constructs, coordinates, schedules, and monitors simulation runs. It is capable of running a simulation containing a multitude of different types of models, each accepting and generating a myriad of data types. The computational engine is also able to analyze a simulation configuration, determine execution order, marshal system resources to create model instances, and coordinate the flow of data through the simulation framework. Tasks that require specific knowledge

المنارة للاستشارات
www.manaraa.com

about a data type or model are relegated to either the detachable UI or to a specific model, thus keeping the computational engine highly generalized and lightweight code-wise.

Important functions that the computational engine controls can be broken down into several pieces for explanation: configuration, data handling, error handling, relationship to the detachable UI, scheduling, and relationship to the models. The configuration of a simulation, provided by a detachable UI, is the primary data structure used by the computational engine. Nearly all algorithms utilized, such as proper data flow, scheduling, and resource allocation, depend on this topology. This configuration is constructed from the XML schema contained within the VE-Open specification that was discussed previously. The XML data structure contains information about how one virtual object connects to another and allows multiple virtual objects to share information about what data types to expect from another object. Through this XML schema, it is possible for other engines to be developed that can accept the scheduling data structure from the UI. The scheduler that uses this configuration data is capable of handling single and embedded feedback loops, iterative solves and, eventually, transient simulation runs.

Because there is an unlimited number of possible models capable of being integrated into the framework (with each model having a different input/output set), the computational engine operates with generalized data types. To address this requirement, the CORBA IDL interfaces between the computational engine and the component models use mapped string blocks in combination with common dimensions of array data. With the computational engine as the central intelligence behind a simulation run, all errors that occur while performing this task, whether originating within the engine’s own structure or on an attached model, must be properly handled within the context of the overriding

المنارة للاستشارات
www.manaraa.com

structure. Thus, the computational engine has error handling routines and messaging facilities to alert attached users. The computational engine does not require a connection to a UI during a simulation run.

The computational engine, with its CORBA interface, is able to connect to the various component models available for a simulation. Information passed through this connection includes inputs (user supplied and stream data), outputs, results, and general messages. The importance of the CORBA interface being used for this purpose is discussed in detail in the Model Integration section above.

Graphical Engine: VE-Xplorer

The graphical engine (VE-Xplorer) provides the core functionality for the virtual engineering aspect of the framework, enabling the engineering analysis and design process to take place in a virtual environment. For maximum graphical performance on multiple operating systems, it is built upon VRJuggler [VRJuggler 2007], OpenSceneGraph [OSG Community 2007], and Kitware's Visualization ToolKit [Kitware 2005]. This visual interface, controlled by the UI and the computational engine, provides a graphical representation of the simulation under review.

The graphical engine is generalized to load data not only from comprehensive models, but also from other engineering sources and other generalized datasets (e.g., experimental data from a test rig). The engine is also being modified to make use of the high-level CORBA interface specifications used throughout the software framework. This interface allows the visualization engine to communicate directly with the component models, computational engine, and UI. To communicate with the graphical engine, an external socket connection is made between individual component models and the

المنارة للاستشارات
www.manaraa.com

respective graphical objects. This connection allows large high-fidelity datasets to be transferred to the graphical environment without interrupting the overall communication network.

The graphical engine is also designed to allow graphics objects to be added to the virtual environment in the same way that objects are added in the UI. This allows the graphical environment to be a direct representation of the system being designed by the engineer. In much the same way that the UI auto-discovers the plugins for use by the engineer, the graphical engine also dynamically discovers plugins. Unlike the UI, the graphical engine is controlled by the network string that is created by the UI. This represents a significant capability because the graphical engine has no a priori knowledge of the system under interrogation.

المنارة للاستشارات
www.manaraa.com

Appendix B: Example DOMDocument


```xml
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<network name="Network" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="veshader.xsd">
 <veSystem id="4b7b94c1-bf85-4564-84d5-341555e37cc5">
  <network>
   <conductorState dataName="m_xUserScale" id="76568581-a29c-4442-99b7-300a934b8aa2">
    <dataValue type="xs:double">1</dataValue>
   </conductorState>
   <conductorState dataName="m_yUserScale" id="22fc8d04-d28f-734c-bf6c-e3646f058d28">
    <dataValue type="xs:double">1</dataValue>
   </conductorState>
   <conductorState dataName="nPixX" id="32554c86-43c0-dd48-8712-bee05134d2b6">
    <dataValue type="xs:integer">10</dataValue>
   </conductorState>
   <conductorState dataName="nPixY" id="0cb96330-e297-ce4f-9b1e-9b65b3653326">
    <dataValue type="xs:integer">10</dataValue>
   </conductorState>
   <conductorState dataName="nUnitX" id="0dcba897-aba5-fa48-b8ef-ecfb6f8f6cea">
    <dataValue type="xs:integer">240</dataValue>
   </conductorState>
   <conductorState dataName="nUnitY" id="26621844-e51d-894b-8473-968d5f77c23b">
    <dataValue type="xs:integer">240</dataValue>
   </conductorState>
  </network>
  <model ID="102" id="56b3dba4-fa6b-4b34-9226-c181996af2ca" name="DefaultPlugin"
vendorUnit="DefaultPlugin">
   <iconLocation xLocation="10" yLocation="10"/>
   <icon iconMirror="0" iconRotation="0" iconScale="1" type="xs:string">DefaultPlugin</icon>
   <informationPackets id="5e62c92e-8820-4972-be32-a211a7b55fdc">
    <blockID type="xs:unsignedInt">105</blockID>
    <blockName type="xs:string">simple</blockName>
    <transform objectType="Transform">
     <translation>
      <value>0</value>
      <value>0</value>
      <value>0</value>
     </translation>
     <scale>
      <value>1</value>
      <value>1</value>
      <value>1</value>
     </scale>
     <rotation>
```


المنارة للاستشارات
www.manaraa.com

```
          <value>0</value>
          <value>0</value>
          <value>0</value>
        </rotation>
      </transform>
      <properties dataName="VTK_DATA_FILE" id="b55e4c6c-2ad8-044e-aa1c-862b7bf8f040"
objectType="DataValuePair">
        <dataValue type="xs:string">3scl2vec.vtu</dataValue>
      </properties>
      <properties dataName="VTK_TEXTURE_DIR_PATH" id="9a96837e-722f-b847-ade6-eccdc9d1619b"
objectType="DataValuePair">
        <dataValue type="xs:string">simpleScalars/scalars/200_to_1000</dataValue>
      </properties>
      <properties dataName="VTK_TEXTURE_DIR_PATH" id="2c3aa880-6cb0-b541-a758-d9cbfb398a2e"
objectType="DataValuePair">
        <dataValue type="xs:string">simpleScalars/scalars/first-scalar</dataValue>
      </properties>
    </informationPackets>
    <informationPackets id="d58ba6c4-b064-49f6-b79c-4386d1fe4191">
      <blockID type="xs:unsignedInt">108</blockID>
      <blockName type="xs:string">Dataset2</blockName>
      <transform objectType="Transform">
        <translation>
          <value>-4</value>
          <value>0</value>
          <value>0</value>
        </translation>
        <scale>
          <value>1</value>
          <value>1</value>
          <value>1</value>
        </scale>
        <rotation>
          <value>40</value>
          <value>40</value>
          <value>0</value>
        </rotation>
      </transform>
      <properties dataName="VTK_DATA_FILE" id="57183100-f39b-4ae4-8d76-d660cc67881b"
objectType="DataValuePair">
        <dataValue type="xs:string">3scl.vtu</dataValue>
      </properties>
      <properties dataName="VTK_PRECOMPUTED_DIR_PATH" id="90cbb734-1030-4480-b9d3-
7b2a3bed49ee" objectType="DataValuePair">
        <dataValue type="xs:string">POST_DATA1</dataValue>
      </properties>
    </informationPackets>
    <informationPackets id="df0dd685-f485-41ff-ad3c-66bb5bc15655">
      <blockID type="xs:unsignedInt">111</blockID>
      <blockName type="xs:string">Dataset3</blockName>
      <transform objectType="Transform">
        <translation>
          <value>0</value>
          <value>0</value>
```

www.manaraa.com

```
          <value>0</value>
        </translation>
        <scale>
          <value>0.25</value>
          <value>0.25</value>
          <value>0.25</value>
        </scale>
        <rotation>
          <value>-45</value>
          <value>0</value>
          <value>0</value>
        </rotation>
      </transform>
      <properties dataName="VTK_DATA_FILE" id="5c6059ea-db96-4317-9b70-64b366bac7e1"
objectType="DataValuePair">
        <dataValue type="xs:string">mb.vtu</dataValue>
      </properties>
    </informationPackets>
    <geometry associatedDataset="NONE" friction="1" id="5593f230-9f76-3449-acca-41bdeb92bc7a"
mass="1" physics="false" restitution="0" visibility="true">
      <type>Assembly</type>
      <name>Model_Geometry</name>
      <parent type="xs:string"></parent>
      <transform>
        <translation>
          <value>0</value>
          <value>0</value>
          <value>0</value>
        </translation>
        <scale>
          <value>1</value>
          <value>1</value>
          <value>1</value>
        </scale>
        <rotation>
          <value>0</value>
          <value>0</value>
          <value>0</value>
        </rotation>
      </transform>
      <numChildren>3</numChildren>
      <children>
        <child friction="1" id="1cfa8fb2-b11b-0645-8a2e-7ddc3fc7c582" mass="1" physics="false"
restitution="0" visibility="true">
          <type>Part</type>
          <name>eightCorners</name>
          <parent type="xs:string">5593f230-9f76-3449-acca-41bdeb92bc7a</parent>
          <transform>
            <translation>
              <value>0</value>
              <value>0</value>
              <value>0</value>
            </translation>
            <scale>
```

www.manaraa.com

```
      <value>1</value>
      <value>1</value>
      <value>1</value>
    </scale>
    <rotation>
      <value>0</value>
      <value>0</value>
      <value>0</value>
    </rotation>
  </transform>
  <attribute>
    <type type="xs:string">Material</type>
    <blending type="xs:boolean">true</blending>
    <material>
      <kDiffuse>
        <value>1</value>
        <value>0</value>
        <value>0</value>
        <value>1</value>
      </kDiffuse>
      <kEmissive>
        <value>0</value>
        <value>0</value>
        <value>0</value>
        <value>1</value>
      </kEmissive>
      <kAmbient>
        <value>1</value>
        <value>1</value>
        <value>1</value>
        <value>1</value>
      </kAmbient>
      <specular>
        <value>1</value>
        <value>1</value>
        <value>1</value>
        <value>1</value>
      </specular>
      <opacity>1</opacity>
      <shininess>50</shininess>
      <materialName>red</materialName>
      <face>Front_and_Back</face>
      <colorMode>Ambient_and_Diffuse</colorMode>
    </material>
  </attribute>
  <activeAttributeName type="xs:string">red</activeAttributeName>
  <fileName>eightCorners.stl</fileName>
</child>
<child friction="1" id="6382ac8c-ebe1-5543-85b8-2f511424db9d" mass="1" physics="false"
restitution="0" visibility="true">
  <type>Part</type>
  <name>Surface0.75</name>
  <parent type="xs:string">5593f230-9f76-3449-acca-41bdeb92bc7a</parent>
  <transform>
```

www.manaraa.com

```
        <translation>
          <value>0</value>
          <value>0</value>
          <value>0</value>
        </translation>
        <scale>
          <value>1</value>
          <value>1</value>
          <value>1</value>
        </scale>
        <rotation>
          <value>0</value>
          <value>0</value>
          <value>0</value>
        </rotation>
      </transform>
      <fileName>Surface0.75.stl</fileName>
    </child>
    <child friction="1" id="72bd0eb4-bac9-9f4e-894d-2bb98e5c7bef" mass="1" physics="false"
restitution="0" visibility="true">
      <type>Part</type>
      <name>Surface0.75_cloned</name>
      <parent type="xs:string">5593f230-9f76-3449-acca-41bdeb92bc7a</parent>
      <transform>
        <translation>
          <value>-4</value>
          <value>0</value>
          <value>0</value>
        </translation>
        <scale>
          <value>1</value>
          <value>1</value>
          <value>1</value>
        </scale>
        <rotation>
          <value>40</value>
          <value>40</value>
          <value>0</value>
        </rotation>
      </transform>
      <fileName>Surface0.75.stl</fileName>
    </child>
   </children>
  </geometry>
 </model>
</veSystem>
<User id="231bf178-8cad-4174-b8ad-25a1ca28681d" userID="User" veControlStatus="MASTER">
  <stateInfo>
    <Command commandName="CHANGE_BACKGROUND_COLOR">
      <parameter dataName="Background Color" id="9b147e9c-22dc-47b1-90c7-3dd46e7b8081">
        <genericObject objectType="OneDDoubleArray">
          <data>0</data>
          <data>0</data>
          <data>0</data>
```

www.manaraa.com

```
<data>1</data>
</genericObject>
</parameter>
</Command>
</stateInfo>
</User>
</network>
```

www.manaraa.com

Bibliography

Abodeely, J. (2007). Virtual design for the interactive placement of baffles in air flow. . Iowa State University.

Abrahamson, S., Wallace, D., Senin, N., & Sferro, P. (2000). Integrated design in a service marketplace, *Computer-Aided Design*, 32, 97-107.

Abrahamson, S., Wallace, D., Senin, N., & Borland, N. (1999). Integrated engineering, geometric, and customer modeling: LCD projector design case study. In DETC/CIE-9084. Las Vegas, NV: American Society of Mechanical Engineers.

Allan, B., Armostrong, R., Lefantzi, S., Ray, J., Walsh, E., & Wolfe, P. (2005)., Ccaffeine - a CCA component framework for parallel computing, *The Common Component Architecture Forum*. Retrieved March 31, 2008, from http://www.cca-forum.org/ccafe.

Antoniou, G., & van Harmelen, F. (2004). *A Semantic Web Primer*. . Cambridge, MA: The MIT Press.

Apple Computers, Inc. (2005). Bundle Programming Guide. . Apple Computers, Inc. Retrieved March 24. 2008 from http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFBundles/CFBundles.pdf.

Arnaud, R., & Barnes, M. C. (2006). *Collada: Sailing the gulf of 3D digital content creation*. Wellesley, Massachusetts: AK Peters, Ltd.

Arndt, M. (2007, July 2). Deere's revolution on wheels, *Business Week*(4041), 78.

AspenTech (2008), AspenOne, *AspenTech*. Retrieved March 31, 2008 from http://www.aspentech.com.

Bak, P. (1996). *How Nature Works: The Science of Self-Organized Criticality*. New York: Springer-Verlag.

www.manaraa.com

Baldwin, C. Y., & Clark, K. B. (2000). *Design Rules, Volume I: The power of modularity*. Cambridge, Massachusetts: MIT Press.

Baldwin, C. Y., & Clark, K. B. (2006). Modularity in the design of complex engineering systems. In *Complex Engineered Systems, ed. D. Braha, AI. A. Minai, Y. Bar-Yam* (1st) (pp. 175-205). New York: Springer.

Barabasi, A. (2003). *Linked: How everything is connected to everything else and what it means*. New York: Plume.

Belleman, R. G., Kaandorp, J. A., & Sloot, P. M. A. (1998). A virtual environment for the exploration of diffusion and flow phenomena in complex geometries, *Future Generation Computer Systems*, 14, 209-214.

Bellinger, G. (2004). Creating knowledge objects, *Systems Thinking*. Retrieved February 19, 2008, from http://www.systems-thinking.org/cko/guide.htm.

Blaize, M., Knight, D., & Rasheed, K. (1998). Automated optimal design of two-dimensional supersonic missile inlets. *Journal of Propulsion and Power*, 14(6), 890-898.

Blanchard, B. S., & Fabrycky, W. J. (2005). *Systems Engineering and Analysis*. , Prentice-Hall International Series in Industrial and Systems. (4th). Indianapolis, IN: Prentice Hall.

Blascovich, J., Loomis, J., Beall, A., Swinth, K., Hoyt, C., & Bailenson, J. (2002). Immersive virtual environment technology as a methodological tool for social psychology. *Psychological Inquiry*, 13(2), 103-124.

Bliznakov, P. I. (1996). Design information framework to support engineering design process. Arizona State University.

Bliznakov, P. I., Shah, J. J., & Urban, S. D. (1996). Integration infrastructure to support concurrence and collaboration in engineering design. In 96-DETC/EIM-1420. Irvine, CA: ASME.

Booch, G. (1982). Object-oriented design, *ACM SIGAda Ada Letters*, 1(3), 64-76.

Bricken, M. (1990). *Virtual Worlds: No Interface to Design*. . Seattle, WA: Human Interface Technology Laboratory, Washington Technology Center, University of Washington.

www.manaraa.com

Bryden, K. M., & McCorkle, D. S. (2004). VE-Suite: A foundation for building virtual engineering models of high performance, low emission power plants. In the 29th International Technical Conference on Coal Utilization & Fuel Systems. Clearwater, FL.

Bryden, K. M., Ashlock, D. A., McCorkle, D. S., & Urban, G. L. (2003). Optimization of heat transfer utilizing graph based evolutionary algorithms, *International Journal of Heat and Fluid*, 24, 267-277.

Bumpus, W., Sweitzer, J. W., Thompson, P., Westerinen, A. R., & Williams, R. C. (2000). *Common information model: Implementing the object model for enterprise management*. Hoboken, NJ: John Wiley & Sons.

Cao, Q., Senin, N., & Wallace, D. R. (2005). Functional classification of computational services in an internet-based distributed modeling environment. In *Proceedings of IDETC/CIE 2005*, DETC2005-85276. Long Beach, CA: ASME.

Chang, T. K., Yu, D. L., & Yu, D. W. (2004). Neural network model adaptation and its application to process control. *Advanced Engineering Informatics*, 18(1), 1-8.

Colombo, G., Mosca, A., & Sartori, F. (2007). Towards the design of intelligent CAD systems: An ontological approach. *Advanced Engineering Informatics*, 21(2), 153-168.

Cubert, R. M., & Fishwick, P. A. (1998). MOOSE: an object-oriented multimodeling and simulation application framework, *Simulation*, 70(6), 379-395.

Culler, G. J., & Fried, B. D. (1963). An on-line computing center for scientific problems. In *Proceedings of the 1963 Pacific Computer Conference* (pp. 44-53).

Cutkosky, M. R. et al. (1993). Pact: An experiment in integrating concurrent engineering systems, *Computer Magazine*, 26(1), 28-37.

Dabke, P., Cox, A., & Johnson, D. (1998). NetBuilder: An environment for integrating tools and people, *Computer-Aided Design*, 30(6), 465-472.

Dahl, O., Myhrhaug, B., & Nygaard, K. (1968). Some features of the SIMULA 67 language. In *Proceedings of the Second Conference on Applications of Simulations* (pp. 29-31). New York: Winter Simulation Conference.

Dahl, O., & Nygaard, K. (1966). SIMULA: An ALGOL-based simulation language, *Communications of the ACM*, 9(9), 671-678.

www.manaraa.com

Davis, J. (1999). Improving intelligence analysis at CIA: Dick Heuer's contribution to intelligence analysis. Retrieved December 8, 2005, from http://www.odci.gov/csibooks/19104/art3.html.

Dialog (2007). Distributed information architectures for collaborative logistics, *Dialog* Retrieved March 31, 2008, from http://dialog.hut.fi/.

Distributed Interactive Simulation DIS-Java-VRML Working Group. (1999). Frequently Asked Questions (FAQs), *Distributed Interactive Simulation DIS-Java-VRML Working Group*. Retrieved March 31, 2008, from http://web.nps.navy.mil/~brutzman/vrtp/dis-java-vrml/faq.html.

Dodgson, M., Gann, D., & Salter, A. (2005). *Think, Play, Do: Technology, innovation, and organization* (1). New York: Oxford University Press.

Dörner, R., Grimm, P., & Abawi, D. F. (2002). Synergies between interactive training simulations and digital storytelling: a component-based framework, *Computers & Graphics*, 26, 45-55.

Drashansky, T. T., Weerawarana, S., Joshi, A., Weerainghe, R. A., & Houstis, E. N. (1996). Software architecture of ubiquitous scientific computing environments for mobile platforms. *Mobile Networks and Applications*, 1, 421-432.

Duffy, A. H. B., Persidis, A., & MacCallum, K. J. (1996). NODES: A numerical and object based modelling system for conceptual engineering design, *Knowledge-Based Systems*, 9, 183-206.

Eckert, C., & Boujut, J. (2003). The role of objects in design co-operation: Communication through physical or virtual objects, *Computer Supported Cooperative Work*, 12, 145-151.

Engelbrecht, J. J. (2007). Optimization of a hydraulic mixing nozzle. Iowa State University.

Engineous Software (2007). Fiper, *Engineous Software*. Retrieved March 31, 2008, from http://www.engineous.com/product_FIPER.htm.

Engineous Software (2007). iSIGHT – Integrate, automate, and optimize your manual design processes, *Engineous Software*. Retrieved April 4, 2008 from http://www.engineous.com/product_iSIGHT.htm.

www.manaraa.com

Ergen, E., Akinci, B., & Sacks, R. (2007). Life-cycle data management of engineered-to-order components using radio frequency identification. *Advanced Engineering Informatics*, 21(4), 356-366.

Evektor (2008). Avionics and Electrical System, *Evektor – Design & Engineering.* Retrieved April 5, 2008 from http://www.evektor.cz/evektor/en/aeroElectroSystems.asp.

Fabbri, G. (1998). Optimization of heat transfer through finned dissipators cooled by laminar flow. *International Journal of Heat and Fluid Flow*, 19, 644-654.

Fan, H. (1998). An inverse design method of diffuser blade by genetic algorithms. *Proceedings of the I MECH E Part A Journal of Power and Energy*, 212(4), 261-268.

Feldman, A. (2005). ReachMedia: On-the-move interaction with everyday objects. Massachusetts Institute of Technology.

Feldman, A., Tapia, E. M., Sadi, S., Maes, P., & Schmandt, C. (2005). ReachMedia: On-the-move interaction with everyday objects. In *Proceedings of the Ninth IEEE International Symposium on Wearable Computers* (pp. 52-59). Osaka, Japan: IEEE.

Fishwick, P. A. (1996a). Extending object-oriented design for physical modeling, *ACM Transactions on Modeling and Computer Simulation*.

Fishwick, Paul A. (2006) The language of modeling for multidisciplinary design optimization. In *Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Portsmouth, VA: AIAA.

Fishwick, P. A. (1996b). *Toward a convergence of systems and software engineering.* University of Florida: Department of Computer and Information Science and Engineering.

Fishwick, P. A. (2004). Toward an integrative multimodeling interface: A human-computer interface approach to interrelating model structures, *SIMULATION*, 80(9), 421-432.

Foster, G. F., & Dulikravich, G. S. (1997). Three-dimensional aerodynamic shape optimization and gradient search algorithms. *Journal of Spacecraft and Rockets*, 34(1), 36-42.

Foucault, M. (1994). *The order of things*. New York: Vintage Books.

Fourman, M. (2002). *Informatics*. Edinburgh, Scotland: Division of Informatics, University

المنارة للاستشارات
www.manaraa.com

of Edinburgh.

Framling, K., Ala-Risku, T., Kärkkäinin, M., & Hölmstrom, J. (2007). Design patterns for managing product life cycle information, *Communications of the ACM*, 50(6), 75-79.

Gallopoulos, E., Houstis, E., & Rice, J. R. (1994). Computer as thinker/doer: Problem-solving environments for computational science, *Computational Science & Engineering*, 1(2), 11-23.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design patterns: abstraction and reuse of object-oriented design. In *Lecture Notes in Computer Science, Proceedings of the 7th European Conference on Object-Oriented Programming*, 707 (pp. 406-431). London: Springer-Verlag.

Garcia, A. C. B., Kunz, J., Ekstrom, M., & Kiviniemi, A. (2004). Building a project ontology with extreme collaboration and virtual design and construction, *Advanced Engineering Informatics*, 18, 71-83.

Gatenby, D. A. G. (2005, September). Galatea: Personalized interaction with augmented objects. Massachusetts Institute of Technology.

Gehlert, A., & Esswein, W. (2007). Toward a formal research framework for ontological analyses. *Advanced Engineering Informatics*, 21(2), 119-131.

George, D., & Hawkins, J. (2004). *Invariant pattern recognition using Bayesian inference on hierarchical sequences.* Tempe, AZ: RNI, Inc.

George, D., & Hawkins, J. (2005). A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks*, 3 (pp. 1812-1817). Montreal, Quebec, Canada: IEEE.

.

Goldstein, I. P., & Bobrow, D. G. (1980). Extending object oriented programming in Smalltalk. In *Proceedings of the 1980 ACM Conference on LISP and Functional Programming* (pp. 75-81). Stanford University, CA: ACM Press.

Google (2008). Google Maps, *Google Maps.* Retrieved April 5, 2008 from http://maps.google.com/.

Graphviz (2008). Graphviz - Graph Visualization Software, *Graphviz.* Retrieved March 31, 2008, from http://www.graphviz.org/.

www.manaraa.com

Graphviz (2008). The DOT Language, Graphviz – Graph Visualization Software, *Graphviz*. Retrieved March 31, 2008, from http://www.graphviz.org/doc/info/lang.html.

Gruber, T. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 199-220.

Hawkins, J. (2004). *On intelligence*. New York: Times Books.

Hawkins, J., & George, D. (2006). *Hierarchical temporal memory: concepts, theory, and terminology*. Numenta Inc.

Heim, J. A. (1997). Integrating distributed simulation objects. In *Proceedings of the 1997 Winter Simulation Conference* (pp. 532-538). Atlanta, GA: ACM.

Herman, Ivan (2007). Web Ontology Language (OWL), *W3C Semantic Web*. Retrieved March 31, 2008, from http://www.w3.org/2004/OWL/.

Hopkins, J. F., & Fishwick, P. A. (2001a). The rube framework for personalized 3-d software visualization, *Lecture Notes in Computer Science*, Revised Lectures on Software Visualization, International Seminar., 2269, 368-380.

Hopkins, J. F., & Fishwick, P. A. (2001b). A three-dimensional synthetic human agent metaphor for modeling and simulation, *Proceedings of the IEEE*, 89(2), 131-147.

Horn, B. (1993, November 28). Constrained objects. Carnegie Mellon University.

Horváth, L. (1997). Some possibilities for integrated intelligent object based engineering modeling. In *Proceedings of the 5th International Symposium of Hungarian Researchers on Computational Intelligence* (pp. 527-532). Budapest, Hungary: IEEE.

Horváth, L., & Rudas, I. J. (1994). Human computer interactions at decision making and knowledge acquisition in computer aided process planning systems. In *Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics* (pp. 1415-1419). San Antonio, TX: IEEE.

Horváth, L., & Rudas, I. J. (2001). Modeling of the background of human activities in engineering modeling. In *Proceedings of the 27th Annual Conference of the IEEE Industrial Electronics Society*, 1 (pp. 273-278). Denver, CO: IEEE.

Horváth, L., & Rudas, I. J. (2003). Integrated environment-adaptive virtual model objects

www.manaraa.com

for product modeling. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1 (pp. 498-503). Washington DC: IEEE.

Horváth, L., & Rudas, I. J. (2004a). Modeling behavior of engineering objects using design intent model. In *Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society*, 1 (pp. 872-876). Roanoke, VA: IEEE.

Horváth, L., & Rudas, I. J. (2004b). Some possibilities for integrated intelligent object based engineering modeling. In *Proceedings of the 5th International Symposium of Hungarian Researchers on Computational Intelligence*. Budapest, Hungary.

Horváth, L., & Rudas, I. J. (2004c). Adaptive objects for behavior based product models. In *Proceedings of the 2004 IEEE International Symposium on Industrial Electronics*, 1 (pp. 651-656). Ajaccio, France: IEEE.

Horváth, L., & Rudas, I. J. (2004d). Active description of engineering objects for modeling in extended companies. In *Proceedings of the 2004 IEEE International Conference on Systems, Man, and Cybernetics*, 4 (pp. 3312-3317). The Hague, Netherlands: IEEE.

Huang, G. Q., & Brandon, J. A. (1993). *Cooperating expert systems in mechanical design* (1st). Hoboken, NJ: John Wiley & Sons.

Hunt, K. & Cremer, J. (2002). Refiner: A problem-solving environment for scientific simulator creation, *SIMULATION,* 78(11), 655-680.

Jang, M., & Lee, J. (2000). Genetic algorithm based design of transonic airfoils using Euler equations. In*Collection of Technical Papers--AIAA/ASME/ASCE/ASC Structures, Structural Dynamics and Materials Conference*, 1 (pp. 1396-1404). Atlanta, GA: AIAA.

Jouhaud, J., Sagaut, P., Montagnac, M., & Laurenceau, J. (2007). A surrogate-model based multidisciplinary shape optimization method with application to a 2D subsonic airfoil, *Computers & Fluids*, 36, 520-529.

Kanukolanu, D., Lewis, K. E., & Winer, E. H. (2006). A multidimensional visualization interface to aid in trade-off decisions during the solution of coupled subsystems under uncertainty, *Transactions of the ASME*, 6, 288-299.

Kay, A. C. (1977, September). Microelectronics and the personal computer, *Scientific American*, 237(3), 230-244.

www.manaraa.com

Kay, A. C. (1993). The early history of SmallTalk, *ACM SIGPLAN Notices*, 28(3), 69-95.

Kerrigan, S. L. (2003). A software infrastructure for regulatory information management and compliance assistance. . Stanford University.

Kirby, M. (2000). *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns*. . Hoboken, NJ: Wiley-Interscience.

Kitamura, Y., Kashiwase, M., Fuse, M., & Mizoguchi, R. (2004). Deployment of an ontological framework of functional design knowledge, *Advanced Engineering Informatics*, 18, 115-127.

Kitware (2005). VTK User's Guide, *Kitware.* Retrieved March 31, 2008 from http://www.kitware.com/products/vtkguide.html.

Kriete, A., & Eils, R. (Eds.). (2005). *Computational systems biology* (1). Burlington, MA: Academic Press.

LMS International (2008). Solutions overview, 3D virtual prototype simulation, LMS Virtual Lab, *LMS Engineering Innovation.* Retrieved March 31, 2008, from http://www.lmsintl.com/simulation/lmsvirtuallab.

Luck, M., Griffiths, N., & d'Inverno, M. (1996). From agent theory to agent construction: a case study. In *Proceedings of the ECAI'06 Workshop on Agent Theories, Architectures, and Languages*, 1193 (pp. 49-63). Budapest, Hungary: Springer-Verlag.

Mäkinen, R., Periaux, J., & Toivanen, J. (1999). Multidisciplinary shape optimization in aerodynamics and electromagnetics using genetic algorithms. *International Journal of Numerical Methods in Fluids*, 30(2), 149-159.

Männisto, T., Peltonen, H., & Sulonen, R. (1999). Modelling generic product structures in step, *Computer-Aided Design*, 30(14), 1111-1118.

Martino, T. D., Falcidieno, B., & Haßinger, S. (1998). Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment. *Computer-Aided Design*, 30(6), 437-452.

McCorkle, D. S., Bryden, K. M., & Kirstukas, S. J. (2003). Building a foundation for power plant virtual engineering. In *The 28th International Technical Conference on Coal Utilization & Fuel Systems*. Clearwater, FL.

Meer, P. (1998). Efficient invariant representations, *International Journal of Computer*

المنارة للاستشارات
www.manaraa.com

Vision, 26(2), 137-152.

Merrill, D., & Maes, P. (2005). Invisible Media: Attention-sensitive informational augmentation for physical objects. In *Proceedings of the Seventh International Conference on Ubiquitous Computing*. Tokyo, Japan.

Messner, J. I., Sanvido, V. E., & Ikeda, M. (1994). Developing an object based planning system for precast concrete building structures. In *Computing in Civil Engineering*, 1 (pp. 1426-1429). Washington DC: ASCE Publications.

Metz, C. (2001, September). The perfect architecture, *PC Magazine*. Retrieved February 19, 2008, from http://www.pcmag.com/article2/0,2817,32905,00.asp.

Modelica Association (2008). Modelica and the Modelica Association, *Modelica.* Retrieved March 31, 2008, from http://www.modelica.org/.

Mohammadi, B., & Pironneau, O. (2002). Applied optimal shape design. *Journal of Computational and Applied Mathematics*, 149, 193-205.

Mohammadi, B., & Pironneau, O. (2001). *Applied shape optimization for fluids*. Oxford, UK: Oxford University Press.

Montgomery, C. J., Swensen, D. A., Harding, T. V., Cremer, M. A., & Bockelie, M. J. (2002). A computational problem solving environment for creating and testing reduced chemical kinetic mechanisms, *Advances in Engineering Software*, 33, 59-70.

Object Management Group, Inc. (2008). CORBA Basics, *Object Management Group.* Retrieved March 31, 2008, from http://www.omg.org/gettingstarted/corbafaq.htm.

Object Management Group, Inc. (2008). OMG Systems Modeling Language: The Official OMG SysML site, Object Management Group. Retrieved March 31, 2008, from http://www.omgsysml.org/.

Ong, Y. S., & Keane, A. J. (2002). A domain knowledge based search adviser for design problem solving environments. *Engineering Application of Artificial Intelligence*, 15, 105-116.

Open CASCADE (2008). Open CASCADE Technology, 3D modeling & numerical simulation, *Open CASCADE Technology.*. Retrieved March 31, 2008, from http://www.opencascade.org/.

OpenDX.org (2006). OpenDX: The Open Source Software Project Based on IBM’s Visualization Data Explorer, *OpenDX.* Retrieved March 31, 2008 from

www.manaraa.com

http://www.opendx.org.

OSG Community (2007). Welcome to the OpenSceneGraph website, *OpenSceneGraph.* Retrieved March 31, 2008 from http://www.openscenegraph.org/projects/osg.

Padula, S. L., & Gillian, R. E. (2006). Multidisciplinary environments: A history of engineering framework development. AIAA 2006-7083. Portsmouth, VA: AIAA.

Pahng, F., Senin, N., & Wallace, D. (1997). Modeling and evaluation of product design problems in a distributed design environment. In *Proceedings of the 1997 ASME Design Engineering Technical Conferences*, DETC97/DFM-4356. Sacramento, CA: ASME.

Pahng, G. F., Bae, S., & Wallace, D. (1998). Web-based collaborative design modeling and decision support. In *Proceedings of DETC '98*. Atlanta, GA: ASME.

Papamichael, K., Chauvet, H., LaPorta, J., & Dandridge, R. (1999). Product modeling for computer-aided decision-making, *Automation in Construction*, 8, 339-350.

Papamichael, K., LaPorta, J., & Chauvet, H. (1997). Building Design Advisor: Automated integration of multiple simulation tools, *Automation in Construction*, 6, 341-352.

Peak, R. S. (2002). *Part 1: Overview of the constrained object (cob) engineering knowledge representation*. Atlanta, GA: Georgia Institute of Technology, Manufacturing Research Center.

Phoenix Integration, Inc. (2008). Products, *Phoenix Integration.* Retrieved March 24, 2008, from http://www.phoenix-int.com/products/modelcenter.php.

Pidd, M. (1992). Object orientation & three phase simulation. In *Proceedings of the 1992 Winter Simulation Conference* (pp. 689-693). Arlington, VA: ACM.

Poloni, C., Giurgevich, A., Onesti, L., & Pediroda, V. (2000). Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 186, 403-420.

Pons, M. (2003). Industrial implementations of the cape-open standard, *AIDIC Conference Series*, 6, 253-262.

Pro-Trax Off-Road Adventures (2008). Home Page, *Pro-Trax Off-Road Adventures.* Retrieved March 31, 2008 from http://www.protrax.co.uk.

www.manaraa.com

Pushpendran, M. (2006, September). A constrained object approach to systems biology. State University of New York at Buffalo.

Quagliarella, D., & Vicini, A. (2001). Viscous single and multicomponent airfoil design with genetic algorithms. *Finite Elements in Analysis and Design*, 37, 365-380.

Qureshi, S. M. (1997, May). Integration framework for design information of electromechanical systems. Arizona State University.

Reed, J. A., & Afjeh, A. A. (1994). Development of an interactive graphical propulsion system simulator. In *Proceedings of the 30th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA 94-3216. Indianapolis, IN: AIAA.

Reed, J. A. (1998, December). Onyx: An object-oriented framework for computational simulation of gas turbine systems. The University of Toledo.

Reed, J. A., & Afjeh, A. A. (2000). Computational simulation of gas turbines, Part 1--Foundations of component-based models, *ASME Journal of Turbomachinery*, 122, 366-376.

Reed, J. A., Follen, G. J., & Afjeh, A. A. (2000). Improving the aircraft design process using web-based modeling and simulation, *Modeling and Computer Simulation*, 10(1), 58-83.

Reichard, G., & Papamichael, K. (2005). Decision-making through performance simulation and code compliance from the early schematic phases of building design, *Automation in Construction*, 14, 173-180.

Ross, D. T., Goodenough, J. B., & Irvine, C. A. (1975). Software Engineering: Processes, Principles, and Goals. *Computer*, 8(5), 17-27.

Rosse, C., & Mejino, J. L. V. (2003). A reference ontology for biomedical informatics: The foundational model of anatomy. *Journal of Biomedical Informatics*, 36(6), 478-500.

Rothenberg, J. (1986). Object-oriented simulation: Where do we go from here? In *Proceedings of the 18th Conference on Winter Simulation* (pp. 464-469). Washington DC: ACM Press.

Sampath, R., Kolonay, R. M., & Kuhne, C. (2002). 2D/3D CFD design optimization using the federated intelligent product environment (FIPER) technology. In *Proceedings*

www.manaraa.com

of the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA-2002-5479. Atlanta, GA: AIAA.

Schmit, T. S., Dhingra, A. K., Landis, F., & Kojasoy, G. (1996). Genetic algorithm optimization technique for compact high intensity cooler design. *Journal of Enhanced Heat Transfer*, 3, 281-290.

SCI Institute (2008). SCIRun, *Scientific Computing and Imaging Institute.* Retrieved March 31, 2008, from http://software.sci.utah.edu/scirun.html.

Senin, N., Wallace, D., Borland, N., & Jakiela, M. J. (1999). *Distributed modeling and optimization of mixed variable design problems*. Boston, MA: MIT CADlab.

Senin, N., Wallace, D. R., & Borland, N. (1999). Object-based design modeling and optimization with genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 12 (pp. 1715-1722). Orlando, FL: Morgan Kaufmann.

Senin, N., Wallace, D. R., & Borland, N. (2003a). Distributed object-based modeling in design simulation marketplace, *Transactions of the ASME*, 125, 2-13.

Senin, N., Wallace, D. R., & Borland, N. (2003b). Distributed object-based modeling in design simulation marketplace, *Journal of Mechanical Design*, 125, 2-13.

Sharpe, J. E. E., & Bracewell, R. H. (2000). Handling complexity in object based modeling and simulation. In *IEE Seminar on Tools for Simulation and Modelling*, 2000/043 (pp. 1/1-1/4). London, England: IEEE.

Shoch, J. F. (1979). An overview of the programming language Smalltalk, *ACM SIGPLAN Notices*, 14(9), 64-73.

Siek, J. G., Lee, L., & Lumsdaine, A. (2001). *The Boost Graph Library: User Guide and Reference Manual*. , C++ In-Depth Series. Indianapolis, Indiana: Addison-Wesley Professional.

Simpson, J. (2004). *An introduction to 3D knowledge objects*. Vancouver, British Columbia, Canada: NGRAIN Corporation.

Skov, M. B., & Stage, J. (2002). Designing interactive narrative systems: is object-orientation useful?, *Computers & Graphics*, 26, 57-66.

Smith, D. A., Kay, A., Raab, A., & Reed, D. P. (2003). Croquet--a collaboration system

www.manaraa.com

architecture. In *Proceedings of the First Conference on Creating, Connecting and Collaborating through Computing* (pp. 2-11). Kyoto, Japan: IEEE.

Soanes, C. & Stevenson, A. (2005). *Concise Oxford English Dictionary.* New York: Oxford University Press, USA.

Stroustrup, B. (1993). A history of c++: 1979-1991, *ACM SIGPLAN Notices*, 28(3), 271-297.

Tambay, P. Y. (2003, October). Constrained objects for modeling complex systems. University of New York at Buffalo.

The MathWorks, Inc. (2008). Simulink, *The MathWorks: Accelerating the pace of engineering and science.* Retrieved March 31, 2008 from http://www.mathworks.com/products/simulink.

Torstenfelt, B., & Klarbring, A. (2007). Conceptual optimal design of modular car product families using simultaneous size, shape and topology optimization, *Finite Elements in Analysis and Design*, 43, 1050-1061.

Toye, G., Cutkosky, M. R., Leifer, L. J., Tenenbaum, J. M., & Glicksman, J. (1994). SHARE: A methodology and environment for collaborative product development, *International Journal of Intelligent & Cooperative Information Systems*, 3, 129-153.

Trigg, M. A., Tubby, G. R., & Sheard, A. G. (1999). Automatic genetic optimization approach to two dimensional blade profile design for steam turbines. *Transactions of the ASME: Journal of Turbomachinery*, 121, 11-17.

Verbaeck, A. (2004). Component-based distributed simulations: the way forward? In *18th Workshop on Parallel and Distributed Simulation* (pp. 141-148). Kufstein, Austria: IEEE.

VeSuite.org (2008). VE-Suite, *VeSuite.org*. Retrieved March 31, 2008 from http://www.vesuite.org.

VRJuggler (2007). VRJuggler, *VRJuggler.* Retrieved March 31, 2008 from http://www.vrjuggler.org.

W3C (2008). W3C Semantic Web Activity, *W3C Semantic Web.* Retrieved March 31, 2008 from http://www.w3.org/2001/sw/.

W3C (2007). XML Schema, *W3C Architecture Domain*. Retrieved March 31, 2008, from

www.manaraa.com

http://www.w3.org/XML/Schema.

W3C (2008). The extensible stylesheet language family (XSL), *W3C Semantic Web.* Retrieved April 5, 2008 from http://www.w3.org/Style/XSL/.

W3C (1999). XSL transformations (XSLT), version 1.0: W3C recommendation 16 November 1999, *W3C Semantic Web.* Retrieved April 5, 2008 from http://www.w3.org/TR/xslt.

Wallace, D., Yang, E., & Senin, N. (2001). Integrated simulation and design synthesis, in distributed object-based modeling environment (DOME), *DSpace at MIT*. Retrieved from http://hdl.handle.net/1721.1/3802.

Wang, C. (1993, September). An approach to managing manufacturing exceptions using object-oriented information integration. Georgia Institute of Technology.

Weerawarana, S., Houstis, E. N., Rice, J. R., Joshi, A., & Houstis, C. E. (1996). PYTHIA: A knowledge-based system to select scientific algorithms, *ACM Transactions on Mathematical Software*, 22(4), 447-468.

Wilson, M. W. (2000). The constrained object (COB) representation for engineering analysis integration. Georgia Institute of Technology.

Wilson, M. W., Peak, R. S., & Fulton, R. E. (2001). Enhancing engineering design and analysis interoperability, Part 1: Constrained objects. In *Proceedings of the First MIT Conference Computational Fluid and Structural Mechanics*. Boston, MA: Elsevier

Winsberg, E. (2003). Simulated experiments: Methodology for a virtual world. *Philosophy of Science*, 70(1), 105-125.

Wladawsky-Berger, I. (2006, October 23). Information - It's All Around Us. *Irving Wladawsky-Berger*. Retrieved March 24, 2008, from http://blog.irvingwb.com/blog/2006/10/information_its.html.

Wong, A., & Sriram, D. (1993). SHARED: An information model for cooperative product development, *Research in Engineering Design*, 5, 21-39.

Wujek, B., Koch, P. N., & Chiang, W. (2000). A workflow paradigm for flexible design process configuration in FIPER. In *Proceedings of the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, AIAA-2000-4868. Long Beach, CA: AIAA.

www.manaraa.com

WxWidgets (2008). What is wxWidgets? *wxWidgets: Cross-platform GUI library.* Retrieved March 31, 2008 from http://www/wxwidgets.org.

Xue, D., & Xu, Y. (2003). Web-based distributed system and database modeling for concurrent design, *Computer-Aided Design*, 35, 433-452.

Young, R. M., & Riedl, M. (2003). Towards an architecture for intelligent control of narrative in interactive virtual worlds. In*Proceedings of the International Conference on Intelligent User Interfaces*. Miami, FL: ACM.

Zha, G., Smith, D., Schwabacher, M., Rasheed, K., & Doyle, A. G. (1997). High performance supersonic missile inlet design using automated optimization. *Journal of Aircraft*, 34(6), 697-705.

Zha, X. F. (2000). An object-oriented knowledge based Petri net approach to intelligent integration of design and assembly planning, *Artificial Intelligence in Engineering*, 14, 83-112.

Zitney, S. Onsite Research: Advanced Process Simulation. *National Energy Technology Laboratory*. Retrieved March 24, 2008, from http://www.netl.doe.gov/onsite_research/Facilities/apecs.html.

www.manaraa.com