
Masters Theses

Student Theses and Dissertations

Spring 2011

Integration of model-based systems engineering and virtual engineering tools for detailed design

Akshay Kande

Follow this and additional works at: https://scholarsmine.mst.edu/masters_theses



Part of the [Systems Engineering Commons](#)

Department:

Recommended Citation

Kande, Akshay, "Integration of model-based systems engineering and virtual engineering tools for detailed design" (2011). *Masters Theses*. 5155.

https://scholarsmine.mst.edu/masters_theses/5155

This thesis is brought to you by Scholars' Mine, a service of the Missouri S&T Library and Learning Resources. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

INTEGRATION OF MODEL-BASED SYSTEMS ENGINEERING AND VIRTUAL
ENGINEERING TOOLS FOR DETAILED DESIGN

by

AKSHAY KANDE

A THESIS

Presented to the Faculty of the Graduate School of the
MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

2011

Approved by

Steve Corns, Advisor
Cihan Dagli
Scott Grasman

© 2011

Akshay Kande

All Rights Reserved

ABSTRACT

Design and development of a system can be viewed as a process of transferring and transforming data using a set of tools that form the system's development environment. Conversion of the systems engineering data into useful information is one of the prime objectives of the tools used in the process. With complex systems, the objective is further augmented with a need to represent the information in an accessible and comprehensible manner. The importance of representation is easily understood in light of the fact that the stakeholder's ability to make prompt and appropriate decisions is directly related to his understanding of the available information. Systems Modeling Language (SysML), a graphical modeling language developed by Object Management Group is one such tool used to capture and convey information about a system under development. This work proposes a methodology for integrating the models developed using SysML with virtual engineering software to create an executable, interactive, and user-centered platform for engineering systems. The framework provides an opportunity to combine the benefits offered by both model-based systems engineering and virtual engineering for detail design. This research demonstrates how this framework can be implemented using a biotech fermentor to illustrate the coupling of information between SysML and virtual engineering.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Steve Corns, for his valuable leadership and advice during my graduate studies. I have sincerely enjoyed working on his research projects. I would also like to thank Dr. Cihan Dagli and Dr. Scott Grasman for their valuable input and participation in my committee.

I would like to thank my parents, Shekhar and Shilpa Kande, for their love and commitment to make me what I am today. I would also like to thank my grandmother, Vijaya Kande, for her blessings and support.

I would like to thank my roommates, Juned Kazi, Ravi Bhatt, Balaji, and Vinayak Bhagwat, for their unconditional support and encouragement throughout my graduate studies. I would also like to thank a very close friend of mine, Shashank Babele, whose willingness to help me when I needed it the most was truly commendable.

TABLE OF CONTENTS

	Page
ABSTRACT.....	iii
ACKNOWLEDGMENTS	iv
LIST OF ILLUSTRATIONS.....	vii
LIST OF TABLES.....	viii
SECTION	
1. INTRODUCTION.....	1
2. COMPLEXITY AND SYSTEMS ENGINEERING	4
2.1. DEFINITION.....	4
2.2. BACKGROUND	5
3. MODEL-BASED SYSTEMS ENGINEERING	8
3.1. BACKGROUND	8
3.2. SYSTEMS MODELING LANGUAGE.....	11
3.2.1. SysML Blocks..	13
3.2.2. SysML Structural Diagrams..	13
3.2.3. Profiles and Stereotypes..	14
3.3. PREVIOUS WORK ON EXECUTABLE MODELS	14
4. VIRTUAL ENVIRONMENT	15
4.1. BACKGROUND	15
4.2. VIRTUAL ENGINEERING.....	16
4.3. VIRTUAL ENGINEERING TOOL	16
5. VIRTUAL SYSTEMS MODELING APPROACH.....	19
5.1. OVERVIEW	19
5.2. METHODOLOGY	20
5.3. C++ PROFILE	25
5.4. VIRTUAL ENGINEERING MODELS IN AN MBSE ENVIRONMENT	26
6. EXAMPLE MODEL DEVELOPMENT	28
6.1. SYSTEM OVERVIEW	28

6.2. MODEL DEVELOPMENT USING SYSML	29
6.3. DEFINING AN ANALYTICAL MODEL IN SYSML	30
6.4. ADDING VE MODELS	33
7. EXPERIMENTAL RESULTS	35
8. DISCUSSION AND CONCLUSION	40
APPENDICES	
A. OUTPUT CODE	43
B. GUIDELINES	63
BIBLIOGRAPHY	66
VITA	70

LIST OF ILLUSTRATIONS

Figure	Page
2.1 V-model of Systems Engineering	6
2.2 Document-centric Method	7
3.1 OMG's Model-driven Architecture	9
3.2 Elements of an MBSE Environment.....	10
3.3 SysML Diagram Types	11
3.4 Four Pillars of SysML.....	12
4.1 CAVE Setup.....	15
4.2 VE-Suite Architecture.....	17
5.1 Virtual Systems Modeling Approach.....	19
5.2 VE-Suite Model Structure	20
5.3 Analytical Model Structure in SysML.....	21
5.4 Analogy Between Two Model Structures.....	22
5.5 Modeling Methodology	23
5.6 C++ Constructs	25
5.7 VE-UI Module Using SysML Block	26
6.1 Fermentor System Layout.....	29
6.2 Block Definition Diagram of the Fermentor System	30
6.3 Constraints Used in FermentorAnalysis	31
6.4 BDD Showing Fermentor Analysis Model.....	32
6.5 Parametric Diagram Relating Structural and Simulation Properties	33
6.6 Fermentor Analysis Model Structure with VES Modules	34
7.1 Fermentor Analysis Model in VE-Suite	39

LIST OF TABLES

Table	Page
5.1 SysML Relationship Constructs from [1]	24

1. INTRODUCTION

Systems engineering is a process of transforming a set of customer needs into an integrated package of high performing solutions. Since its inception, systems engineering has found applications in solving complex engineering problems. Potential solutions to these problems can involve any combination of hardware, software, data, people, and facilities [1]. The manner in which these elements are selected and combined determines the effectiveness of proposed solutions.

Due to the amount of information generated and processed in systems engineering projects, a key aspect of efficient systems engineering is the management of data and information throughout the development process. Any errors introduced in managing this information would have a direct impact on the value of the end product. With complex systems, the challenge is even greater due to the increased interconnections between subsystems.

Traditionally, information management and exchange in systems engineering has relied on document-centric methods. The underlying principle of such an approach is to use documents as a means of communication between stakeholders of the system. Information concerning every exercise in the systems development process is documented for later use. However, a major drawback of such methods is that managing a large amount of information stored separately in individual files becomes cumbersome. The focus of the process shifts from effective systems engineering towards maintaining the validity and consistency of these documents. The pressure to build systems with increased performance capabilities at a reduced cost and in less time has led researchers to find new ways to aid the systems engineering practice.

Model-based systems engineering is one discipline developed to overcome the limitations of these document-centric methods. A system model developed using this approach mimics the requirements, structural, and, behavioral aspects of the system in an integrated and consistent manner. Use of models in developing systems has the potential to provide an effective means for handling information present in the development environment. Axelsson [3] identifies the necessity of having a modeling language to satisfy general systems engineering needs and proposes an approach to extend the

capabilities of existing object-oriented language. He extended unified modeling language (UML), largely affiliated with software-centric systems, to apply to general purpose systems. Addressing the increasing demand for a modeling language for general systems engineering applications, the Object Management Group (OMG) introduced systems modeling language (SysML) [25, 38]. SysML is an extension to UML that provides a framework to capture and represent information about a system under development. The objective is to use an object-oriented methodology to model systems, subsystems, and their components, along with interrelationships among them.

Although SysML has proven useful in developing formal descriptions of systems, it is not an interactive design tool. A connection with a separate simulation package is required in order to perform engineering analysis using SysML models. This paper addresses this limitation by creating a framework to integrate model-based and virtual engineering disciplines. The work presented here exemplifies the use of virtual engineering in the form of an open source package called VE-Suite [39] in conjunction with SysML to link formal system models to executable engineering models. The use of virtual engineering technology eliminates the need to link the system model with various analytical tools. This is made possible by the fact that virtual engineering technology itself provides the potential to integrate and combine geometric models, analysis, simulation, optimization, and other decision making tools in a single environment [5]. Once the information from a system model is passed on to this framework, VE-Suite has the capability to work with numerous types of analysis packages in addition to executing user-defined computational units. The methodology presented here has the potential to combine the capabilities of model-based systems engineering (MBSE) to effectively manage information complexity and the executable aspects of virtual engineering tools, thus creating an integrated modeling environment. The goal of this research is to demonstrate the capability of generating a computer-based virtual environment to develop, analyze, and optimize a complete system that is formally described using SysML. Using a virtual engineering tool in conjunction with SysML models would allow system architects to develop systems in an interactive and executable design environment.

The organization of the thesis is as follows: Section 2 discusses complexity in systems engineering projects and provides an overview of techniques introduced previously to manage it. Section 3 gives a detailed introduction to model-based systems engineering, describing its history and development. It also provides an overview of SysML. Section 4 provides information about the virtual engineering domain and the software tool used in this research. Section 5 presents the approach taken here to integrate model-based and virtual engineering technologies. Section 6 tests this methodology with an example of a model. Section 7 presents the results of this model integration approach. Finally, the thesis concludes with a discussion of some important aspects of the use of this framework and provides a template for future users to develop their own executable models.

2. COMPLEXITY AND SYSTEMS ENGINEERING

This Section introduces some important concepts relevant to the work discussed later on in this thesis. It defines complexity as it is used in this work, outlines the history of systems engineering, and describes its applicability to complex engineering ventures. It also provides an overview of the traditional document-based approach to systems engineering, describing the limitations of that approach. Finally, it introduces the current model-based systems engineering methodology.

2.1. DEFINITION

Historically, many researchers have attempted to define complexity based on their own perspectives and the context in which they apply it. Edmund [6] defines complexity as “that property of a language expression which makes it difficult to formulate its overall behavior even when given almost complete information about its atomic components and their interrelations.” (page # 6) This definition however, holds true only when there is a possibility of finding significant amount of information about the components of a system [7]. Gershenson and Heylighen [35] offer a simpler interpretation, which states that “in order to have a complex, you need two or more distinct components that are connected in such a way that they are difficult to separate.” (page # 2) Again, the limitation of this definition is that an apple-to-apple comparison is not possible to measure the overall complexity. These definitions and their limitations demonstrate that there is no single accepted definition of complexity. The vastness of the concept and its applicability restricts a general consensus among different authors when they define complexity.

At this point a brief discussion of complexity as it will be treated in this work is in order, to define techniques that can be used to manage it. The definition of complexity used in this research must be understood in the context of its application to a general purpose system. Here, complexity is the property by which behavior of elements becomes interconnected in such a way that changes made have effects beyond a local area. Thus, in complex systems, behavior is governed by the complexity that exists within the architecture of the system itself. The art of building such complex engineering ventures is

known as systems engineering. According to the NASA Handbook [8], “Systems engineering is a holistic, integrative discipline, wherein the contributions of structural engineers, electrical engineers, mechanism designers, power designers, human factors engineers and many more disciplines are evaluated and balanced, one against another, to produce a coherent whole that is not dominated by the perspective of a single discipline.” (page # 21) It is a logical way of handling today’s challenging engineering ventures involving highly interconnected subsystems.

2.2. BACKGROUND

From its beginning systems engineering has addressed complexity. The demand for building systems with increased performance capabilities at reduced cost and time spurred the need to have a streamlined way of managing complexity that these systems necessitate. Systematic design approaches such as those described in [9, 10, and 11] have addressed this need. Figure 2.1 illustrates a typical systems development process. The first part of the V-model represents the decomposition and definition phases. The remainder comprises system integration and realization activities. Thus, the model begins with the initial set of customer requirements and eventually transforms those requirements into a complete system description.

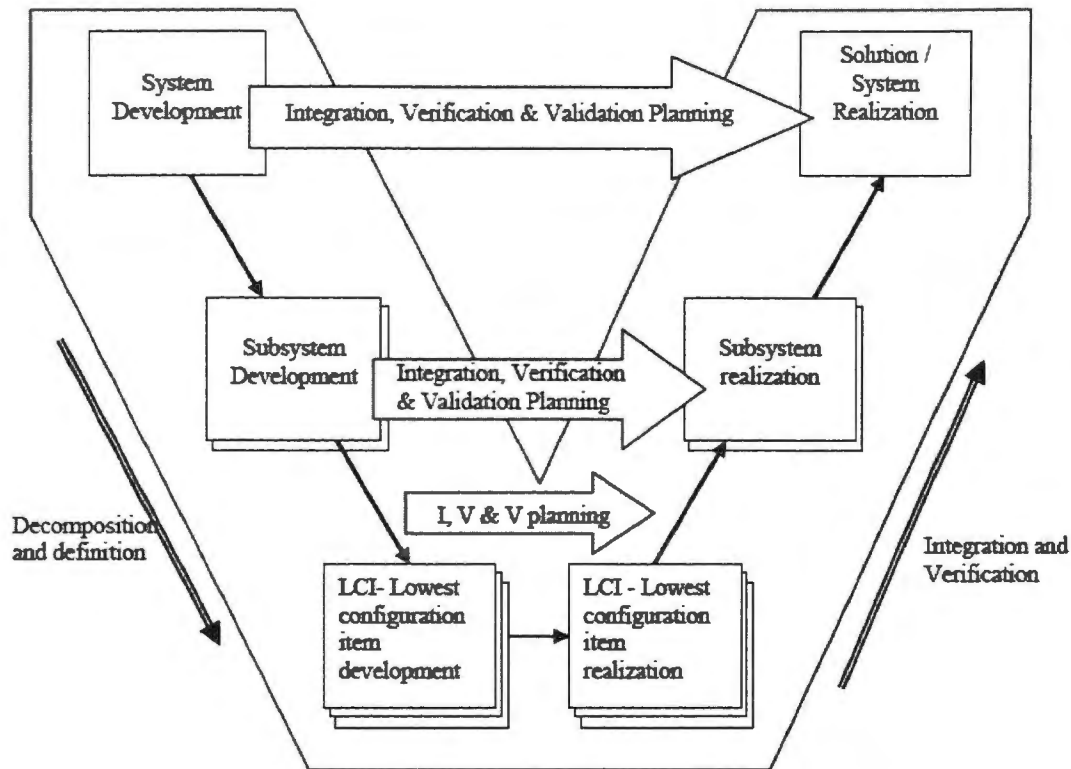


Figure 2.1 V-model of Systems Engineering [12]

A careful study of the above process indicates that the success of the effort to meet stakeholder needs depends on the information flow through the development process. In other words, managing the information flow effectively is one of the key drivers in satisfying end customer needs. Traditionally, this information management has been done using document-centric methods that store information on every step of the process in separate files. Thus, information from requirements to technical specifications to the detail design is documented individually in relevant file formats. Figure 2.2 shows the traditional approach, each step of which has inputs and outputs in the form of documents.

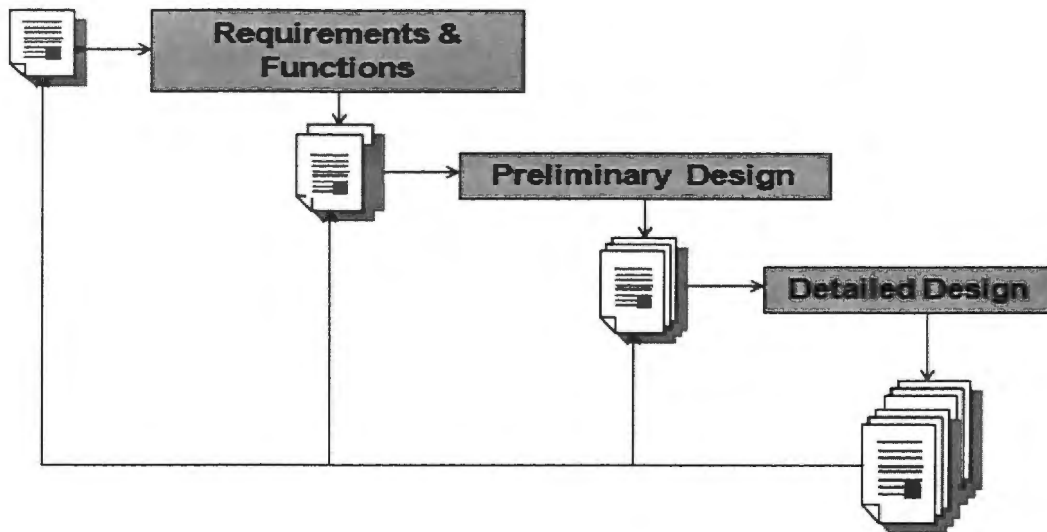


Figure 2.2 Document-centric Method

The document-centric technique has been adequate for many systems engineering projects in the past. However, it has certain limitations. First, maintaining consistency among documents becomes cumbersome, especially when change is the only thing that remains constant in a development process. Additionally, extracting useful information from the pile of documents becomes tedious and time consuming. Finally, today's systems have become increasingly complex. The nature of systems engineering challenges has changed significantly [13], warranting for a new approach that more easily accommodates the increased demands of today's systems.

One way of dealing with the information complexity in a systems development environment is by the use of computer-aided modeling. Model-based systems engineering is one of such technique. The International Council on Systems Engineering (INCOSE) [37] defines MBSE as “the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later lifecycle phases.” MBSE promotes the use of computer-aided modeling approaches to develop models that represent structural, behavioral, and operational characteristics of a system in development. A detailed discussion of MBSE follows in Section 3.

3. MODEL-BASED SYSTEMS ENGINEERING

This Section explains the model-based approach introduced in Section 2. It discusses OMG's SysML and explains some important constructs of that language. This section also reviews some previous work that has also used SysML and describes the approach adapted here to overcome some of its limitations.

3.1. BACKGROUND

Model-driven development in systems engineering can be traced back to the mid-1990's when researchers first began to identify applications of object-oriented techniques to support the systems engineering process. Friedenthal and Lykins [2] document the use of parameter-based representation to define system attributes and their relationships with the help of object-oriented constructs for modeling complex systems. Lykins, Friedenthal, and Meilich [14] trace the evolution of the object-oriented systems engineering method (OOSEM), which uses OMG's Unified Modeling Language (UML) to capture system-level requirements and design information. Addressing the interoperability issue of platform dependent models in software systems, OMG introduced model-driven architecture (MDA). In MDA, platform-independent models (PIMs) are initially defined using a modeling language. These models are then translated to platform specific models (PSM) using transformations [44]. Figure 3.1 below describes MDA and its elements.

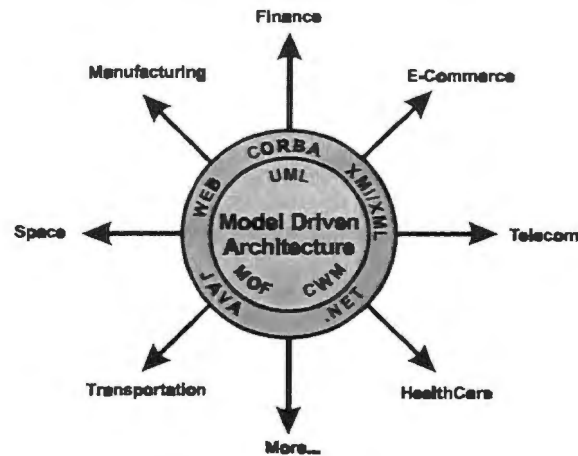


Figure 3.1 OMG's Model-driven Architecture [45]

At the core of the architecture lie the platform independent modeling standards: UML, meta-object facility (MOF) and common warehouse meta-model. The target platforms form the outer thin layer followed by the application areas.

The initial success of OOSEM and MDA in software-intensive systems prompted its application to general purpose systems engineering projects involving software, hardware, people, and other entities. Studies by Bahill and Daniels [15] and Hsu and McDonough [16] were early attempts to apply OOSEM to general systems. Model-based systems engineering offers a more formalized way to use object-oriented principles to solve complex systems engineering problems. Estafen's work [17] identifies several model-based systems engineering methodologies catering to the requirements of modern and highly complex systems engineering projects. Arthurs [18] studied the vital components of an MBSE environment and the relationships among them. His work identifies three core elements of an MBSE environment: modeling language, modeling tools and the modeling process. Figure 3.2 illustrates these MBSE elements and their interrelationships.

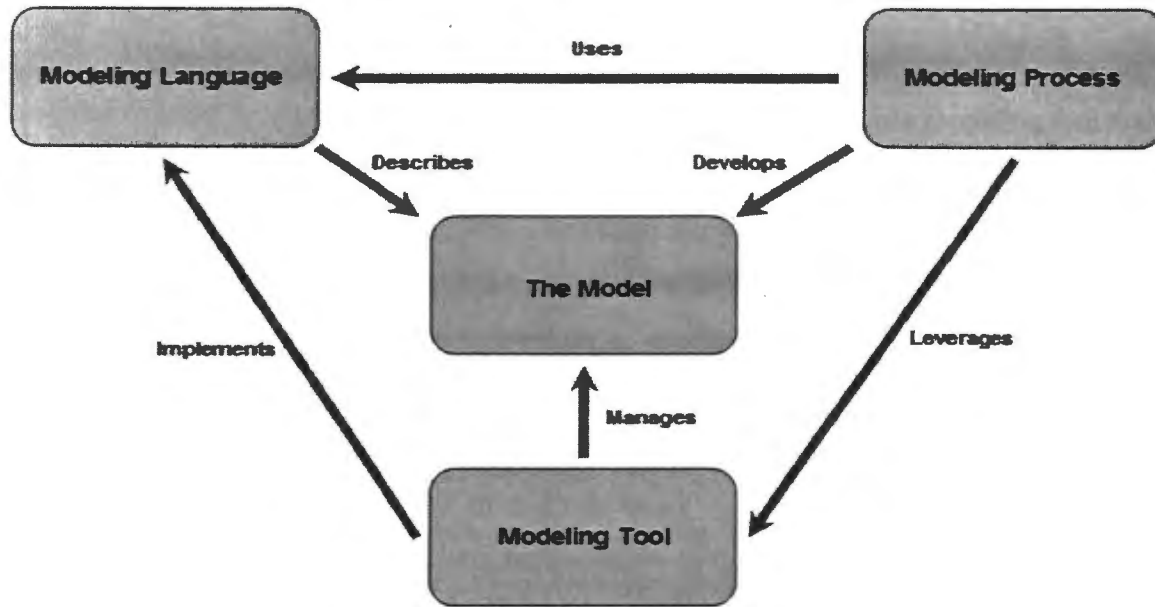


Figure 3.2 Elements of an MBSE Environment [18]

A brief explanation of the three core elements is as follows:

- **Modeling language:** Provides the semantics to define the constructs for modeling systems characteristics in an MBSE environment
- **Modeling tool:** Provides the means to implement the modeling language and the interface to develop the system model
- **Modeling process:** This is specific to the users of an MBSE tool. Requirements can either be defined in an MBSE tool and assigned to functions before building a physical architecture, or traced directly to the physical architecture

Currently, several modeling languages are available for developing system models in an MBSE environment; these include OMG's SysML, along with object process methodology (OPM) and OMG's unified profile for Department of Defense Architecture Framework/Ministry of Defense Architecture Framework (UPDM) for architecture and system-of-systems modeling. The present work uses SysML. Detailed information about the other two languages can be found in [19, 20, 21, 22, and 23]. A detailed comparison between SysML and OPM is documented by Grobshtein [24].

3.2. SYSTEMS MODELING LANGUAGE

SysML is a graphical modeling language used to specify, analyze, design, and verify complex systems [25]. It is an extension of UML, which is a modeling tool that has been used extensively in the software industry to manage complex software engineering projects. OMG introduced SysML in cooperation with the International Council on Systems Engineering to support the implementation of model-based systems engineering. The language is based on object-oriented principles with a semantic foundation for creating models of physical systems using well defined visual constructs. As an extension of UML, it has inherited various properties of the parent language. Additionally, many new features have been added in order to make it suitable for the systems engineering domain. For example, it supports both requirements modeling and parametric modeling (to develop mathematical and engineering models). Figure 3.3 shows SysML diagram taxonomy, including diagrams inherited from UML and those that represent new additions.

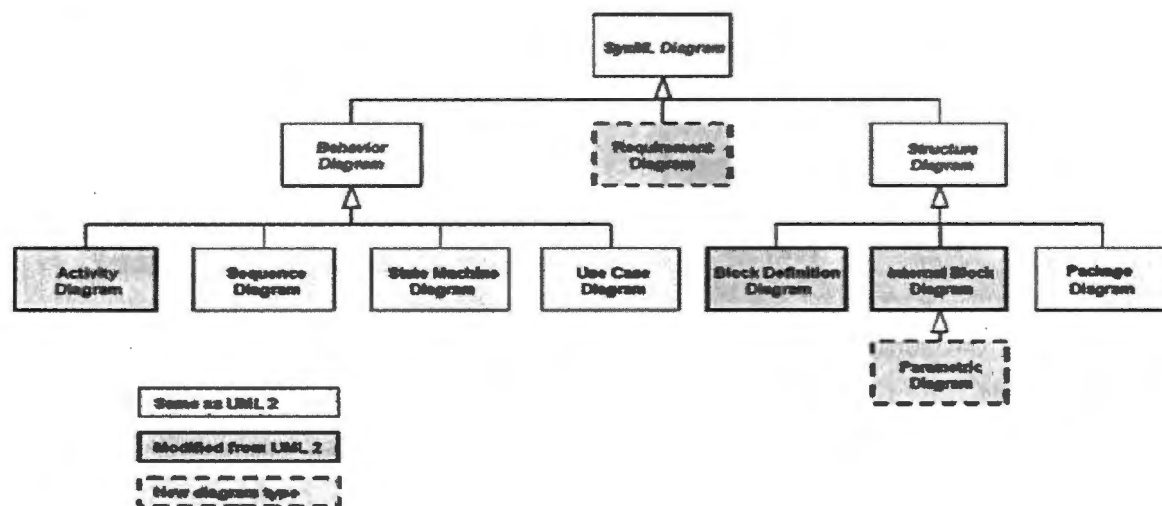


Figure 3.3 SysML Diagram Types [39]

Each diagram type offered by SysML corresponds to activities to be performed in a typical systems development process. All the diagram types are organized under one of three major categories: structure, behavior, and the newly added requirements. This allows the development of a system model beginning with a requirements diagram to specify system requirements followed by behavior and structure diagrams to further detail the system. Figure 3.4 shows a screenshot of sample SysML diagrams developed in an MBSE tool.

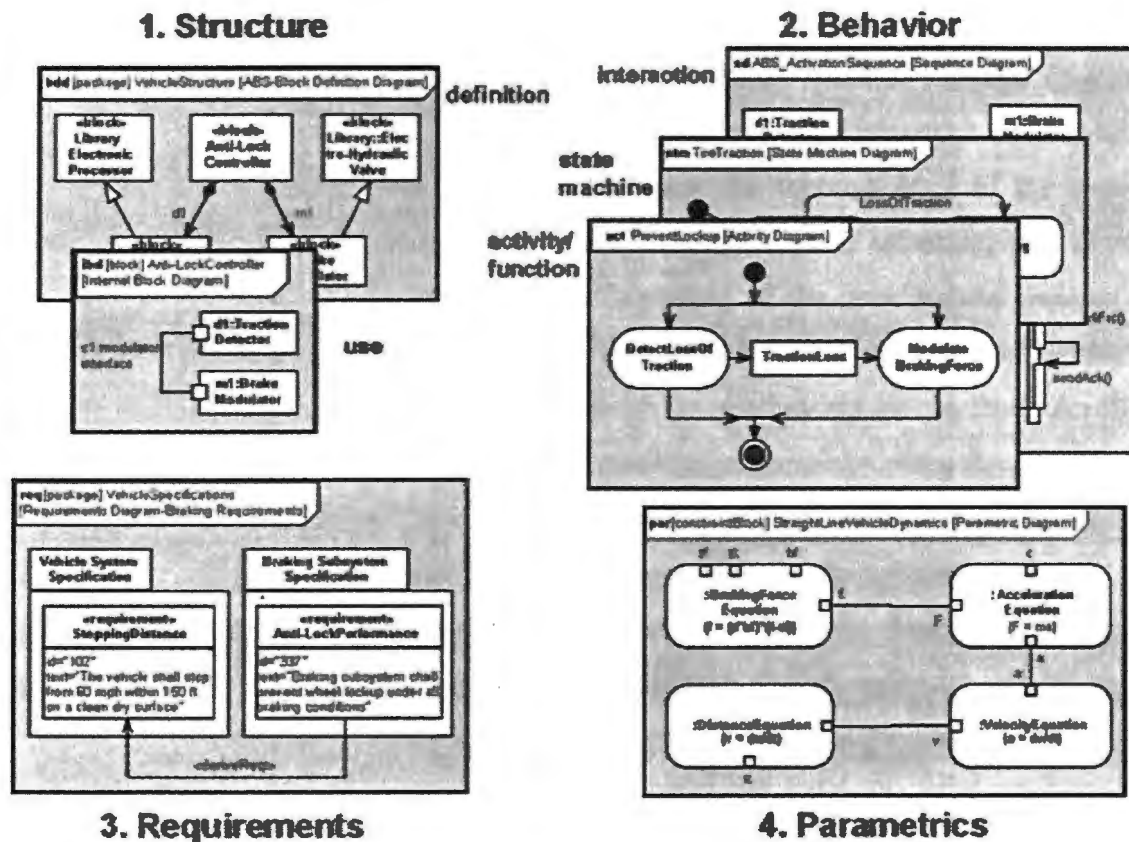


Figure 3.4 Four Pillars of SysML [39]

Since majority of the work reported here is based on model development in SysML, the following is an overview of the basic SysML constructs relevant to the example model described in Section 6.

3.2.1. SysML Blocks. The block is the primary modeling unit in SysML. It is an extension of a UML class, and it represents elements of the system whether hardware, software, personnel, facilities, or some other entity. Blocks provide a means to describe the system features in the form of reusable components. Additionally, a collection of blocks permits decomposition of a system into several layers of detail.

3.2.2. SysML Structural Diagrams. SysML offers three major structural constructs: the package diagram, the block definition diagram (BDD), and the internal block diagram (IBD). These diagrams provide an interface to model the physical architecture of the system and the relationships among subsystems. In SysML, a package groups a large collection of data pertinent to a particular domain. Package diagrams provide an interface to group these elements in a hierarchical structure and establish relationships among them. They typically represent the topmost level of the system model. Examples of domains include system behavior, system structure, and system requirements defined by the modeler at the beginning of the development process. A BDD describes the architectural hierarchy of a system. It is defined by grouping blocks representing elements of the system and establishing relationship among them. An IBD describes the internal structure of a block by defining relationships using flow properties and connectors. In addition to these constructs, SysML permits creation of an engineering model that uses constraint blocks to specify constraints on the system, subsystem, or component properties. A collection of these blocks along with their relationships is represented using the parametric diagram. In addition, SysML provides valuetypes to capture the quantifiable characteristics of each element of the engineering model under development.

3.2.3. Profiles and Stereotypes. OMG indicates that the extension of the metamodel to add domain specific information is possible using profile packages. For example, SysML is a profile created to suit the systems engineering domain on top of the UML metamodel thus using and extending the capabilities of the original profile. Similarly, SysML profile can be further customized to suit certain user-specific requirements. Stereotypes are the primary mechanism for creating these profiles. A combination of three different types of profiles: SysML, UML, and C++ are used while creating the example model presented in Section 6. The model based systems engineering software used in this work is Artisan Studio [36]. A detailed reading about SysML and its semantics can be found in [25].

3.3. PREVIOUS WORK ON EXECUTABLE MODELS

SysML has attracted the attention of the systems engineering community ever since its introduction. One of the main areas researchers have sought to develop techniques for is to overcome its non-executable nature. Johnson [27 and 28] integrates SysML and Modelica constructs using triple graph grammars to generate executable, continuous, dynamic model output. Huang, Ramamurthy, and McGinnis [4] demonstrate the use of SysML as a formal modeling language to create conceptual models and transform them into Arena simulation language constructs for execution. Peak et al. [29 and 30] document the use of SysML parametrics for engineering design and analysis and demonstrate its support for simulation-based design. The work presented here introduces virtual engineering domain as a companion to MBSE in order to execute engineering models in an interactive design environment. Such a model integration framework would not only extend the capabilities of SysML and MBSE, but also clarify systems performance characteristics and thus promote informed decision making. Section 4 explains the virtual engineering domain and the tool used in this research.

4. VIRTUAL ENVIRONMENT

This Section provides an overview of the virtual engineering technology used in this research in conjunction with the MBSE tool. It offers a brief explanation of the virtual engineering software along with an insight into the architecture of this software.

4.1. BACKGROUND

Howard Rheingold [42 and 46] defines virtual reality (VR) as an experience in which a person is “surrounded by a three dimensional computer-generated representation, and is able to move around in the virtual world and see it from different angles, to reach into it, grab it, and reshape it.”(page # 2) In other words, VR is a computer simulation that uses three-dimensional graphics to create a virtual environment and gives user the feeling of being immersed in that environment. The level of immersion experienced by the user depends upon the type of hardware used. For example, a desktop virtual environment uses stereo-enabled graphics cards, shutter glasses, and a three-dimensional mouse to create the sense of immersion and allow the user to interact with the environment. Single- and multiple-wall virtual environments provide a larger display, creating a high level of immersion and a more natural interface. Figure 4.1 below shows a setup of the CAVE Automatic Virtual Environment with four walls.

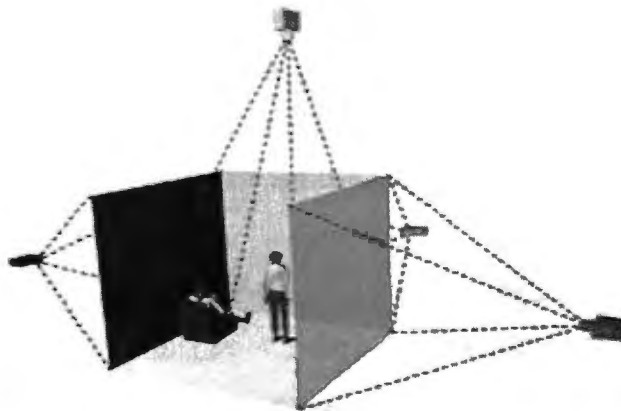


Figure 4.1 CAVE Setup

4.2. VIRTUAL ENGINEERING

The engineering community soon recognized the interactive capability of a virtual workspace, and subsequent efforts to exploit its potential led to the foundation of virtual engineering as a discipline. Virtual engineering provides a means of creating a replica of a physical system in a computer-generated virtual environment. The model underlying such an environment can be a combination of geometric, physical, qualitative, and quantitative data associated with the system [33]. The objective of virtual engineering is to allow designers to observe how a system reacts to changes in design and operation without the need to create a physical prototype. In addition, it provides an accessible visual format for the presentation of information that is of value to all stakeholders. The combination of geometric models and a variety of decision support tools creates an environment in which engineers can make appropriate decisions by interacting with the system naturally and exploring details that might otherwise remain undetected. With the increased demand for building complex systems, virtual engineering can be an effective way of building, operating, and testing such a system in an integrated design environment with a user-centered perspective.

4.3. VIRTUAL ENGINEERING TOOL

This research uses VE-Suite, an open-source software package developed at Iowa State University to facilitate virtual engineering. VE-Suite addresses the need to have a common platform for performing the engineering process by providing open interfaces that allows software packages to exchange data in a comprehensive design environment [40]. Thus, engineers can work in a single environment that accommodates information from various software tools, and they can interact with engineering models to create a virtual decision-making environment.

The architecture of VE-Suite is composed of three core engines: VE_Xplorer, VE_CE, and VE_Conductor. Figure 4.2 shows the architecture of VE_Suite. A brief description of the architectural components is provided below.

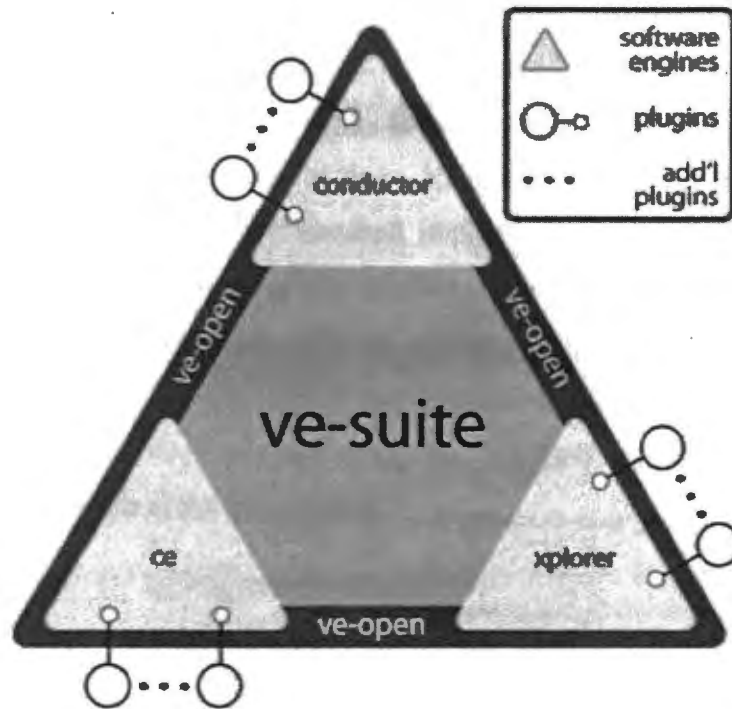


Figure 4.2 VE-Suite Architecture

VE_Conductor forms the front end user interface. It uses open-source cross-platform user interface (UI) libraries offered by Wxwidgets [41]. Depending on the user's requirements they can create an interface using the base classes provided with the VE-Suite Application Programming Interface (API).

VE_CE is the computational engine that handles the coordination, scheduling and monitoring of the simulation runs and provides a means of linking commercial analysis software packages with the virtual engineering framework. Thus, the communication with other software packages to send and receive data is handled by the computational engine. Additionally, users can define their own computational unit and attach it to the service by creating digital library files to create these plugins.

VE_Xplorer is the graphical engine that is responsible for creating the virtual environment. It provides the visual interface to display the simulation under observation. Additionally, physical architecture prototypes can be created using the OpenSceneGraph libraries which form the underlying layer of this engine. Once we have the prototype, Xplorer can be connected with an immersive virtual reality environment like the CAVE

using VRJuggler as the device manager. Thus, a stakeholder can decide the level of interactivity he/she wants and switch from a desktop-based display to a highly immersive CAVE environment without having to worry about the compatibility issue.

In addition to the above core engines, VE-Suite uses Common Object Request Broker Architecture (CORBA) [38] standard implemented in VE-Open to establish a communication medium between its core engines. This also provides the capability to operate engines independently from different locations. A typical VE-Suite model is a combination of three plugins: UI plugin, Computational unit plugin and graphical plugin. Users can build their custom plugins using the VE-Suite application programming interface (API) comprised of the base classes.

5. VIRTUAL SYSTEMS MODELING APPROACH

This Section explains the model integration approach proposed in this thesis. It also discusses the model transformation procedure used to integrate SysML and virtual engineering models.

5.1. OVERVIEW

The present work seeks to integrate virtual engineering models with system models described using SysML. The MBSE tool used here work is Artisan Studio. Figure 5.1 outlines the model integration approach; where we have system formal model developed using MBSE methods on one side and an executable virtual engineering model on another. Once the information from a system model is passed on to the virtual engineering framework, analytical models of varying types and levels of fidelity can be executed.

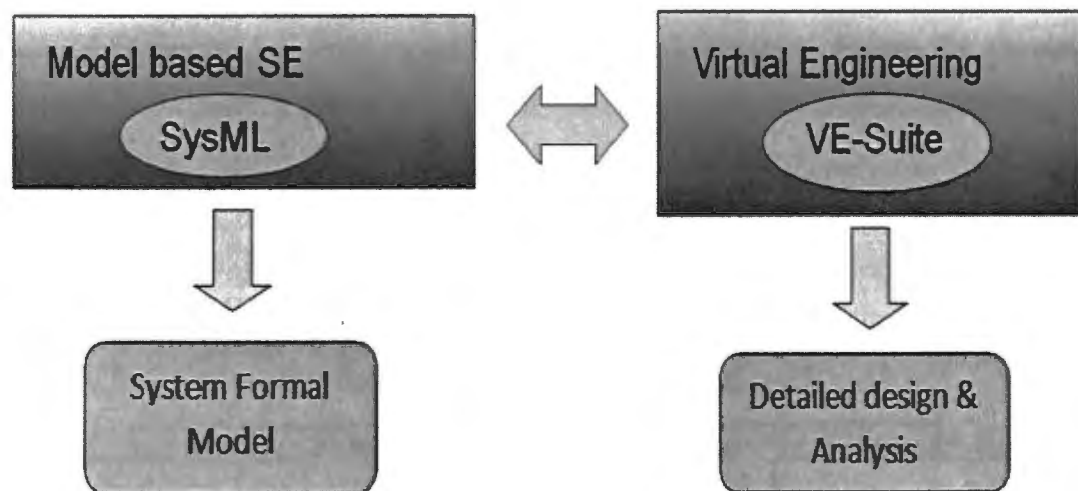


Figure 5.1 Virtual Systems Modeling Approach

The use of virtual engineering in conjunction with SysML demonstrates the value of this approach for meeting the following objectives:

- Execute analysis models developed using SysML in an MBSE tool.
- Provide a visual interface that clarifies the system and its operations in a better way.
- To create an appropriate decision making environment.
- To identify potential effects of changes in design parameters on system performance.
- To maintain consistency in information flow by creating an integrated design environment.

A single environment capable of meeting these objectives would allow engineers to concentrate on systems development rather than focusing on the tool interfaces and transformations necessary to gather the required information.

5.2. METHODOLOGY

As explained in section 4.2, a typical virtual engineering model designed in VE-Suite is composed of three components: the VE-UI (User interface that displays the design parameters considered in a simulation and acts as the data feeding point), VE-Unit (Backend computational unit), VE-GP (the graphical Plugin to display the simulation results and CAD data). Figure 5.2 shows the VE-Suite model structure composition.

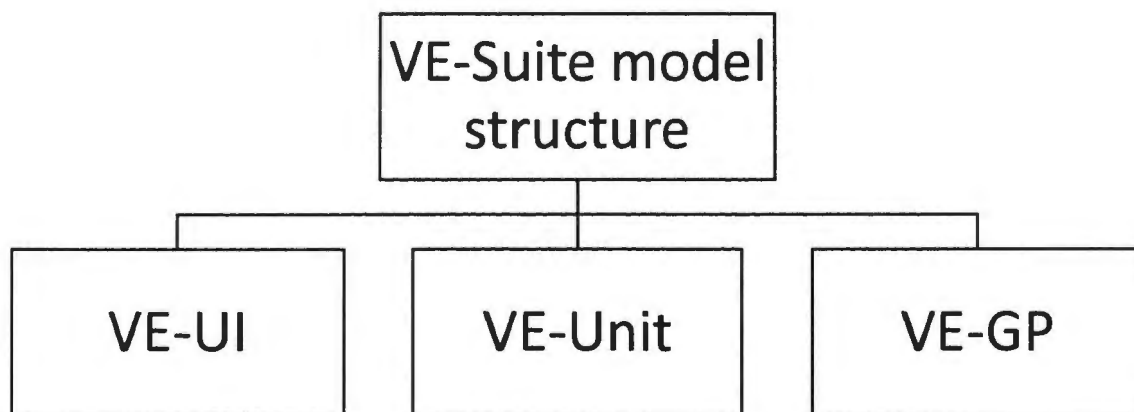


Figure 5.2 VE-Suite Model Structure

Similarly, an analysis model in SysML can be viewed as being composed of three essential components: 1) the simulation block, which defines the input/output parameters and time constraints to control simulation start and stop times, 2) the constraint block, which defines the constraints to be applied to the design parameters in the simulation, and 3) the system model, which defines the structure and value properties to be analyzed. Figure 5.3 shows the structure of the analytical model.

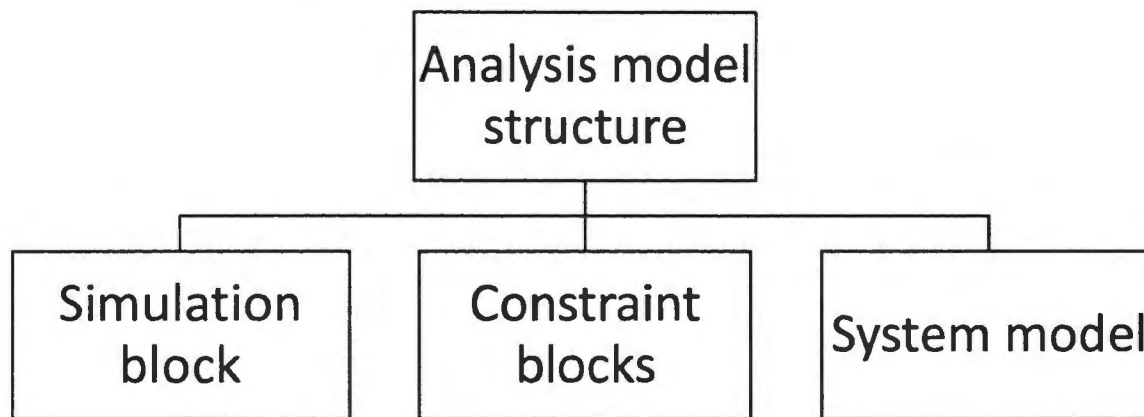


Figure 5.3 Analytical Model Structure in SysML

When using this type of model structures, it becomes clear that the two model compositions can be thought of as being analogous to each other. Figure 5.4 shows the analogy between the two model structures. Thus, models from the MBSE domain can be the source of the information on which executable virtual models are built.

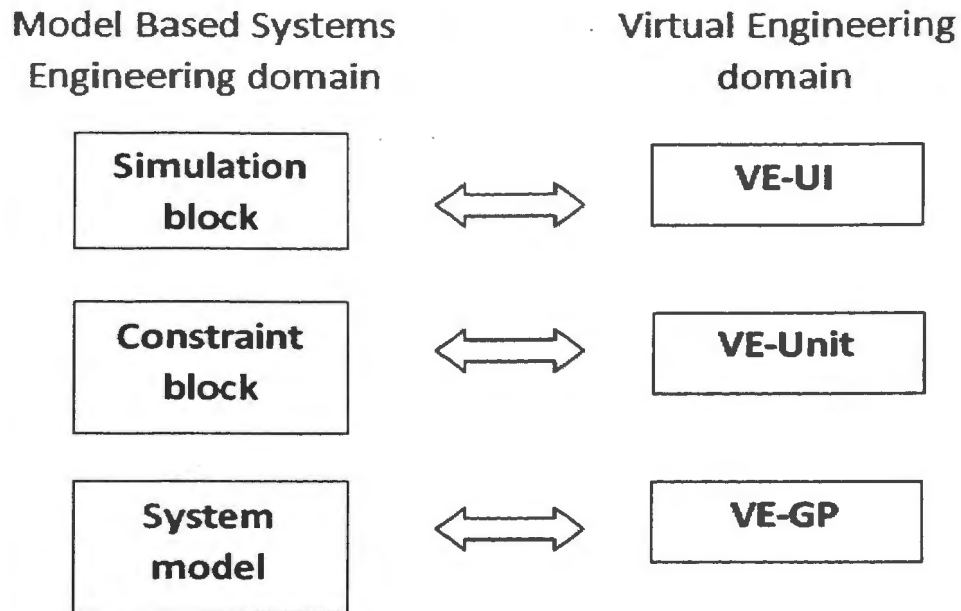


Figure 5.4 Analogy Between Two Model Structures

Another important consideration is that VE-Suite models are C++ based. This gives increased flexibility in the types of information that can be represented, but to create executable virtual engineering models there has to be a mechanism for information exchange between SysML and C++ based VE-Suite models. A combination of SysML, UML and C++ profiles provides such a mechanism in an MBSE environment. Figure 5.5 illustrates the modeling methodology.

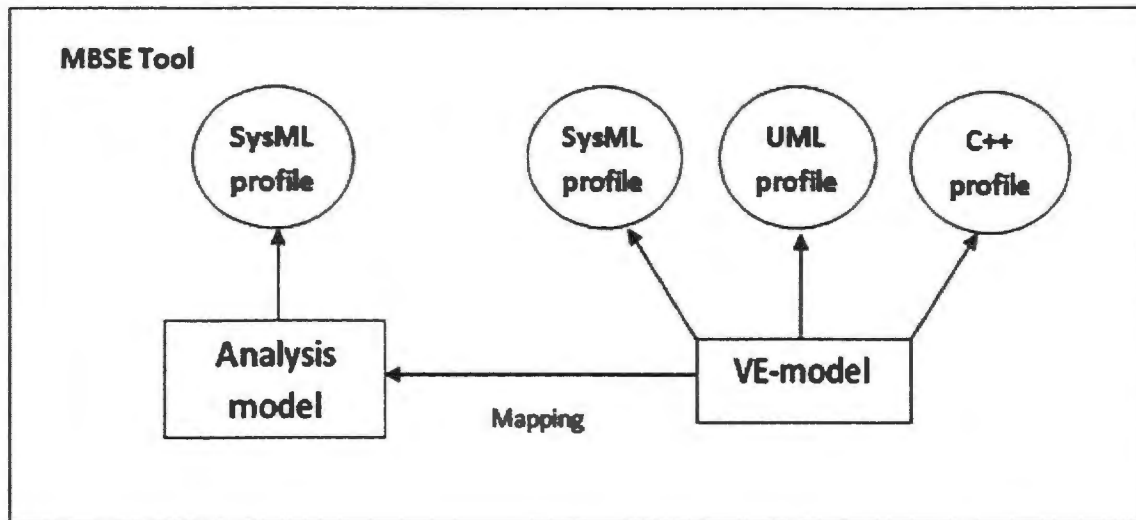




Figure 5.5 Modeling Methodology

The connection between the two types of models is done using relationship constructs offered by SysML. Table 5.1 below gives a detailed description of composite, reference, and generalization relationship constructs used in this work.

Table 5.1 SysML Relationship Constructs from [1]

Diagram Element	Notation	Description
Composite Association Path	<p>The notation for Composite Association Path includes two main types of associations. The first type has a diamond at the whole end and an arrow at the part end, with multiplicity labels at both ends. The second type has an arrow at the whole end and a diamond at the part end, also with multiplicity labels. Below these are two tree diagrams illustrating how these associations branch into multiple paths.</p>	<p>A composite association relates a whole to its parts showing the relative multiplicity at both whole and part ends. A composite association always defines a part property in the whole (indicated by <code><Part></code>). Where there is no arrow on the nondiamond end of the association it also specifies a reference property to the whole in the part (indicated by <code><Reference></code>).</p> <p>Otherwise when there is an arrow, the name at the whole end simply gives a name to the association end (indicated by <code><End></code>).</p>
Reference Association Path	<p>The notation for Reference Association Path includes two main types of associations. The first type has a diamond at the whole end and an arrow at the part end, with multiplicity labels at both ends. The second type has an arrow at the whole end and a diamond at the part end, also with multiplicity labels. Below these are two tree diagrams illustrating how these associations branch into multiple paths.</p>	<p>A reference association can be used to specify a relationship between two blocks. A reference association can specify a reference property on the blocks at one or both ends. The white diamond is the same as no diamond, but profiles can be used to differentiate them by specifying additional constraints.</p>

Table 5.1 SysML Relationship Constructs from [1] (Continued)

Generalization Path	 	<p>A generalization describes the relationship between the general classifier and specialized classifier. A set of generalizations may either be {disjoint} or {overlapping}. They may also be {complete} or {incomplete}.</p>
---------------------	--	--

5.3. C++ PROFILE

The C++ profile package supplied by the MBSE tool infuses the model with information specific to C++. It contains stereotypes and tag Definitions required to model C++ code that can be used in combination with SysML and UML profiles. Figure 5.6 shows a screen shot of the template of the C++ profile elements provided in Artisan Studio.

Tag Definition Name	Tag Value
C++ Header Include	
C++ Implementation Include	
C++ Inheritance List	
C++ Specialization Parameters	
C++ Non Member	FALSE
C++ Forward Declaration Text	
C++ Forward Declarations	
C++ Linkage	

Figure 5.6 C++ Constructs

By applying C++ stereotypes to objects defined using SysML, information specific to C++ can be entered using tag values. Thus, a SysML block can contain SysML properties such as values, parts, and constraints, as well as tag values defined in the C++ stereotype applied to it.

5.4. VIRTUAL ENGINEERING MODELS IN AN MBSE ENVIRONMENT

Virtual engineering models in an MBSE tool can be represented using SysML blocks. Each VE block in the tool corresponds to the VE-Suite model structure described in section 5.2. These blocks are composed of the sets of operations necessary to create models compatible with VE-Suite using the information supplied by the corresponding SysML analytical model. Figure 5.7 shows a sample VE-Suite UI module to create the user interface plugin defined using SysML block along with its primary operations.

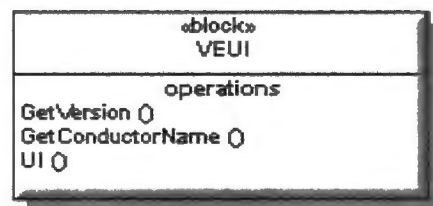


Figure 5.7 VE-UI Module Using SysML Block

Once all the VE-Suite modules are created, relationship constructs are used to establish a connection between the two models. Information about the design parameters and time constraints is supplied by the simulation block to the corresponding VE-UI block. The latter contains the operations to create a user interface for the simulation. The constraint information used in the analysis is passed from one module to another by creating a relationship between the constraint block and VE-Unit block, which stores the operations to create the backend computational unit. Information about the computer-aided design (CAD) models associated with structural units of the system can be passed on to VE-GP using the reference association.

Plugin dll files required to run the application in VE-Suite can then be created by compiling the auto generated C++ code in an integrated development environment (IDE). The model organization created in the MBSE tool makes the code readily available for compilation to create the plugins for VE-Suite.

6. EXAMPLE MODEL DEVELOPMENT

This Section provides a proof of concept for the methodology introduced in Section 5. It introduces a fermentor system and develops an example model in the MBSE environment. The sections of this Section outline the steps to build a model using the virtual systems modeling approach.

6.1. SYSTEM OVERVIEW

An example model of a fermentor system demonstrates the feasibility of integrating the SysML model into a virtual engineering environment. Bio-processing industry uses fermentor systems to produce compounds like ethanol and citric acid. A virtual engineering model of the fermentor has already been created as part of a training project at Indian Hills Community College Bioprocess Training Center. The purpose of this model was to train students and operators to understand the effects of varying chemical and biological inputs on the fermentation process. This fermentor system provides a baseline sufficiently broad to display the utility of the method proposed here, but still small enough to permit rapid modifications to evaluate the performance of our method. A typical fermentation process requires a set of inputs including both physical compounds and controlled working conditions. Production is governed by the chemical reactions that result from these input sets. Figure 6.1 shows a conventional fermentor system to demonstrate the composition and internal interfaces of the system.

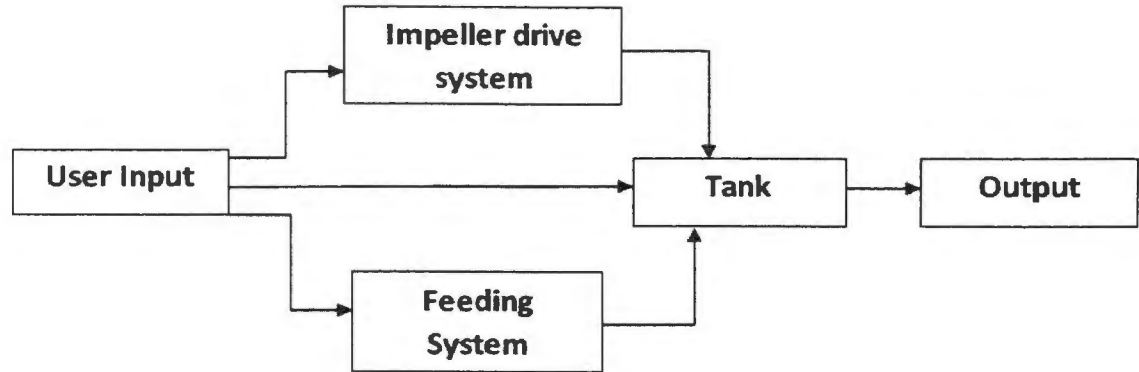


Figure 6.1 Fermentor System Layout

Development of the fermentor model began with the identification of input and output parameters required to create a realistic model of the fermentation process. The inputs considered here are impeller speed, initial pH level, initial nitrate concentration, and initial temperature. The output of the system is the concentration of citric acid produced with the given set of input parameters.

6.2. MODEL DEVELOPMENT USING SYSML

With the layout of the system as the baseline, the first step was to create a formal model of the fermentor using SysML constructs. Figure 6.2 shows the fermentor structure, modeled using a block definition diagram offered by SysML.

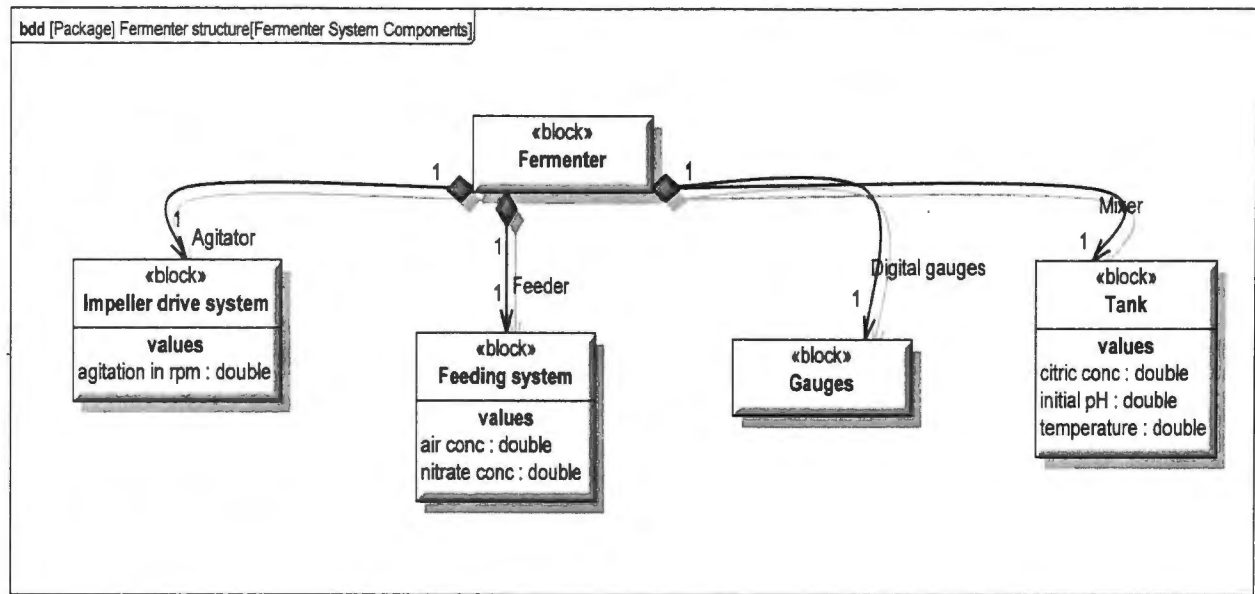


Figure 6.2 Block Definition Diagram of the Fermentor System

The fermentor block shown here is composed of four key elements: the impeller drive system, the feeding system, the gauges, and the tank. Each element has value properties that represent the quantifiable characteristics of the blocks. These quantities also represent the input and output parameters considered in the experiment.

6.3. DEFINING AN ANALYTICAL MODEL IN SYSML

Developing an analytical model requires the knowledge of the model type and the purpose that it is going to satisfy. The purpose here refers to the effective understanding of a particular type of system behavior under controlled input conditions. In the case of the fermentor system, the objective was to observe the behavior of the fermentation process by varying input conditions. An analytical model was developed using SysML diagram types to accommodate the computational information required to run the simulations. The first type of information was the constraints required to calculate the concentration of citric acid. Figure 6.3 shows the constraints used to calculate the concentration, defined using a SysML construct.

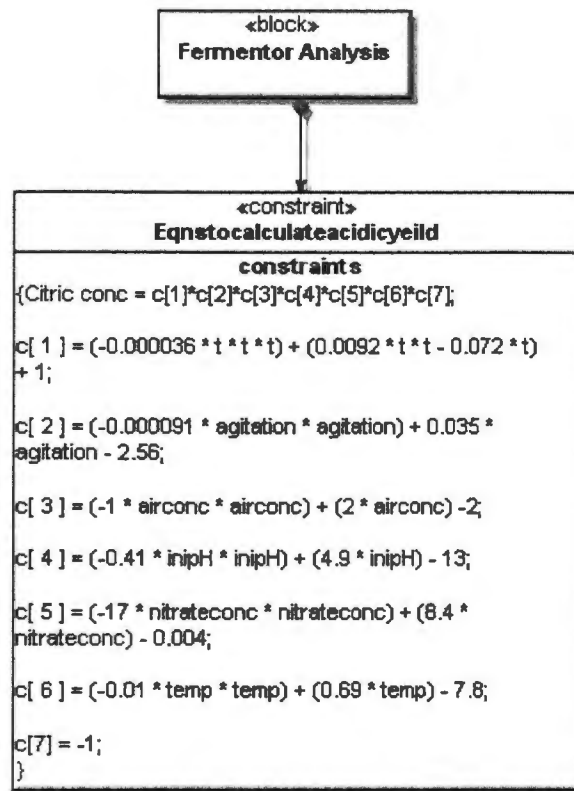


Figure 6.3 Constraints Used in Fermentor Analysis

Once the constraints were defined, the next step was to add the properties of the various subsystems defining the actual system model and the simulation model which is comprised of experiment specific information. The actual systems model is the fermentor model structure described previously; the simulation model includes input and output parameters and time constraints to run the experiment. Figure 6.4 shows the structure of the fermentor analysis model using a BDD of SysML.

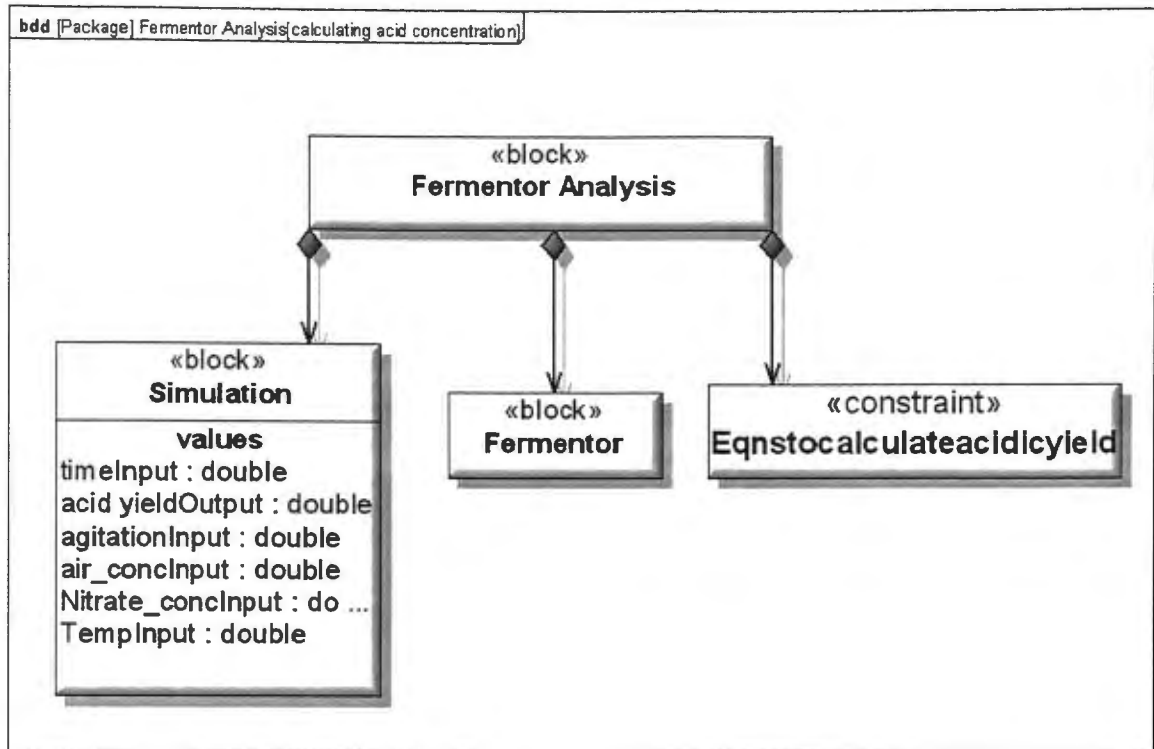


Figure 6.4 BDD Showing Fermentor Analysis Model

Developing a system structural model separately and then introducing it in the analysis context gives the advantages of having a modular design. The analysis can be changed by varying the constraints and simulation properties while keeping the primary system model unaffected. Thus, different analyses involving the same system model properties can be undertaken easily. Value properties defined in the simulation block must be constrained by the value properties of the actual system model in order to maintain consistency throughout the model. Such constraints are imposed by a parametric model that relates value properties of both the blocks using an equality constraint. Figure 6.5 shows the parametric diagram developed to constrain the simulation block properties.

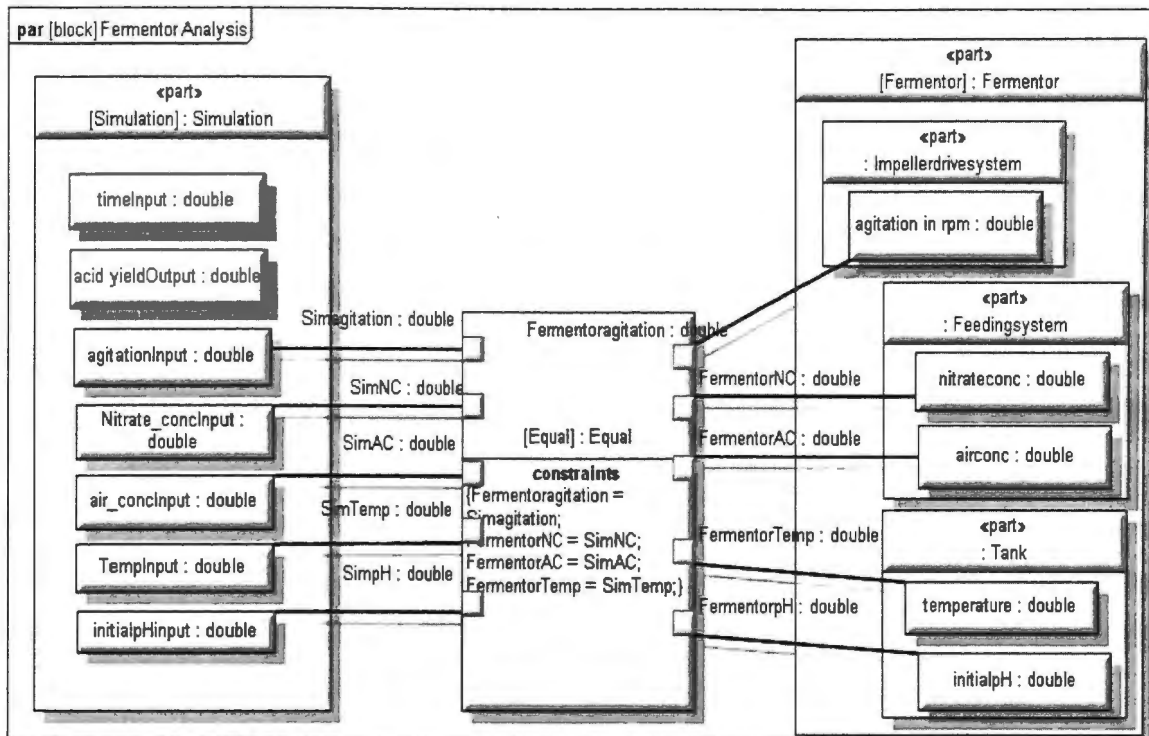


Figure 6.5 Parametric Diagram Relating Structural and Simulation Properties

6.4. ADDING VE MODELS

This section describes the creation of a VE-Suite model in the MBSE tool. First, the three main components representing the VE model are created (see Section 3.2). Each VE-Suite module in the MBSE environment is composed of operations specific to the plugin that it is creating. The FermentorUI block represents the user interface module that will control the simulation; the FermentorUnit represents the backend computational unit, and the FermentorGP stores the operation to create the display. In order for the VE models to have analysis-specific information, relationships are established with the model already developed using SysML. Thus, the FermentorUI inherits the simulation-specific properties from the simulation block, and the FermentorUnit has a reference to the constraint block that stores the operation to calculate the concentration of citric acid.

One of the key advantages of using a virtual engineering model is that information can be presented in a manner readily understood even by a non-expert. This

is due in part because of the use of CAD models to represent a system. Subsystem models can be represented by individual CAD models. VE-Suite's graphical engine provides the capacity to create such a model architecture using OpenSceneGraph libraries. To exploit this option, part properties of the fermentor system model are referenced with individual CAD files that replicate their structure using an UML operation. This operation reads data from the CAD file using an OpenSceneGraph function. Each subsystem associated with a CAD file has a member function to read the CAD data and store it in the form of an OpenSceneGraph node. These part properties are then referenced in the FermentorGP module using relationship semantics. The module has operations to create a complete scene graph composed of individual nodes by calling the member function defined in the part properties using the reference handle. Figure 6.6 shows the organization of the complete fermentor analysis model. It represents VE models in the diagram explicitly using the <<VES>> stereotype.

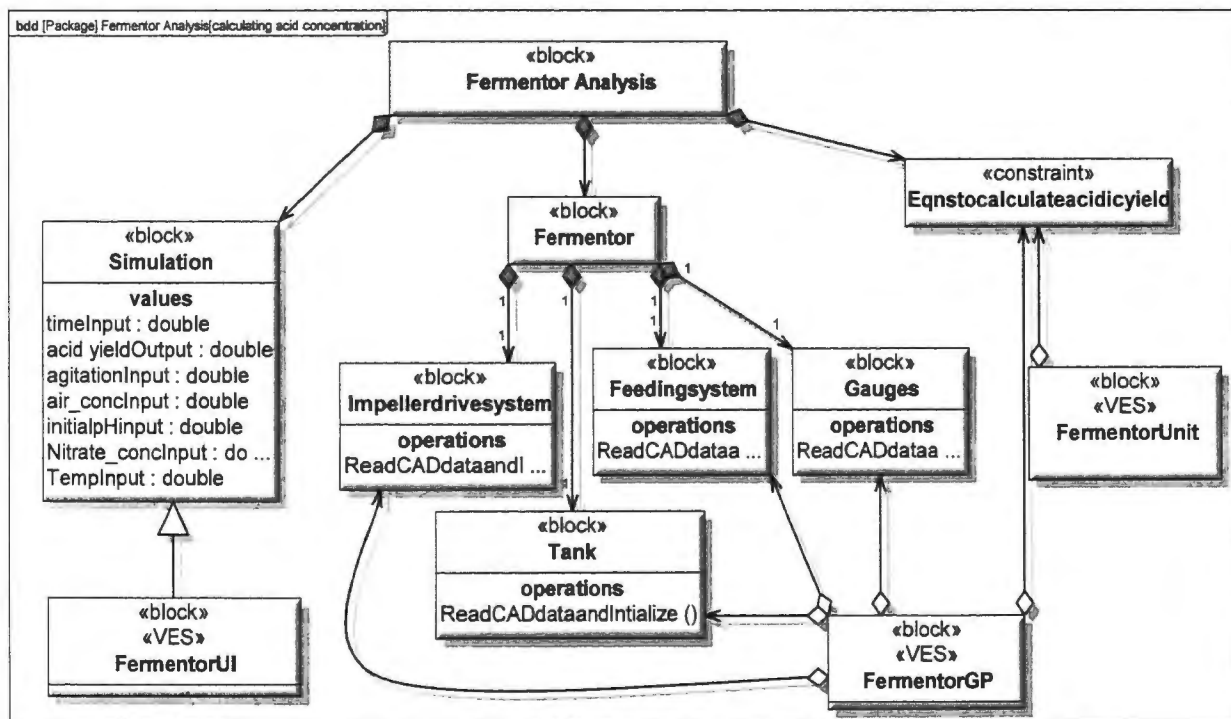


Figure 6.6 Fermentor Analysis Model Structure with VES Modules

7. EXPERIMENTAL RESULTS

This section describes the output from the MBSE tool and its compilation to create the VE-Suite model. The auto code synchronizer (ACS) function in Artisan was used to generate and synchronize the C++ code with the SysML model. The objective here was to convert a model developed from one language (SysML) to another (C++) so that it can take advantage of the execution capabilities provided by the virtual engineering tools. The advantage of using the ACS while generating output is that the code in the IDE (Visual Studio, in this case) remains synchronized with the model at all times. In this way the code and the model can be updated whenever a change is made. In addition, the code generated as an output of this process remains consistent with the relationships defined in the SysML model. For example, the FermentorGP module has references with both the constraint block and the structural components of the fermentor system; therefore, when we output the code for the GP module it will have information access to the related files. The following pseudo code of the FermentorGP module explains how reference pointers provide access to the CAD data defined individually in the structural components and to the constraint information.

```
//FermentorGP.h

.
.
.

class VEFermentorGraphicalPlugin: inheriting from VE-Suite Xplorer base
class
{
public:
    .
    .
    .
// Public|Package operations
    Constructor for the FermentorGP class
    Destructor for the FermentorGP class

//Base class operations

    virtual void InitializeNode( osg::Group* veworldDCS );
    virtual void PreFrameUpdate();
    virtual void ProcessOnSubmitJob();

private:
```

```

        // Private attributes

// Reference pointers

//Pointer to access constraints information

        Eqntocalculateconc* constraint;

//Pointer to access CAD data from the system model

        Impellerdrivesystem* impeller;
        Tank* tank;
        Feedingsystem* feedingsystem;

        .
        .
        .
    };
endif

```

The complete C++ output code is presented in the appendix. Each structural component of the fermentor system has an associated CAD file that is read using OSG functions and stored in a node accessed by the Fermentor GP module. This process creates a tree structure to display all the elements in the Xplorer interface. The following pseudo code represents the feeding system:

```

// File : .\Feedingsystem.cpp

# include <Feeding system header file>

# include <Osg header file to read CAD data>

// Operation implementation of the read CAD data function
osg::ref_ptr< osg::Node > Feedingsystem::ReadCADdata()
{
    // ## OperationBody [11feb10e-328a-4f94-a6c9-473b14683f60]

    //feedingsystem.ive is the CAD file passed as argument to the OSG
    function; output of the function is in the form OSG node

    _fermentorGeometry = osgDB::readNodeFile( "Models/feedingsystem.ive"
    );

    // The return type is in the form of OSG node

```

```
// FermentorGP module has access to this OSG node through the pointers
defined in its header file

return _fermentorGeometry;
// ## OperationBody End

}
```

The .ive file is the CAD file which is read and stored in an OSG node defined as `_fermentorGeometry`. This process is repeated for all other structural components and the OSG nodes are available to the GP module to access.

The simulation block in the fermentor analysis model defines the experimental data elements and their value types. The FermentorUI module inherits this information from the simulation block. Shown below is the pseudo code of the simulation block followed by the FermentorUI module:

```
// File : .\Simulation.h

#ifndef __simulation
#define __simulation
.
.
.
class simulation
{
Public:
    //user defined experimental parameters

    double agitationInput;
    double air_concInput;
    .
    .
    .
    double TempInput
    //All these parameters are defined in the simulation block using
    SysML value properties

};
#endif

// File : .\FermentorUI.h

#ifndef __FermentorUI
#define __FermentorUI

#include <header file to provide access to simulation block properties>

#include <header file of the conductor base class>
```

```

.
.
.
class FermentorUI : inherits from conductor base class and simulation
class
{
    .
    .
public:

    // Public|Package operations

    FermentorUI constructor
    FermentorUI destructor

//ves conductor base class operations

    virtual double GetVersion();
    .
    .
    .
    virtual ves::conductor::UIDialog* UI( wxWindow* parent );

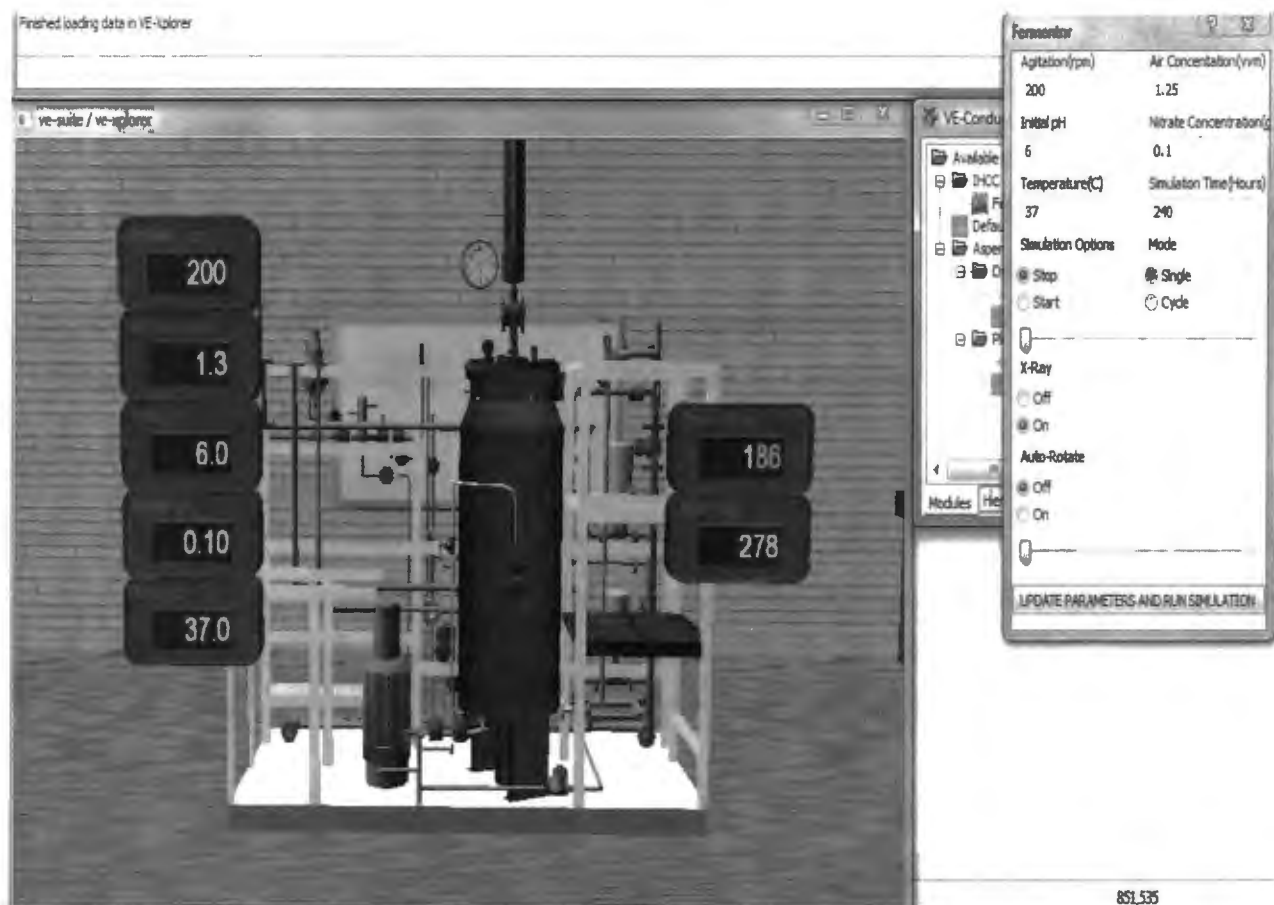
    virtual wxString GetConductorName();

    wxString GetName();
    .
    .
};

#endif

```

Once the output C++ code of the fermentor analysis model has been generated, the next step is to generate the dll files that can be plugged into the virtual engineering software to run the experiment. The code was compiled using SCons [43], a software construction tool used to build the VE-Suite source code. Before compiling, the SConscript files were created with the software environment necessary for VE-Suite. The output of the compilation process was in the form of dll files that VE-Suite can read and execute in its environment. Figure 7.1 shows a screenshot of the fermentor model in VE-Suite as the final output. The user interface can be used to change the design parameters, run the simulation, and view the results in the Xplorer window.



851,535

Figure 7.1 Fermentor Analysis Model in VE-Suite

8. DISCUSSION AND CONCLUSION

The power of using Model based design has been known to the engineering community for quite some time. It provides a scope to enhance the traditionally used relatively inflexible form of doing engineering design by introducing modularity, reusability, and easy maintainability of design information. Model-based systems engineering has been introduced to utilize these opportunities to solve more complex engineering problems confronted while undertaking systems engineering projects. Additionally, the activities involved in designing complex systems demand that different modeling formalisms work together synergistically. This increases the importance of having an effective model integration approach that satisfies multiple requirements. This paper has introduced an approach that integrates formal system models with detailed engineering models using the profiles and relationship constructs offered by an MBSE tool.

As a language supporting MBSE, SysML has gained attention for both the capability it offers and its limitations. It addresses various aspects of systems engineering activity and provides an easy-to-use library of graphical constructs to facilitate those activities. In addition, SysML permits the use of relationships to maintain consistency and coherence in models. However, its lack of self-execution capability forces it to rely on external analytical tools to solve mathematical models. This work proposes the use of virtual engineering as a means of analysis that retains the formal system models developed using SysML as the source of information. By integrating virtual engineering models with SysML, the potential of SysML to manage information complexity can be used in conjunction with executable detailed design models. The example model developed here demonstrates the effectiveness of this approach for:

- creating an executable model of a simulation that is synchronized with the systems engineering information,
- using CAD data to create a realistic view of the system, thus creating an effective decision making environment,

- understanding the effects of changing design parameters on system performance.

The integration of an MBSE tool with the virtual environment brings high-level formal models closer to detailed engineering models so that decisions can be made with the help of interactive engineering analysis. The model integration approach illustrated here also maintains information consistency throughout the process. Thus system models in both the environments remain synchronized so that changes can be tracked and information can be updated dynamically. In addition, the manner in which SysML model is defined allows for extensive model reuse. System structural components can be tested by performing different types of analyses without making major changes to the original model. For example, if the user decides to change the type of analysis for the fermentor from calculating the concentration of citric acid to a different product, they can manipulate the model structure without destroying the original one. Also, the VES modules defined above can be reused to support the new analysis. Once the first model is created and executed, amount of rework required to test the system for a different set of parameters and constraints is much reduced.

The objectives accomplished in this research demonstrate our efforts towards larger integration of software tools for systems engineering management and engineering decision making. The example model demonstrates the creation of a user-defined computational unit, which is one of numerous capabilities that VE-Suite offers. Furthermore, external analytical tools such as CFD and FEA packages can be synchronized with the virtual platform to vary the levels of fidelity. The use of high fidelity models in conjunction with systems engineering information in a virtual environment can provide an interactive, user-centered, and thorough working environment for the stakeholders to work in. In addition, the open interfaces offered by the virtual engineering tool permit for integration of different software tools used in systems development in a single, comprehensive design environment. The work presented here represents the preliminary steps required to use the capabilities of the virtual engineering platform to create an integrated development environment. Also, a template giving basic information on using the methodology for creating engineering models is presented in Appendix B. By providing this template we demonstrate the

breadth of the methodology described in this thesis in developing newer engineering models using a combination of model-based and virtual engineering domains. Thus, future users can use this as a guide in developing executable models relevant to their needs.

The next step in this research will be to use the MBSE tool API and integrate it with the VE-Suite open interface. A linkage of this kind will provide a more seamless integration between the two tools for data exchange and could minimize the amount of manual work required to create the dll files after generating the C++ code. Additionally, the access to the MBSE tool API would allow component traceability between the two domains. Thus, if an engineer decides to change the composition of the system model by replacing a component, the outcome of this decision can be instantaneously visualized in the corresponding virtual environment. This would also help in analyzing the effects of changing requirements on the system architecture as the model traceability will span from requirements to the detailed engineering models.

APPENDIX A
OUTPUT CODE

```

// File : .\FermentorUI.h

#ifndef __FermentorUI
#define __FermentorUI

#include "simulation.h"
#include <ves/conductor/UIPluginBase.h>
#include <wx/image.h>
#include <string>

class FermentorUI : public ves::conductor::UIPluginBase, public
simulation
{
    DECLARE_DYNAMIC_CLASS( Fermentor )

public:

    // Public|Package operations

    FermentorUI();
    virtual ~FermentorUI();

    virtual double GetVersion();

    virtual ves::conductor::UIDialog* UI( wxWindow* parent );

    virtual wxString GetConductorName();

    wxString GetName();

    long cycle_ID;
    long rotation_ID;
    long xray_ID;
    long loop_ID;

    double rot_speed;
    double sim_speed;

};

#endif

```

```

// File : .\FermentorUI.cpp

#include "StdAfx.h"
#include "FermentorUI.h"

```

```

#include "FermentorUIDialog.h"
#include <wx/wx.h>
#include <iostream>
#include <fstream>
#include <wx/wx.h>

IMPLEMENT_DYNAMIC_CLASS( FermentorUI, ves::conductor::UIPluginBase )

// Operation implementations

FermentorUI::FermentorUI()
{
    // ## OperationBody [8e8d3b2c-2lad-4003-8726-73501bafa0e2]
    RegistVar( "agitation", &agitation );
    RegistVar( "air_conc", &air_conc );
    RegistVar( "ini_ph", &ini_ph );
    RegistVar( "nitrate_conc", &nitrate_conc );
    RegistVar( "temperature", &temperature );
    RegistVar( "hours", &hours );
    RegistVar( "cycle_ID", &cycle_ID );
    RegistVar( "rotation_ID", &rotation_ID );
    RegistVar( "xray_ID", &xray_ID );
    RegistVar( "loop_ID", &loop_ID );
    RegistVar( "rot_speed", &rot_speed );
    RegistVar( "sim_speed", &sim_speed );

    mPluginName = wxString( _( "Fermentor" ) );

    wxImage my_img( _( "Icons/fermentor.jpg" ) );
    SetImage( my_img );

    // ## OperationBody End
}

double FermentorUI::GetVersion()
{
    // ## OperationBody [0456d9c9-1a98-408a-bead-1b2b5fd3769d]
    double result = 1.0;
    //Your code

    return result;
    // ## OperationBody End
}

wxString FermentorUI::GetConductorName()
{
    // ## OperationBody [55012352-f872-4996-b77b-7056cb46363b]
    //Your name
    wxString result( _( "IHCC_Fermentor" ) );

```

```

        return result;
    // ## OperationBody End

}

wxString FermentorUI::GetName()
{
    // ## OperationBody [25b85835-f0b8-4750-941b-1f7f2b55148f]
    return mPluginName;
    // ## OperationBody End

}

ves::conductor::UIDialog* FermentorUI::UI( wxWindow* parent )
{
    // ## OperationBody [128919d1-2502-4a6c-9d46-05b116cf3bd8]
    if( dlg )
    {
        return dlg;
    }

    dlg = new FermentorUIDialog( parent, wxID_ANY, &agitation,
                                &air_conc,
                                &ini_ph,
                                &nitrate_conc,
                                &temperature,
                                &hours,
                                &cycle_ID,
                                &rotation_ID,
                                &xray_ID,
                                &loop_ID,
                                &rot_speed,
                                &sim_speed );

    ConfigurePluginDialogs( dlg );

    return dlg;

    // ## OperationBody End

}

FermentorUI::~FermentorUI()
{
    // ## OperationBody [196070f4-e928-4128-8e08-a8aa6646adb9]
    ;
    // ## OperationBody End
}

// File : .\Simulation.h

#ifndef __simulation
#define __simulation

```

```
//#include "C:\Documents and Settings\askz82\Desktop\SysML -VESuite
Fermentor model\..\Program2\double.h"
```

```
class simulation
{
```

```
public:
```

```
    double agitation;
    double air_conc;
    double ini_ph;
    double nitrate_conc;
    double temperature;
    double hours;
```

```
};
```

```
#endif
```

```
// File : ..\Eqntocalculateconc.h
```

```
#ifndef EQNTOCALCULATECONC_H
#define EQNTOCALCULATECONC_H
```

```
class Eqntocalculateconc
{
public:
    Eqntocalculateconc();
```



```

double
calculateacidyield(int,double,double,double,double,double);

double c[ 8 ];
int time;
double agitation;
double airconc;
double iniph;
double nitratconc;
double temp;

};

// File : .\Eqntocalculateconc.cpp

#include "Eqntocalculateconc.h"

Eqntocalculateconc::Eqntocalculateconc()
{
    ;
}

double Eqntocalculateconc:: calculateacidyield(int t, double ag, double
ac, double iph, double niconc, double tempture)
{
    time =t;
    agitation=ag;
    airconc =ac;
    iniph = iph;
    nitratconc =niconc;
    temp = tempture;

    c[ 0 ] = 1;
    c[ 1 ] = ( -0.000036 * time * time * time ) + ( 0.0092 *
time * time ) - ( 0.072 * time ) + 1;
    c[ 2 ] = ( -0.000091 * agitation * agitation ) + 0.035 *
agitation - 2.56;
    c[ 3 ] = ( -1 * airconc * airconc ) + ( 2 * airconc ) - 2;
    c[ 4 ] = ( -0.41 * iniph * iniph ) + ( 4.9 * iniph ) - 13;
    c[ 5 ] = ( -17 * nitratconc * nitratconc ) + ( 8.4 *
nitratconc ) - 0.004;
    c[ 6 ] = ( -0.01 * temp * temp ) + ( 0.69 * temp ) - 7.8;
    c[ 7 ] = -1;

    for( int i = 1; i < 8; ++i )
    {
        c[ 0 ] = c[ 0 ] * c[ i ];
    }

    if( c[ 0 ] <= 0 )
    {
        c[ 0 ] = 0.0;
    }
}

```

```

        return c[0];
    }

// File : .\Feedingsystem.h

#ifndef __Feedingsystem
#define __Feedingsystem

//#include "C:\Documents and Settings\askz82\Desktop\SysML -VESuite
Fermentor model\..\Program2\double.h"
#include <osg/ref_ptr>
#include <osg/MatrixTransform>

class Feedingsystem
{

public:

    // Public|Package operations

    osg::ref_ptr< osg::Node > ReadCADdata();

    osg::ref_ptr< osg::Node > _fermentorGeometry;

    double airconc;

    double nitrateconc;

};

#endif

// File : .\Feedingsystem.cpp

#include "StdAfx.h"
#include "Feedingsystem.h"

#include <osgDB/ReadFile>

#include <iostream>
#include <iomanip>

// Operation implementations

osg::ref_ptr< osg::Node > Feedingsystem::ReadCADdata()
{

// ## OperationBody [11feb10e-328a-4f94-a6c9-473b14683f60]

```

```

    _fermentorGeometry = osgDB::readNodeFile(
"Models/fermentor_noimpeller.ive" );

    return _fermentorGeometry;
// ## OperationBody End
}

// File : .\Impellerdrivesystem.h

#ifndef __Impellerdrivesystem
#define __Impellerdrivesystem

// #include "C:\Documents and Settings\askz82\Desktop\SysML -VESuite
Fermentor model\..\Program2\double.h"
#include <osg/ref_ptr>
#include <osg/MatrixTransform>

class Impellerdrivesystem
{

public:

    // Public|Package operations

    osg::ref_ptr< osg::Node > ReadCADdataandInitialize();

    osg::ref_ptr< osg::Node > _impellerGeometry;

    double agitation_in_rpm;

};

#endif

// File : .\Impellerdrivesystem.cpp

#include "StdAfx.h"
#include "Impellerdrivesystem.h"

#include <osgDB/ReadFile>

#include <iostream>
#include <iomanip>

// Operation implementations

osg::ref_ptr< osg::Node >
Impellerdrivesystem::ReadCADdataandInitialize()
{

```

```

// ## OperationBody [6d290d37-6f08-46ea-a108-bf56d44ca489]
_impellerGeometry = osgDB::readNodeFile( "Models/impeller_fixed.ive" );

return _impellerGeometry;

// ## OperationBody End
}

// File : .\Tank.h

#ifndef __Tank
#define __Tank

//#include "C:\Documents and Settings\askz82\Desktop\SysML -VESuite
Fermentor model\...\Program2\double.h"
#include <osg/ref_ptr>
#include <osg/MatrixTransform>

class Tank
{

public:

    // Public|Package operations

    osg::ref_ptr< osg::Node > ReadCADdata();

    osg::ref_ptr< osg::Node > _tankGeometry;

    double initialpH;

    double temperature;

};

#endif

// File : .\Tank.cpp

#include "StdAfx.h"
#include "Tank.h"
#include <osgDB/ReadFile>

#include <iostream>
#include <iomanip>

// Operation implementations

osg::ref_ptr< osg::Node > Tank::ReadCADdata()
{

```

```

// ## OperationBody [a90f6320-fa65-4ae9-9913-0b9f75103d84]

_tankGeometry = osgDB::readNodeFile( "Models/opaque_tank.ive" );

return _tankGeometry;

// ## OperationBody End }

// File : .\FermentorGP.h

#ifndef __FermentorGP
#define __FermentorGP

class Shaders;

#include "Eqntocalculateconc.h"
#include "Impellerdrivesystem.h"
#include "Tank.h"
#include "Feedingsystem.h"

namespace display
{
    class DigitalGauge;
}

#include <ves/xplorer/plugin/PluginBase.h>

namespace ves
{
    namespace xplorer
    {
        namespace scenegraph
        {
            class DCS;
        }
    }
}

#include <map>

class VE_USER_PLUGIN_EXPORTS VEFermentorGraphicalPlugin : public
ves::xplorer::plugin::PluginBase
{
public:
    // Public|Package operations
    VEFermentorGraphicalPlugin();
    virtual ~VEFermentorGraphicalPlugin();

    virtual void InitializeNode( osg::Group* veworldDCS );
    virtual void PreFrameUpdate();
    virtual void ProcessOnSubmitJob();

    void UpdateGauges( double, double, double, double, double, double,
double );

```

```

private:
    // Private attributes
    bool mSimulationStart;

    int frame_count;
    int frame_speed_control;

    double _agitation;
    double _air_conc;
    double _ini_ph;
    double _nitrate_conc;
    double _temperature;
    double _hours;
    double _rot_speed;
    double _sim_speed;

    long _cycle_ID;
    long _rotation_ID;
    long _xray_ID;
    long _loop_ID;

    std::vector< double > time_steps;
    std::vector< double > result_steps;

    Shaders* shader;
    Eqntocalculateconc* constraint;
    Impellerdrivesystem* impeller;
    Tank* tank;
    Feedingsystem* feedingsystem;

    std::map< int, osg::ref_ptr< display::DigitalGauge > > _gauges;

    osg::ref_ptr< osg::Sequence > capsule_sequence;

    //osg::ref_ptr< osg::Node > _fermentorGeometry;
    //osg::ref_ptr< osg::Node > _impellerGeometry;
    osg::ref_ptr< osg::Node > _tankGeometry;

    osg::ref_ptr< osg::MatrixTransform > _roomGeometry;
    osg::ref_ptr< osg::MatrixTransform > fermentorGroup;

    osg::ref_ptr< osg::MatrixTransform > transform_ferm;
    osg::ref_ptr< osg::MatrixTransform > transform_imp;
    osg::ref_ptr< osg::MatrixTransform > transform_tank;
};

CREATE_VES_XPLORER_PLUGIN_ENTRY_POINT( VEFermentorGraphicalPlugin )

#endif

```

```

// File : .\FermentorGP.cpp

#include "FermentorGP.h"
#include "DigitalGauge.h"
#include "Shaders.h"
#include "Eqntocalculateconc.h"
#include "Impellerdrivesystem.h"
#include "Tank.h"
#include "Feedingsystem.h"

#include <ves/open/xml/model/Model.h>
#include <ves/open/xml/Command.h>
#include <ves/open/xml/shader/Shader.h>
#include <ves/open/xml/DataValuePair.h>

#include <ves/xplorer/scenegraph/SceneManager.h>

#include <osg/MatrixTransform>
#include <osg/AnimationPath>
#include <osg/ShapeDrawable>
#include <osg/Sequence>

#include <osgText/Text>

#include <osgDB/ReadFile>

#include <osgSim/ColorRange>

// Operation implementations
VEFermentorGraphicalPlugin::VEFermentorGraphicalPlugin()
:
  PluginBase(),
  _agitation( 200 ),
  _air_conc( 1.25 ),
  _ini_ph( 6 ),
  _nitrate_conc( 0.1 ),
  _temperature( 37 ),
  _hours( 240 ),
  _cycle_ID( 0 ),
  _rotation_ID( 0 ),
  _xray_ID( 0 ),
  _loop_ID( 0 ),
  _rot_speed( 0 ),
  _sim_speed( 0 ),
  frame_count( 0 ),
  frame_speed_control( 0 ),

  shader( new Shaders() ),
  constraint( new Eqntocalculateconc() ),
  impeller( new Impellerdrivesystem() ),

```

```

        tank ( new Tank() ),
        feedingsystem ( new Feedingsystem() ),

capsule_sequence( new osg::Sequence() ),

        _fermentorGeometry( 0 ),
        _impellerGeometry( 0 ),
        _tankGeometry( 0 ),

        _roomGeometry( new osg::MatrixTransform() ),
        fermentorGroup( new osg::MatrixTransform() ),

        transform_ferm( new osg::MatrixTransform() ),
        transform_imp( new osg::MatrixTransform() ),
        transform_tank( new osg::MatrixTransform() ),
        mSimulationStart( false )
    {
        mObjectName = "FermentorUI";
    }
}

VEFermentorGraphicalPlugin::~VEFermentorGraphicalPlugin()
{
    if( !mSceneManager )
    {
        return;
    }

    osg::ref_ptr< osg::Group > rootNode =
        mSceneManager->GetRootNode();

    if( !rootNode.valid() )
    {
        return;
    }

    rootNode->removeChild( _roomGeometry.get() );

    for( std::map< int, osg::ref_ptr< display::DigitalGauge >
>::iterator
        itr = _gauges.begin(); itr != _gauges.end(); ++itr )
    {
        rootNode->removeChild( itr->second.get() );
    }

    _gauges.clear();

    delete shader;
}

void VEFermentorGraphicalPlugin::InitializeNode( osg::Group* veworldDCS
)
{
    PluginBase::InitializeNode( veworldDCS );

    osg::ref_ptr< osg::Group > rootNode =

```



```

        ves::xplorer::scenegraph::SceneManager::instance()-
>GetRootNode();

        osg::ref_ptr< osg::Node > temp = osgDB::readNodeFile(
"Models/fermentor_room.ive" );
        _roomGeometry->addChild( temp.get() );
        rootNode->addChild( _roomGeometry.get() );
        mDCS->addChild( fermentorGroup.get() );

        osg::ref_ptr< osg::Node > Feedingsystemnode = feedingsystem-
>ReadCADdata();
        //_fermentorGeometry = osgDB::readNodeFile(
"Models/fermentor_noimpeller.ive" );

        //_impellerGeometry = osgDB::readNodeFile(
"Models/impeller_fixed.ive" );

        osg::ref_ptr< osg::Node > impellernode = impeller-
>ReadCADdataandInitialize();

        _tankGeometry = osgDB::readNodeFile( "Models/opaque_tank.ive" );

        //osg::ref_ptr< osg::Node > tanknode = tank->ReadCADdata();

        shader->XRay( _tankGeometry.get() );

        //shader->XRay( tanknode.get() );

        transform_ferm->addChild( Feedingsystemnode.get() );
        transform_imp->addChild( impellernode.get() );

        //transform_imp->addChild( _impellerGeometry.get() );
        //transform_tank->addChild( tanknode.get() );

        transform_tank->addChild( _tankGeometry.get() );
        fermentorGroup->addChild( capsule_sequence.get() );

        fermentorGroup->addChild( transform_ferm.get() );
        fermentorGroup->addChild( transform_imp.get() );
        fermentorGroup->addChild( transform_tank.get() );

        double trans[ 3 ] = { 0.8, 13.5, 0.15 };
        mDCS->SetTranslationArray( trans );

        _roomGeometry->setMatrix( osg::Matrix::scale( 3.28, 3.28, 3.28 ) *
                                osg::Matrix::translate( -4.5, 0.0, -3.4 )
*
                                osg::Matrix::rotate( 0.0, 0, 1, 0 ) );

```

```

        transform_ferm->setMatrix( osg::Matrix::scale( 3.28, 3.28, 3.28 ) *
                                   osg::Matrix::translate( -0.67, 0.8, -
1.36 ) );

        transform_tank->setMatrix( osg::Matrix::scale( 3.28, 3.28, 3.28 ) *
                                   osg::Matrix::translate( 0.005, -0.02, -
0.05 ) );

        _gauges.insert( std::make_pair( 0, new display::DigitalGauge(
"Time: Hours" ) ) );
        _gauges.insert( std::make_pair( 1, new display::DigitalGauge( "Acid
Yield" ) ) );
        _gauges.insert( std::make_pair( 2, new display::DigitalGauge(
"Agitation: rpm" ) ) );
        _gauges.insert( std::make_pair( 3, new display::DigitalGauge( "Air
Conc: vvm" ) ) );
        _gauges.insert( std::make_pair( 4, new display::DigitalGauge(
"Initial pH" ) ) );
        _gauges.insert( std::make_pair( 5, new display::DigitalGauge(
"Nitrate: g/L" ) ) );
        _gauges.insert( std::make_pair( 6, new display::DigitalGauge(
"Temp: C" ) ) );

        for( std::map< int, osg::ref_ptr< display::DigitalGauge >
>::iterator
            itr = _gauges.begin(); itr != _gauges.end(); ++itr )
        {
            rootNode->addChild( itr->second.get() );

            itr->second->GetNameText()->setCharacterSize( 0.12 );
            itr->second->GetDigitalText()->setCharacterSize( 0.22 );
            itr->second->GetNameText()->setColor( osg::Vec4( 0.3, 0.3, 0.3,
1.0 ) );
            itr->second->GetDigitalText()->setColor( osg::Vec4( 0.0, 1.0,
0.0, 1.0 ) );
        }

        _gauges[ 0 ]->SetPrecision( 0 );
        _gauges[ 1 ]->SetPrecision( 0 );
        _gauges[ 2 ]->SetPrecision( 0 );
        _gauges[ 3 ]->SetPrecision( 1 );
        _gauges[ 4 ]->SetPrecision( 1 );
        _gauges[ 5 ]->SetPrecision( 2 );
        _gauges[ 6 ]->SetPrecision( 1 );

        _gauges[ 0 ]->setMatrix( osg::Matrix::translate( 2.5, 6, 3.5 ) );
        _gauges[ 1 ]->setMatrix( osg::Matrix::translate( 2.5, 6, 2.9 ) );
        _gauges[ 2 ]->setMatrix( osg::Matrix::translate( -2.5, 6, 4.7 ) );
        _gauges[ 3 ]->setMatrix( osg::Matrix::translate( -2.5, 6, 4.1 ) );
        _gauges[ 4 ]->setMatrix( osg::Matrix::translate( -2.5, 6, 3.5 ) );
        _gauges[ 5 ]->setMatrix( osg::Matrix::translate( -2.5, 6, 2.9 ) );
        _gauges[ 6 ]->setMatrix( osg::Matrix::translate( -2.5, 6, 2.3 ) );
    }

    void VEFermentorGraphicalPlugin::ProcessOnSubmitJob()
    {

```

```

    mXmlModel->GetInput( "agitation" )->GetDataValuePair( "agitation"
)->GetData( _agitation );
    mXmlModel->GetInput( "air_conc" )->GetDataValuePair( "air_conc" )-
>GetData( _air_conc );
    mXmlModel->GetInput( "ini_ph" )->GetDataValuePair( "ini_ph" )-
>GetData( _ini_ph );
    mXmlModel->GetInput( "nitrate_conc" )->GetDataValuePair(
"nitrate_conc" )->GetData( _nitrate_conc );
    mXmlModel->GetInput( "temperature" )->GetDataValuePair(
"temperature" )->GetData( _temperature );
    mXmlModel->GetInput( "hours" )->GetDataValuePair( "hours" )-
>GetData( _hours );
    mXmlModel->GetInput( "cycle_ID" )->GetDataValuePair( "cycle_ID" )-
>GetData( _cycle_ID );
    mXmlModel->GetInput( "rotation_ID" )->GetDataValuePair(
"rotation_ID" )->GetData( _rotation_ID );
    mXmlModel->GetInput( "xray_ID" )->GetDataValuePair( "xray_ID" )-
>GetData( _xray_ID );
    mXmlModel->GetInput( "loop_ID" )->GetDataValuePair( "loop_ID" )-
>GetData( _loop_ID );
    mXmlModel->GetInput( "rot_speed" )->GetDataValuePair( "rot_speed"
)->GetData( _rot_speed );
    mXmlModel->GetInput( "sim_speed" )->GetDataValuePair( "sim_speed"
)->GetData( _sim_speed );

    _rot_speed = _rot_speed / 10.0f;

    {
        _sim_speed = 1.1 - ( _sim_speed / 10.0 );
        std::cout << "Sim speed will be " << _sim_speed << std::endl;
    }

    std::fstream results;
    results.open( "results.txt", std::ios::out );

    double acidyield;

    double min = 10000000000;
    double max = -10000000000;

    time_steps.clear();
    result_steps.clear();

    if( _rotation_ID == 0 )
    {
        fermentorGroup->setUpdateCallback( new
osg::AnimationPathCallback(
            osg::Vec3( 0, 0, 0 ), osg::Z_AXIS, 0.0f ) );
    }
    else if( _rotation_ID == 1 )
    {
        fermentorGroup->setUpdateCallback( new
osg::AnimationPathCallback(

```

```

        osg::Vec3( 0, 0, 0 ), osg::Z_AXIS, _rot_speed ) );
    }

    if( _cycle_ID == 0 )
    {
        time_steps.push_back( 0 );
        result_steps.push_back( 0 );

        capsule_sequence->removeChildren( 0, static_cast< int >(
            capsule_sequence->getNumChildren() ) );
        capsule_sequence->setMode( osg::Sequence::STOP );
    }
    else if( _cycle_ID == 1 )
    {
        capsule_sequence->removeChildren( 0, static_cast< int >(
            capsule_sequence->getNumChildren() ) );

        results << "Agitation(rpm):\t" << _agitation << "\n";
        results << "Air Conc(vvm):\t" << _air_conc << "\n";
        results << "Initial pH:\t" << _ini_ph << "\n";
        results << "Nitrate(g/L):\t" << _nitrate_conc << "\n";
        results << "Temp(C): \t" << _temperature << "\n\n";

        results << "t (hours):\t\t";
        results << "Acid Yield:\n";

        for( int t = 0; t <= _hours; ++t )
        {
            acidyield = constraint->calculateacidyield( t,
                _agitation, _air_conc, _ini_ph, _nitrate_conc, _temperature);
            results << t << "\t\t\t";
            results << acidyield << "\n";
            time_steps.push_back( t );
            result_steps.push_back( acidyield );
        }

        if( max == min )
        {
            min = 0.0f;
            max = 0.0000001;
        }

        //Create a custom color set
        std::vector< osg::Vec4 > cs;
        cs.push_back( osg::Vec4( 0.0f, 0.0f, 1.0f, 0.4f ) ); //Blue
        cs.push_back( osg::Vec4( 0.0f, 1.0f, 1.0f, 0.4f ) ); //Cyan
        cs.push_back( osg::Vec4( 0.0f, 1.0f, 0.0f, 0.4f ) ); //Green
        cs.push_back( osg::Vec4( 1.0f, 1.0f, 0.0f, 0.4f ) ); //Yellow
        cs.push_back( osg::Vec4( 1.0f, 0.0f, 0.0f, 0.4f ) ); //Red

        osg::ref_ptr< osgSim::ColorRange > cr = new osgSim::ColorRange(
            min, max, cs );

        for( int t = 0; t <= _hours; ++t )
        {
            //Create concentration color capsules

```

```

        osg::ref_ptr< osg::Geode > geode_0 = new osg::Geode;

        osg::ref_ptr< osg::Capsule > capsule = new osg::Capsule(
osg::Vec3( 0.0, 0.0, 1.95 ), 0.75, 2.7 );
        osg::ref_ptr< osg::TessellationHints > hints = new
osg::TessellationHints();
        osg::ref_ptr< osg::ShapeDrawable > sd = new
osg::ShapeDrawable( capsule.get(), hints.get() );

        hints->setDetailRatio( 1.0f );

        sd->setColor( cr->getColor( result_steps.at( t ) ) );

        osg::ref_ptr< osg::StateSet > stateset_0 = new
osg::StateSet();
        stateset_0->setMode( GL_BLEND, osg::StateAttribute::ON );
        stateset_0->setRenderBinDetails( 8, std::string(
"DepthSortedBin" ) );
        sd->setStateSet( stateset_0.get() );

        geode_0->addDrawable( sd.get() );

        capsule_sequence->addChild( geode_0.get(), _sim_speed );

    }

    double _imp_speed = 0.0f;
    if( _sim_speed > 0.0f )
    {
        _imp_speed = _agitation / 30.0f;
    }

    transform_imp->setUpdateCallback( new
osg::AnimationPathCallback(
        osg::Vec3( 0, 0, 0 ), osg::Z_AXIS, _imp_speed ) );

    mSimulationStart = true;
}

if( _sim_speed == 0 )
{
    capsule_sequence->setMode( osg::Sequence::PAUSE );
}

results.close();

capsule_sequence->setInterval( osg::Sequence::LOOP, 0, -1 );

if( _loop_ID == 0 )
{
    capsule_sequence->setDuration( 1.0f, 1 );
}

if( _loop_ID == 1 )
{

```

```

        capsule_sequence->setDuration( 1.0f, -1 );
    }

    if( _xray_ID == 0 )
    {
        shader->Phong( _tankGeometry.get() );
    }
    else if( _xray_ID == 1 )
    {
        shader->XRay( _tankGeometry.get() );
    }

}

void VEFermentorGraphicalPlugin::PreFrameUpdate()
{
    if( time_steps.empty() || result_steps.empty() )
    {
        return;
    }

    if( mSimulationStart )
    {
        capsule_sequence->setMode( osg::Sequence::START );
        mSimulationStart = false;
    }

    int seqVal = capsule_sequence->getValue();
    if( seqVal > -1 )
    {
        UpdateGauges( time_steps[ seqVal ],
                      result_steps[ seqVal ],
                      _agitation,
                      _air_conc,
                      _ini_ph,
                      _nitrate_conc,
                      _temperature );
    }

    if( seqVal == _hours && _loop_ID == 0 )
    {
        transform_imp->setUpdateCallback(
            new osg::AnimationPathCallback( osg::Vec3( 0, 0, 0 ),
            osg::Z_AXIS, 0.0f ) );
        capsule_sequence->setMode( osg::Sequence::STOP );
    }
}

void VEFermentorGraphicalPlugin::UpdateGauges( double time_for_update,
double
result_for_update,
double agitation,
double air_conc,
double ini_ph,
double nitrate_conc,

```

```
double temperature )  
{  
  _gauges[ 0 ]->UpdateText( time_for_update );  
  _gauges[ 1 ]->UpdateText( result_for_update );  
  _gauges[ 2 ]->UpdateText( agitation );  
  _gauges[ 3 ]->UpdateText( air_conc );  
  _gauges[ 4 ]->UpdateText( ini_ph );  
  _gauges[ 5 ]->UpdateText( nitrate_conc );  
  _gauges[ 6 ]->UpdateText( temperature );  
}
```

APPENDIX B
GUIDELINES

To help users in creating the virtual engineering interface in the MBSE tool, basic steps and cautions that are important in the model development are mentioned below.

Basic steps for creating the user interface:

- Step1: Decide the type of analysis that you want to perform using information available in the system formal model. This is important because depending on the type of analysis it will be decided whether a CFD package will be required or a custom-made computational unit capable of running differential equations be enough.
- Step2: Select the parameters that will be used in the analysis. These parameters will be the ones that will be used while running the experiment in the virtual environment; thus both the input and output parameters of the experiment will have to be determined.
- Step 3: Create a simulation block to store these experimental parameters.
- Step 4: Specify value types of all the parameters using UML datatypes library. This is important because VE-Suite is C++ based, thus SysML value types cannot be used directly in the IDE.
- Step 5: Once the parameters are defined in the simulation block the next step is to link these parameters with value properties defined in subsystem blocks. This linkage can be created using a parametric diagram and by defining an equivalent constraint in it. This also helps to establish traceability between the simulation model and the system model.
- Step 6: Define a new block in the MBSE tool to represent the user interface module (conductor) of the virtual engineering software. Establish parent-child relationship between simulation and VE-UI block using inheritance association. At this point, apply the C++ profile to the entire model. Information on inserting a profile can be found in the help section of the MBSE tool (Artisan Studio in this case). Development of the virtual engineering interface in the MBSE environment depends on the specification of information about the header files and operations that will be required to create the plugins. Operations will have to be specified using UML operations construct whereas header information is provided using C++ stereotype applied to the newly created VE-UI block. The developer has to specify the body of the operations. The type of header files required for the UI module can be found in the code provided in Appendix A.

Basic steps for creating the graphical interface:

- Step1: Creating a graphical interface in the MBSE tool requires a thorough definition of the structural components in the system model. Each component or subsystem to be represented in the graphical engine of the virtual engineering software requires a CAD file associated with it. Thus, the first step in developing the graphical interface is to have the CAD files in place for the entire system.
- Step2: The way VE-Suite works with these files is by using OSG libraries to create a tree structure composed of OSG nodes. Each node can be used to represent a structural component/subsystem of the SysML model. Thus for each subsystem block define a UML operation to read and store the CAD file in the form of an OSG node. An <OSGDB/Readfile> header file is required to perform this operation.
- Step3: Once these individual CAD files are associated with the relevant structural elements of the system, the next step is to create the graphical interface block in the MBSE tool. This block will store operations and header information that is required to generate the VE-Suite graphical plugin.
- Step 4: Connect each structural element of the system to this block using reference association construct. By doing this, the graphical interface block will have access to the CAD files stored as OSG nodes. The formation of the tree structure to bring these nodes together is done using an initialize operation. This operation will specify the root node and its association with the display coordinate system.

BIBLIOGRAPHY

1. S. Friedenthal, A. Moore, and R. Steiner, A practical guide to SysML: The Systems Modeling Language, Morgan Kaufmann, San Francisco, 2008.
2. S. Friedenthal, H. Lykins, "Parameter-based representation for modeling complex systems", Proceedings IEEE Symposium and Workshop on Engineering of Computer-Based Systems, 11-15 Mar, 1996, 65-71.
3. J. Axelsson, "Model based systems engineering using a continuous time extension of the unified modeling language", Systems Engineering 5(3) (2002), 165-179.
4. E. Huang, R. Ramamurthy and L.F. McGinnis, "System and simulation modeling using SYSML", Proceedings of the Winter Simulation Conference, 2007.
5. G. Huang, K.M. Bryden, "Introducing virtual engineering technology into interactive design process with high-fidelity models", Proceedings of the 37th conference on winter simulation, 2005, 1958-1967.
6. B. Edmund. "What is Complexity? – The philosophy of complexity per se with application to some examples in evolution" in F. Heylighen & D. Aerts (eds.): The Evolution of Complexity, Kluwer, Dordrecht, 1998.
7. B. Edmund. "Syntactic Measure of Complexity" Doctoral thesis, 1999, University of Manchester.
8. NASA Systems Engineering Handbook, NASA/SP-2007-6105 Rev 1.
9. W. W. Royce, "Managing the development of large software systems" Proceedings of the IEEE Wescon, 1970, Vol. 26, pp. 1-9.
10. K. Forsberg, H. Mooz, T.N. Chattanooga, N. Oslo, "The relationship of system engineering to the project cycle", Center for Systems Management, 1994, Vol. 9.
11. G. Pahl, W. Beitz, K. Wallace, Engineering design: a systematic approach, Springer Verlag, 1996.
12. K. Forsberg, H. Mooz, H. Cotterman, Visualizing project management: Models and frameworks for mastering complex systems, Wiley, 2005.
13. C. N. Calvano, P. John, "Systems engineering in an age of complexity", Systems Engineering, 2004, Vol. 7, pp. 25–34.

14. H. Lykins, S. Friedenthal, A. Meilich, "Adapting UML for an Object Oriented Systems Engineering Method", Proceedings of the tenth annual international symposium on the International Council on Systems Engineering, 2000, Vol. 2000.
15. T. Bahill, J. Daniels, "Using Objected-Oriented and UML Tools for Hardware Design: A Case Study", Systems Engineering, 2003, Vol. 6, pp. 28–48.
16. J. Hsu, J.M. McDonough, "Applying the object oriented systems engineering method to a simple hardware system", Conference on Systems Engineering Research, 2004, Paper no 142.
17. J. A. Estafen, "Survey of model-based systems engineering (MBSE) methodologies Rev.A", INCOSE MBSE Focus Group, 2007.
18. G. Arthurs, "Model based systems engineering- Elements for deploying an Efficient Development Environment", Telelogic White paper version 1.0, 2008.
19. D. Mavris, "A Review of the Proposed UML Profile for the Department of Defense Architecture Framework and Ministry of Defense Architecture Framework (UPDM)", INCOSE, 2007.
20. M. Hause, "An Overview of UPDM—a Unified Profile for DoDAF/MODAF", Military Embedded Systems, 2008.
21. M. Hause, "Model-Based System of Systems Engineering with UPDM", Omg.org, 2010.
22. D. Dori, I. Reinhartz-Berger, "OPCAT-a bimodal CASE tool for object-process based system development", 5th International Conference on Enterprise Information Systems (ICEIS 2003), 2003, pp. 286–291.
23. N. R. Soderborg, E. F. Crawley, D. Dori, "System function and architecture: OPM-based definitions and operational templates", Communications of the ACM, 2003, Vol. 46, pp. 67–72.
24. Y. Grobshtein, V. Perelman, E. Safra, D. Dori, "Systems Modeling Languages: OPM Versus SysML", ICSEM'07. International Conference on Systems Engineering and Modeling, 2007, pp. 102–109.
25. Object Management Group (OMG), 2007b, OMG Systems Modeling Language (OMG SysML™), V1.0, OMG Document Number: formal/2007-09-01, <http://www.omg.org/spec/SysML/1.0/PDF>, Accessed November 10, 2010.

26. Object Management Group (OMG), 2009a, Unified Profile for the Department of Defense Architecture Framework (DoDAF) and the Ministry of Defence Architecture Framework (MODAF), <http://www.omg.org/spec/UPDM/>, Last Accessed November 10, 2010.
27. T. A. Johnson, C.J.J. Paredis, J.M. Jobe and R. Burkhart, "Modeling continuous system dynamics in SysML", International Mech. Eng. Congress and Exposition, 2007.
28. T. A. Johnson, "Integrating models and simulations of continuous dynamic system behavior into SysML", M.S. Thesis, Georgia Institute of Technology, 2008.
29. R.S. Peak, R.M. Burkhart, S.A. Friedenthal, M.W. Wilson, M. Bajaj, and I. Kim, "Simulation-Based Design Using SysML-Part1: A Parametrics Primer", INCOSE Intl. Symposium, San Diego, CA, 2007.
30. R.S. Peak, S.A. Friedenthal, R.M. Burkhart, M.W. Wilson, and M. Bajaj, "Simulation-Based Design Using SysML-Part 2: Celebrating Diversity by Example", INCOSE Intl. Symposium, San Diego, CA, 2007.
31. D. E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In Proceedings of the Third International Conference on Genetic Algorithms, pages 70–79. Morgan Kaufmann Publishers Inc., 1989.
32. Smith, R. E "Adaptively resizing population: An algorithm and analysis", in proceedings of fifth international conference on genetic algorithms Morgan Kauffman Publishers 1993.
33. D.S. McCorkle, and K.M. Bryden, Virtual engineering and design of power systems, Section 3, Thermal Engineering in Power Systems, R.S. Amano and B. Sunden, Editors, WIT Press, do Computational Mechanics Inc., 25 Bridge St., Billerica, MA 01821, 2008.
34. S. Wölkl, and K. Shea, "A computation product model for conceptual design using SysML", Proceedings of the ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE, San Diego, California, USA, 2009.
35. C. Gershenson, F. Heylighen, "How can we think the complex? in: Richardson, Kurt (ed.)Managing the Complex Vol. 1: Philosophy, Theory and Application", Institute for the Study of Coherence and Emergence/Information Age Publishing, 2005, p.47-62.
36. Artisan Studio website www.Atego.com (last accessed: Aug. 9, 2010).
37. INCOSE website www.incose.org (last accessed: Aug. 9, 2010).

38. Object Management Group website www.omg.org(last accessed: Aug. 9, 2010).
39. www.omgsysml.org (last accessed: Dec. 3, 2010).
40. VE-Suite website www.vesuite.org (last accessed: Aug. 9, 2010).
41. WxWidgets website www.wxwidgets.org(last accessed: Aug. 9, 2010).
42. Rheingold, H. Virtual Reality. Summit, New York, 1991.
43. SCons website www.scons.org (last accessed: Dec 3, 2010).
44. J.D.Poole, “Model-Driven Architecture: Vision, Standards and Emerging Technologies”, position paper submitted to European Conference on Object-Oriented Programming, 2001.
45. R.Soley and the OMG Staff Strategy Group, “Model Driven Architecture”, Object Management Group, White paper Draft 3.2, 2000.
46. C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, “Surround-screen projection-based virtual reality: the design and implementation of the CAVE”, Proceedings of the 20th annual conference on Computer graphics and interactive techniques, 1993, pp. 135–142.

VITA

Akshay Kande received his Bachelor's degree in Mechanical Engineering from University of Mumbai, India. He is currently pursuing his Master's degree in Systems Engineering at Missouri S&T, Rolla. Along with his studies, he works as a Graduate Research Assistant at the Smart Engineering Systems Laboratory at Missouri S&T. His research interests include Virtual Engineering, Computer Aided Engineering and Analysis, and model-based systems engineering. With three conference paper publications and a journal article submission, he was also recognized with the Outstanding Graduate Student Research Award for the year 2009-10 from the Engineering Management and Systems Engineering Department. Prior to joining the Master's program he has worked as a Design Engineer for steel plant projects at Mukand Ltd, India. Apart from his academic and research commitments, he is currently working as the President of the International Council on Systems Engineering (INCOSE) student Section at Missouri S&T. In May 2011 he will receive his Master of Science degree in Systems Engineering.