Development of a Systems Engineering Model of the Chemical Separations Process

Yitung Chen
University of Nevada, Las Vegas, yitung.chen@unlv.edu

Sean Hsieh
University of Nevada, Las Vegas, hsiehht@nscee.edu

Follow this and additional works at: https://digitalscholarship.unlv.edu/hrc_trp_separations

Part of the Analytical Chemistry Commons, Nuclear Engineering Commons, Oil, Gas, and Energy Commons, Operational Research Commons, Software Engineering Commons, and the Systems Engineering Commons

Repository Citation

This Report is brought to you for free and open access by the Transmutation Research Program Projects at Digital Scholarship@UNLV. It has been accepted for inclusion in Separations Campaign (TRP) by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
FINAL REPORT

Development of a Systems Engineering Model of the Chemical Separations Process

Submitted to

Transmutation Research Project Program
Technical Focus Area
Fuel Development Research
ATTN: Dr. Anthony Hechanova
Harry Reid Center
University of Nevada Las Vegas

Submitted by

Dr. Yitung Chen, Principal Investigator, uuchen@nscee.edu
Dr. Sean Hsieh, Co-Principal Investigator, hsieht@nscee.edu
Department of Mechanical Engineering
University of Nevada Las Vegas
4505 Maryland Parkway, Box 454027
Las Vegas, NV 89154-4027
Phone: (702) 895-1202
Fax: (702) 895-3936

Collaborators

Dr. James J. Laidler, Senior Scientist
Dr. George F. Vandergrift, III, Senior Scientist
Chemical Technology Division
Argonne National Laboratory, Argonne, IL

Project Dates: 9/1/01 – 8/31/04

September 30, 2004
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES ............................................................................................................... iv</td>
</tr>
<tr>
<td>EXECUTIVE SUMMARY .......................................................................................................... vi</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION AND BACKGROUND .............................................................. 1-1</td>
</tr>
<tr>
<td>1.1 Project Background – TRP Program and Chemical Separation Process .............. 1-1</td>
</tr>
<tr>
<td>1.2 Objective ......................................................................................................................... 1-4</td>
</tr>
<tr>
<td>1.3 Capabilities and Features of the Model ................................................................. 1-6</td>
</tr>
<tr>
<td>1.4 The Structure of this project ..................................................................................... 1-6</td>
</tr>
<tr>
<td>1.5 Software Implementation Tool ................................................................................. 1-7</td>
</tr>
<tr>
<td>CHAPTER 2 SYSTEM ENGINEERING ............................................................................... 2-1</td>
</tr>
<tr>
<td>2.1 Introduction to System Engineering ......................................................................... 2-1</td>
</tr>
<tr>
<td>2.1.1 The Need for the System Engineering? ................................................................. 2-1</td>
</tr>
<tr>
<td>2.1.2 What is a System? .................................................................................................... 2-2</td>
</tr>
<tr>
<td>2.1.3 Definition of Systems Engineering ..................................................................... 2-3</td>
</tr>
<tr>
<td>2.2 Introduction to Modeling ............................................................................................ 2-3</td>
</tr>
<tr>
<td>2.2.1 Importance for Modelling ....................................................................................... 2-3</td>
</tr>
<tr>
<td>2.2.2 Definition of a Model ............................................................................................. 2-5</td>
</tr>
<tr>
<td>2.2.3 Principles of Modeling ........................................................................................... 2-5</td>
</tr>
<tr>
<td>2.3 Implementation of System Engineering Techniques ............................................. 2-6</td>
</tr>
<tr>
<td>2.3.1 System Identification and Familiarization ............................................................ 2-7</td>
</tr>
<tr>
<td>2.3.2 Model Isolation and Boundary Setting ................................................................. 2-7</td>
</tr>
<tr>
<td>2.3.3 Analysis of System Equations ............................................................................... 2-8</td>
</tr>
<tr>
<td>2.4 The System Architecture for TRPSEMPro ............................................................ 2-10</td>
</tr>
<tr>
<td>2.5 TRPSEMPro’s Role in the Design Cycle .................................................................. 2-11</td>
</tr>
<tr>
<td>CHAPTER 3 DOE STUDY AND OPTIMIZATION ............................................................ 3-1</td>
</tr>
<tr>
<td>3.1 Design of Experiments (DOE) Study ....................................................................... 3-1</td>
</tr>
<tr>
<td>3.1.1 Purposes of Using Design of Experiments (DOE) Study ...................................... 3-1</td>
</tr>
<tr>
<td>3.1.2 DOE Study Techniques ......................................................................................... 3-2</td>
</tr>
<tr>
<td>3.2 Optimization ............................................................................................................... 3-6</td>
</tr>
<tr>
<td>3.2.1 Introduction to Optimization .................................................................................. 3-6</td>
</tr>
<tr>
<td>3.2.2 Formal Elements of Optimization Problem .......................................................... 3-7</td>
</tr>
<tr>
<td>3.2.3 Category of Optimization Problem ....................................................................... 3-9</td>
</tr>
<tr>
<td>3.2.4 Basic Mathematical Concepts ............................................................................. 3-9</td>
</tr>
<tr>
<td>3.2.5 Numerical Techniques for Nonlinear Programming Problem ......................... 3-13</td>
</tr>
</tbody>
</table>
6.1.5 Configure the SolutionViewer ................................................................. 6-11
6.1.6 Execution of Study Plan ........................................................................ 6-14
6.1.7 Solution to this Problem ................................................................. 6-16
6.2 Test Case 2 - Minimization of the “Banana Function” .......................... 6-16
  6.2.1 The Rosenbrock function ................................................................ 6-16
  6.2.2 What is the Problem: Minimization of the “Banana Function” ....... 6-17
  6.2.3 Solving the Problem by Using TRPSEMPPro .................................. 6-17
6.3 Test Case 3 – Application of ALM Method ............................................. 6-20
  6.3.1 What is the Problem to Be Solved? ............................................... 6-20
  6.3.2 Solving Problem by Using TRPSEMPPro ....................................... 6-20
6.4 Integration of Chemical Separation with the System Model ................. 6-22

CHAPTER 7 CONCLUSION .............................................................................. 7-1

APPENDIX A XML SCHEMA FOR UREX PROCESS ................................. A-1
APPENDIX B XML SYSTEM ENGINEERING MODEL DESCRIPTION FILE. B-1
REFERENCES ................................................................................................. R-1
| Figure 1-1     Overall Chemical Separations System for TRP Program                      | 1-1 |
| Figure 1-2     UREX Process                                                          | 1-4 |
| Figure 1-3     Project Tasks                                                          | 1-5 |
| Figure 2-1     Definition of System                                                   | 2-2 |
| Figure 2-2     Schematic representation of the process for model development         | 2-6 |
| Figure 2-3     UREX Process                                                          | 2-8 |
| Figure 2-4     System Architecture for TRPSEMPro                                      | 2-11|
| Figure 2-5     Multi-Program Design Cycle                                             | 2-12|
| Figure 3-1     Full-Factorial Study                                                   | 3-3 |
| Figure 3-2     Parameter Study                                                        | 3-3 |
| Figure 3-3     Central Composite Design                                               | 3-5 |
| Figure 3-4     Category of Optimization Problem                                       | 3-9 |
| Figure 3-5     Generic Algorithm for Unconstrained Optimization                      | 3-14|
| Figure 3-6     Algorithm – Random Walk                                               | 3-15|
| Figure 3-7     Algorithm - Conjugate Gradient (Fletcher-Reeves) Method                | 3-17|
| Figure 3-8     Algorithm - Davidon-Fletcher-Powell Method (DFP)                         | 3-18|
| Figure 3-9     Algorithm - Augmented Lagrange Multiplier (ALM) method                 | 3-20|
| Figure 4-1     Phases of Software Process                                             | 4-2 |
| Figure 4-2     OOA&D Roadmap                                                          | 4-5 |
| Figure 4-3     OOA&D Approach for Obtaining Domain Classes                            | 4-6 |
| Figure 4-4     Use Case for System Engineering Model                                   | 4-7 |
| Figure 4-5     Sequence Diagram for System Engineering Model                          | 4-8 |
| Figure 4-6     Use Case for Model Integration                                          | 4-9 |
| Figure 4-7     Sequence Diagram for ModelIntegration                                   | 4-10|
| Figure 4-8     Use case for StudyPlanBuilder                                         | 4-11|
| Figure 4-9     Sequence Diagram for StudyPlanBuilder                                 | 4-12|
| Figure 4-10    Use case for SolutionViewer                                          | 4-13|
| Figure 4-11    Sequence diagram for SolutionViewer                                  | 4-14|
| Figure 4-12    System Architecture                                                 | 4-16|
| Figure 4-13    Static Diagram for System Engineering Data Model                      | 4-17|
| Figure 4-14    Sequence Diagram for Composite Design Pattern in DataModel           | 4-18|
| Figure 4-15    Static Diagram for ModelIntegration                                   | 4-18|
| Figure 5-1     Relationship among System Engineering Model, AMUSESimulator, and     | 5-2 |
|                AMUSE Macro                                                              |     |
| Figure 5-2     Framework of AMUSESimulator                                          | 5-3 |
| Figure 5-3     Identify Objects in UREX Process (Provided by ANL)                    | 5-4 |
| Figure 5-4     Architecture of AMUSESimulator                                        | 5-5 |
EXECUTIVE SUMMARY

The whole chemical separation process is complex to the point that definitely requires certain level of systematic coordination. To perform smoothly and meet the target extraction rates among those processes, this research proposed a general-purpose systems engineering model. Since constructing a system model is generally complex, requiring intensive communication and in-depth understanding, a carefully designed model can be more flexible and useful in the long term. This research project considered the design concepts from requirements definition and conceptual design to system partitioning, and finally system validation. Lengthy pre-coding of the design process and recursive modification provides a system with high degree of flexibility and robustness.

A general purposed systems engineering model, Transmutation Research Program System Engineering Model Project (TRPSEMPPro), was developed based on the above design concept. The system model includes four main parts: System Manager, Model Integration, Study Plan, and Solution Viewer. System Manager supervises all the case (problem) creation, and functionality definition. Model Integration identifies chemical extraction processes and their execution sequence. Study Plan is the key to define modeling scenarios, such as Optimization, Design of Experiments, single-set parameter and multiple parameter-set. No system can be completed without a visualization tool. Solution Viewer provides a visual means to monitor the optimization process during and after model execution. TRPSEMPPro can apply not only to chemical separation process, but also a general system model. TRPSEMPPro allows industries to model their process
quantitatively and to study the interactions between subsystems and performance of the model under the influence of various design parameters.

Software engineering and Object Oriented Analysis and Design (OOA&D) play a critical role during our software development. Through the application of OOA&D, the user can define objects and concepts from our problem domain that is quantitatively described by Unified Modeling Language (UML). The logical software objects were created from the previous definition. Meanwhile, different design patterns were also applied during the detailed design phase. Finally, those designed components were implemented by using Microsoft.NET, the most up-to-date object-oriented programming language framework from Microsoft.

Currently, only the UREX process module is available and ready to be implemented. Since extraction modules can be developed from various agencies with different development concepts and programming conventions, an intermediate bridge or interpreter is generally required. The system connects the only available process, UREX and with the TRPSEMPro system model from the AMUSESimulator interface. The AMUSESimulator communicates with the calculation engine AMUSE macros designed for the UREX process. A user-friendly GUI in AMUSESimulator allows the user to efficiently define the UREX process – flowsheet, input streams, sections, and stages.

The combination of several up-to-date techniques makes this research unique and robust. Those include Microsoft.NET framework, MS SQL Server database, MS ACCESS, eXtensible Markup Language (XML), Design of Experiment and System Optimization. The design and implementation of database provide an effective way to manipulate and store all the input and output data generated during the system design run.
To analyze the system in an optimum way, several optimization technologies have been studied and correspondent algorithms have been developed and implemented into the system model. A universal XML file format is applied for data storage and transport among modules. While waiting more chemical processes to be developed, more improvement will be made to enhance the TRPSEMPro package for solving complex chemical separation process.
CHAPTER 1
INTRODUCTION AND BACKGROUND

Chapter 1 introduces the project background information and then give a brief overview about this project’s content.

1.1 Project Background – TRP\(^1\) Program and Chemical Separation Process

![Diagram](image)

Figure 1-1 Overall Chemical Separations System for TRP Program

The United States is embarking on a national program to develop accelerator transmutation of high-level radioactive waste (ATW) as part of the Transmutation Research Program (TRP) project at its national laboratories. Through the Program TRP,

\(^{1}\) TRP - Transmutation Research Program
the U.S. joins international efforts to evaluate the potential of partitioning and transmutation along with advanced nuclear fuel cycles. Transmutation means nuclear transformation that changes the contents of the nucleus (protons and/or neutrons). The TRP is a developing technology for the transmutation of nuclear waste to address many of the long-term disposal issues. An integral part of this program is the proposed chemical separations scheme. Figure 1-1 shows a block diagram of the current process as envisioned by Argonne National Laboratory (ANL) researchers (Gregory R. Choppin, 1999; G. F. Vandegrift, 1992; Gregory R. Choppin, 2002).

Nearly all issues related to risks to future generations arising from long-term disposal of such spent nuclear fuel is attributable to ~1% of its content. This 1% is made up primarily of plutonium, neptunium, americium, and curium (the transuranic elements) and long-lived isotopes of iodine and technetium created as products from the fission process in power reactors. When transuranics are removed from discharged fuel destined for disposal, the toxic nature of the spent fuel drops below that of natural uranium ore (that was originally mined for nuclear fuel) within a period of several hundred years.

Figure 1-1 depicts the fuel cycle scheme in which the transuranic elements and long-lived fission products from Light Water Reactor (LWR) spent fuel are sent directly to an accelerator-driven subcritical reactor for transmutation. Other schemes under consideration involve intermediate critical reactor steps; this would result in major changes in the design, development and analysis of separations systems. Systems engineering would enhance the ability to respond with such changes.

Removal of plutonium and other transuranics from material destined for geologic disposal also eliminates issues related to long-term (centuries) heat management within
geologic environments. The removal of neptunium, technetium, and iodine render negligible the possibility of radioactive material penetration into the biosphere far in the future. Finally, removal of plutonium negates any incentive for future intrusion into repositories driven by overt or covert recovery of material for nuclear proliferation.

The complete process considers existing LWR spent fuel, separation processes, fuel fabrication, transmutation, disposal as a low-level waste (LLW), and the reprocessing of fuel after transmutation. This is an involved process that can be varied in a number of ways. Any proposed change to the process can have impacts on the fuel design, amount of waste generated by the process, number of cycles through the reactor, etc. In a nuclear growth scenario, the introduction of advanced thermal reactor designs will almost certainly result in changes in separations system requirements that must be met with optimized systems.

The separation process can be systematically identified as a group of blocks that have specific separation functions. One block’s effluent flows into another block as input. Each block has its process target. For a complex process like separation process, a systems engineering model is critical for designing and refining the whole chemical separation process.

However, most of the blocks (subprocesses) are still under development or revision. The research now is focused on the Uranium Extraction process (UREX). The process is for separation of TRU (Transuranic) elements from the dissolved spent fuel as shown in Figure 1-2. The process needs to report 95% of technetium (Tc) and 99.9% of uranium consistent to a separate effluent. The U/Tc–free TRU/ fission product (fp)
stream is then fed to PYRO-A process. The recovered uranium is purified for Low Level Waste (LLW) disposal while recovered technetium is used for transmutation of targets.

Figure 1-2 UREX Process

1.2 Objective

The project is to develop a system engineering modeling software package, TRPSEMPro (Transmutation Research Program System Engineering Model Project) that allows researchers to construct process blocks, connect blocks and analyze the chemical separation process in a systematic and optimized ways. Featured tasks include:

- Integrate chemical separation process into systems engineering model
- Identify core functions of system engineering model for the current or selected separation process scenarios
• Develop and implement software tool that allow to build system model and optimize the partial or entire chemical separation process
• Demonstrate software and model capabilities with various pre-defined scenarios

Due to the availability of the process modules, this research only demonstrates the systems engineering model involved interfacing with the UREX process that is calculated by Argonne Model for Universal Solvent Extraction (AMUSE) software package. The ANL developed AMUSE code, based on the Microsoft Excel macros, is identified as a standalone external module and kept intact. A midware-like Flowsheet-Simulator (Figure 1-3) is developed to interact with AMUSE package. The similar communication pathway can be built while more extraction modules, such as Pu/Np or Cs/Sr modules are available.

![Image of Project Tasks diagram]

Figure 1-3 Project Tasks
1.3 Capabilities and Features of the Model

- Couple simulation codes from multiple disciplines
- Easy to set up design problems
- DOE (Design of Experiments) - study and explore design space
- Optimization
- Combine the best features of existing optimization technologies
- Can use either single technique or a combination of techniques
- Provide Solution Viewer to view the design running results.

1.4 The Structure of this project

Chapter 2 focuses on identifying modules in chemical separation process and on building system engineering model based on the criteria; Chapter 3 will introduce core functions required for system engineering, such as Design of Experiment (DOE) and optimization; Chapter 4 discusses software design for TRPSEMPro where Model-View-Observer pattern and data structures are included. Chapter 4 also zooms into the integration module or interface between existing extraction process software and system engineering model. Chapter 5 introduces the interface module required for connecting any existing modules with the system engineering package. Current research demonstrates the interface module to the AMUSE code. Due to the sensitive nature of the UREX process, the provided AMUSE code is treated as a black box that reads in data and output results. Chapter 6 provides testing scenarios that primarily demonstrate the capabilities of the design software rather than the accurate simulation results from chemical separation process software AMUSE. Conclusions are made in the Chapter 7.
1.5 Software Implementation Tool

Software Implementation Tools include the following four parts:

1. **Microsoft™.Net** - Microsoft™.Net architecture allows programmers to develop components using a number of different programming languages, including C#, C++, Eiffel, J#, and Visual Basic. The resulting components (called “assemblies” by Microsoft) can use each other, regardless of the source language used to construct each one. Microsoft™.Net is the main programming language used in the software development (Deitel 2002, Microsoft 2002, Lowell Manuer 2002 and et al.).

2. **XML (eXtensible Markup Language)** - XML is intended to be a self-describing data format, allowing authors to define a set of element and attribute names that describe the content of a document. XML Schema was chosen as the method for describing the structure of system engineering model, also, XML Database is used to store all the run time data for AMUSE module (Kurt Cagle 2000, Mark Graves 2002, R. Allen Wyke 2002 and et al.).

3. **SQL Server 2000/MS ACCESS database**

4. **UML (Unified Modeling Language)** – UML is a graphical notation for expressing object-oriented designs. It is a melding of the notations of Booch, Rumbaugh, Jacobson, Wirf-Brock, and Herel among others. The official UML standard is managed by the Object Management Group consortium of companies (www.omg.org), and requires hundreds of pages to formally specify. Design models in this project are expressed by using UML (Booch 1999, James Rumbaugh 1999 and et al.).
CHAPTER 2

SYSTEM ENGINEERING

This project is primarily concerned with the development of system engineering model. Some background information will be discussed in this chapter. First, the definitions of system, systems engineering, and the need for systems engineering are discussed; Second, the concept and necessity for modeling are included. Third, the applications of system engineering techniques are discussed.

2.1 Introduction to System Engineering

2.1.1 The Need for the System Engineering?

System engineering is not just some academic exercise to appear knowledge. Three common reasons for project failure and disasters are listed below:

(1) Complexity. This is certainly one of the main causes of system problems. Brooks, often viewed as the father of software engineering and one of the great software philosophers (if such things exists), identifies two main types of complexity that exist in almost all systems (software and otherwise) (Brooks 1995): essential complexity and accidental complexity. Essential complexity is in the essence of the system. This means that it is an inherent part of a system and, as such, can not be eliminated. Accidental complexity is complexity that creeps into the system by accident. It is caused by accident or, to put it another way, by error, but it is possible to do something about it.
(2) Communications. A lack of communication or inefficient communication will contribute to project failure. The channels of communication may exist at an organization level, may exist between different teams within a single organization, between different resources, such as hardware, software, networks, protocols, etc.; may exist between different levels of management.

(3) Lack of understanding. These problems do not simply occur during the design and implementation of a project, but may appear at any phase in the project’s life cycle. Therefore, the way in which the program designers approach systems engineering must be applicable at any point in the project life cycle rather than simply focusing on a single phase or phase activity. This will turn out to be a fundamental requirement for system engineering.

2.1.2 What is a System?

What does the system actually mean? A common misconception is that a system is simply a product that may be delivered at the end of a project. However, this is not the case. A system is any process that converts inputs to outputs (Andrew P. Sage 2000). A system creates outputs based on inputs, over which it has no direct control, and the system’s present state. The current system state and a sequence of inputs allow computation of the future states of the system.

![Figure 2-1 Definition of System](image-url)
2.1.3 Definition of Systems Engineering

The term “systems engineering” means different things to different people. Here are three common definitions:

- **Definition 1**: John G. Truxal, former Dean of Engineering at Brooklyn Polytechnic Institute, says (William 1992), “System Engineering includes two parts: modeling, in which each element of the system and the criterion for measuring performance are described; and optimization, in which adjustable elements are set at values that gives the best possible performance.

- **Definition 2**: A division manager at Hughes Aircraft Company defined systems engineering as performing (William 1992): (a) requirements definition, (b) conceptual design, (c) partitioning of a system into subsystems (guidance, propulsion, etc) for other engineering teams to create, and (d) system validation, i.e., ensuring the system works when the subsystems are put together to form the system. Particular attention must be paid to the interface between the subsystems.

- **Definition 3**: System Engineering is concerned with defining and implementing an approach to solving problems, while managing complexity and communicating over the entire lifetime of a project (Jon Holt 2001).

2.2 Introduction to Modeling

2.2.1 Importance for Modelling

In order to justify the need for models, the easiest is to look at a number of simple examples. The examples used here are based on those defined by the modeling master, Grady Booch (Booch G. 1999).
Behind Booch’s three examples, the kennel, the house and the office block, there is a very serious and fundamental point.

Nobody in their right mind would attempt to build an office block with basic DIY skills. In addition, there is the question of resources, and not only in terms of the materials needed. In order to build an office block, you would need the knowledge to access the necessary human resources, plenty of time and plenty of money. The strange thing is that many people will approach building a complex system with the skills and resources of a kennel-builder, without actually knowing if it is a kennel, house or office block. When contemplation any complex system, you should assume that it will be, or has the potential to turn into, an office block building. Do not approach any project with a ‘kennel’ mentality. If you approach a project as if it was an office block and it turns out to be kennel, you will end up with a very well-made kennel that is the envy of all canines. If, however, you approach a project as if it was a kennel and it turns out to be an office block, the result will be pure disaster!

One of the reasons why it is so easy to misjudge the size and complexity of a project is that, in many cases, many elements of the system will not be tangible or comprehensible. Projects fail for many different reasons; there are three main themes, which have already been discussed: complexity, lack of understanding and communication. However, many projects do succeed. One reason is that they avoid, or minimize, the aforementioned problems due to effective modeling.

There are still other reasons for modeling. Here is the list for the purpose of modeling: (1) visualize a system; (2) specify a system; (3) serve as a template for creation; (4) in order to document decisions made throughout the project.
2.2.2 Definition of a Model

The definition of a model from Booch (Booch G 1999) is that: a model is defined as simplification of reality that is created in order to better understand the system under development, as we cannot comprehend complex systems.

2.2.3 Principles of Modeling

Booch (Booch G 1999) identifies four principles of modeling that are deemed crucial for successful and consistent modeling: the choice of model, the level of abstraction, connection to reality and independent views of the system.

1. Model choice: it will have a profound influence on how a problem is approached. Approaching a problem the right way can make a job much simpler and will be quicker than adapting the wrong approach;

2. Model Abstraction: ‘abstraction’ refers to the level of detail of a model. The point to be made here is that any model will require different levels of abstraction to be represented; otherwise the model will have little chance of being correct;

3. Connection to reality: One problem with modeling is that, according to the definition, models simplify reality. This means that some information must be lost somewhere along the line, which can cause problems. Therefore, it is vital to know both how the model relates to real life and how far it is divorced from real life. From a practical point of view, initial models tend to have quite a loose connection to reality and, as these models evolve, they get closer and closer to reality. The final connection to reality will be at the point when the
model actually becomes reality, which is when the model is implemented or constructed based on the model.

4. Independent views of the same system: A good model requires views modeled form different vantage points and requires views that represent different levels of abstraction. One crucial point that must be made here, however, is that each of these independent views must be consistent with one another, or, to put it another way, they must integrate correctly.

2.3 Implementation of System Engineering Techniques

Implementation of system engineering model is an approachable solution to solve complex process, such as chemical separation process. To help user to build a model on a system level and perform system analysis, the MCACM will develop a software tool that allows building system models through user-friendly interface. System engineering model itself includes analysis tools, such as parameter study, optimization, and Design of Experiments (DOE) study, especially designed for the chemical separation process. Schematically, the model development approach is shown in Figure 2-2.

![Figure 2-2 Schematic representation of the process for model development](image-url)
2.3.1 System Identification and Familiarization

The beginning of any system study is Block 1 in Figure 2-1, System Identification and Familiarization. Often, to a simple system, identifying the essential components of the system that collectively undergo the cause and effect action associated with the system is obvious, such as the illness (the output) that results when a person (the system) consumes toxic food or water (the input). However, the identification and isolation of other systems, such as a study of the causes of inflation where the general system is the world economic system, is undoubtedly complex, diverse, and presents a serious modeling challenge.

For this project, clearly defining the process flow sheet is a critical first step. Figure 1-1 shows one presentation of the proposed process.

2.3.2 Model Isolation and Boundary Setting

All of the present work centers on the three center blocks of Figure 2-1. The involved TRP chemical separation process can be systematically identified as a group of blocks that have specific separation functions: existing LWR spent fuel, separation processes, fuel fabrication, transmutation, disposal as a low-level waste (LLW), and the reprocessing of fuel after transmutation as listed in Figure 1-1. One block’s effluent flows into another block as input. Each block has its process target. Task in ANL is to develop each individual model for these chemical separation processes. Most of the blocks (sub processes) are still under development or revision. So far, research is focused on the Uranium Extraction process (UREX) block as shown in Figure 2-3.

The UREX process is for separation of TRU (Transuranic) elements from the dissolved spent fuel as shown in Figure 2-3. The process needs to report 95% of Tc and
99.9% of U to a separate effluent. The U/Tc–free TRU/Fp stream is then fed to PYRO-A process. The recovered Uranium is purified for low level waste (LLW) disposal while recovered technetium is used for transmutation of targets.

![UREX Process Diagram](image)

**Figure 2-3 UREX Process**

### 2.3.3 Analysis of System Equations

Each of the Process Blocks as shown in Figure 1-1 will have a set of equations or relationships to model the transport of mass through that process. These relationships may be relatively simple, or complex computer codes like the AMUSE code. While chemical engineering systems are getting more complex, the process becomes more difficult to analyze mathematically. The development of Systems Engineering Model allows industries to model the process more quantitatively and to study the interactions between subsystems and performance of the model under the influence of various design parameters. Systems engineering is a multidisciplinary function dedicated to controlling design so that all elements are integrated to provide an optimum, overall system. Detailed process flowchart will be elaborated in Chapter 3.
Developing a systems engineering model of the overall chemical separation process would be beneficial to analyzing complex interactions between proposed process changes. The model will evolve over several years to incorporate all process steps and to improve process modules as more knowledge is gained.

The graphical representation of the systems engineering model, including inputs, outputs and possible feedbacks, is a very useful tool in the initial modeling and formulation stage of a system study. The act of graphically and schematically portraying the system is conducive to accurate identification and improved understanding of what inputs interact with the system components and how these interactions produce the outputs anticipated. It is in the graphical representation stage of modeling that the system investigator or apprentice could be as thorough and critical of all the known or anticipated system factors as possible. The investigator or apprentice could attempt to detail the system and individually “componentize” the system elements as much as possible.

Frequently, a very valuable aid in the initial identification of the inputs and outputs and various subsystems of a given system is to graphically model the overall system, including explicit designation of all subsystems and internal inputs and outputs. The powerful influence of visual and special conception and recognition of the model in a graphic format can be very revealing and productive for both model analysis and synthesis.

To UREX process, the visual definition can be performed in program module AMUSESimulator, and user can actively define inputs/outputs of this process module in
In TRPSEMPro, design problems are specified, and simulation codes from multiple disciplines are coupled into a system model description file written in XML. After a description file is created, the user can use the application interface to set up, monitor, and analyze a design run.

A few critical components are required during the system modeling, input/output parameters, relationships among blocks (such as sequence of execution of extraction processes), system analysis, and process monitoring.

The designed system model includes four main parts as shown in Figure 2-4.

1. System Manager – System Manager is the main application interface, from where a user can launch any of the application interfaces. The System Manager allows the user to set up and run a design problem.

2. Model Integration – Model Integration enables user to couple simulation programs to system engineering model and specify their execution sequence.
Model Integration provides a GUI that acts as a front end for creating an system engineering model description file written in XML language.

3. StudyPlan – StudyPlan provides a convenient means to provide problem formulation information to specific design parameters, the allowing user to control information in a specific problem. Techniques such as Optimization and Design of Experiments (DOE) are available in Study Plan.

4. Solution Viewer – Solution Viewer provides a visual means to monitor the optimization process as it moves through the design space. Solution Viewer provides tables and graphs that can be used to view the runtime changes.

![Figure 2-4 System Architecture for TRPSEMPPro](image)

2.5 TRPSEMPPro’s Role in the Design Cycle

The typical practice design engineers employ in either product or process design is known as the design-evaluate-redesign cycle. As shown in Figure 2-5, this cycle involves
the iterative processing of input files, the running of one or more simulation programs, and the analysis of output files. The cycle continues until the design criteria are reached and the best design (one that satisfies the customer requirements and meets design constraints) is chosen by the design engineer.

The role of TRPSEMPro is to automate the design-evaluate-redesign cycle, eliminate the human intervention bottleneck. In Figure 2-5, the Study Plan (a single, self-contained design unit with its own parameters, optimization strategy) drives the automation of the cycle. By eliminating costly human intervention, TRPSEMPro lets the designer refocus valuable engineering time on design analysis and selective refinement of the design process.

TRPSEMPro system can be harnessed to execute more than just computer simulation programs. TRPSEMPro can automate the execution of a sub model (independent models that are invoked by the main model) to support a hierarchical, multilevel design problems.
CHAPTER 3

DOE STUDY AND OPTIMIZATION

Knowledge of the DOE Study and optimization are discussed here. After integrating all individual models into the system engineering model, the DOE and optimization technologies are core to the system analysis.

3.1 Design of Experiments (DOE) Study

Design Of Experiment (DOE) is a general term that refers to any of the many formal methods available for setting parameter values in a set of experiments.

3.1.1 Purposes of Using Design of Experiments (DOE) Study

The purpose of DOE study is listed as follows:

- To assess design variable impact
- To identify significant design variable interactions
- To analyze a design space and provide a rough estimate of an optimal design, which can be used as a starting point for numerical optimization
- To screen a large set of design variables for a subsequent numerical optimization (design space reduction)

To do the DOE study, you should know the following things:

- Parameters you want to study (i.e., the parameters that may impact performance/quality)
• Number of experiments or trials you can afford to run

• Output quantities of interest (i.e., those that in some way represent the performance/quality of the product)

• Overall goal of the study (i.e., design variable screening, estimation of an optimal design, response surface approximation)

### 3.1.2 DOE Study Techniques

Several techniques about DOE study have been studied, these include Full-Factorial, Parameter Study, Data File, Orthogonal Arrays, Central Composite and Latin Hypercubes (Liu 1993, Guri 1994).

Full-Factorial Design: In a full-factorial study, the user specifies the values at which to study each parameter and then to run all combinations of these values. If \( i \) is the number of factors and \( n \) is the number of levels or values at which to study each factor, then a total of \( \prod n_i \) designs must be evaluated.

Parameter study: The Parameter Study allows one parameter at a time to be studied at different values, with all other parameters held at their base value. After a parameter has been studied at each of the specified values, it is restored to its base value for studying the other parameters. It is similar to a sensitivity analysis. In this case, if \( i \) is the number of parameters and \( n \) is the number of levels at which to study each parameter, then the total number of designs to be evaluated is \( 1 + \sum n_i \), where the one additional evaluation is for the baseline design.

The matrix below illustrates the difference between the Full-Factorial Study (Figure 3-1) and the Parameter-Study (Figure 3-2). In this example, three factors were shown— A,
B, and C. Each is to be studied at two levels – 1 or 2. The baseline design values for the parameter study are designated by the level 0.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 3-1 Full-Factorial Study**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**Figure 3-2 Parameter Study**

Data File: With the Data File study, it is possible to analyze design points in a tab- or space- delimited file by associating the columns of the data file with the user specified parameters.

Orthogonal Arrays: The use of orthogonal arrays let you avoid a costly full-factorial experiment in which all combinations of all inputs (or factors) at different levels are studied (p*n for n factors each at p levels), and instead perform a fractional factorial
experiment. A fractional factorial experiment is a certain fractional subset (1/2, ¼, 1/8, etc.) of the full factorial set of experiments, carefully selected to maintain orthogonality (independence) among the various factors and certain interactions. It is this orthogonality that allows for independent estimation of factor and interaction effects from the entire set of experimental results.

Central Composite Design: Central Composite Design (CCD) is a statistically-based technique in which a 2-level full-factorial experiment is augmented with a center point and two additional points for each factor (called “star points”). Thus, five levels are defined for each factor, and to study n factors using Central Composite Design requires $2^n + 2^n + 1$ design point evaluations. Figure 3-3 shows the Central Composite Design points for three factors.

The center and star points are added to acquire knowledge from regions of the design space inside and outside the 2-level full factorial points, allowing for an estimation of higher-order effects (curvature). The star point(s) are determined by defining a parameter which relates these points to the full-factorial points by:

$$S_{upper} = b + (u - b) \times a$$
$$S_{lower} = b - (b - 1) \times a$$

where:

$b = baseline\ design = lower\ factorial\ point$
$u = upper\ factorial\ point < b < u$

Note that:

$a < 1$, star points are inside the full-factorial design

$a > 1$, star points are outside the full-factorial design

$a = 1$, star points are the same value as the full-factorial levels

(also referred to as face-centered central composite design)
Although Central Composite Design requires a significant number of design point evaluations, it is a popular technique for compiling data for Responses Surface Modeling due to the expense of design space covered, and the higher-order information obtained.

Latin Hypercubes: Another class of experimental design which efficiently samples large design spaces is Latin Hypercube sampling. With this technique, the design space for each factor is uniformly divided (the same number of divisions (n) for all factors). These levels are then randomly combined to specify n points defining the design matrix (each level of a factor is studied only once). For example, the figure illustrates a possible Latin Hypercube configuration for two factors (X1, X2) in which five points are studied. Although not as visually obvious, this concept easily extends to multiple dimensions.

An advantage of using Latin Hypercubes over Orthogonal Arrays is that more points and more combinations can be studied for each factor. The Latin Hypercube technique allows the designer total freedom in selecting the number of designs to run (as long as it
is greater than the number factors). The configurations, however, are more restrictive using the Orthogonal Arrays.

3.2 Optimization

3.2.1 Introduction to Optimization

One of the primary tasks of the systems engineer is to ensure the optimization of the design process. Driven by competition, quality assurance, cost of production, and finally, the success of the business enterprise, the subject of optimization is receiving serious attention from engineers, scientists, managers, and most everybody else. Now, optimization is practiced through software programs and requires significant computer resources. The techniques of optimization have not changed significantly in recent years, but the areas of application in professional practice require at least three prerequisites. They include mathematical modeling of the design problem, knowledge of computer programming, and knowledge of optimization techniques.

Optimization can be applied to all disciplines. Qualitatively, this assertion implies multiple decision choices; implicitly recognizing the necessity of choosing among alternatives. In this project, optimization issue is approached in a quantitative way which means that an outcome of applying optimization to the problem, design, or service must yield numbers that will define the solution, or in other words, numbers or values that will characterize the particular design or service. Quantitative description of the solution requires a quantitative description of the problem itself. This description is called a mathematical model. The design, its characterization, and its circumstances must be expressed mathematically.
3.2.2 Formal Elements of Optimization Problem

To go further in the area of optimization, the formal elements of the optimization problem must be introduced first. It should be understood that optimization presupposes the knowledge of the design rules for the specific problem, primarily the ability to describe the design in mathematical terms. These terms include design variables, design parameters, and design functions. Traditional design practice, that is, design without regard to optimization, includes all of these elements although they were not formally recognized as such. This also justifies the prerequisite that you must be capable of designing the object if you are planning to apply the techniques of optimization. It is also good to recognize that optimization is a procedure for searching the best design among candidates, each of which can produce an acceptable product.

Design variables are entities that identify a particular design. In the search for the optimal design, these entities will change over a prescribed range. The values of a complete set of these variables characterize a specific design. The number and type of entities belonging to this set are very important in identifying and setting up the quantitative design problem.

Design Parameters, in this project, identify constants that will not change as different designs are compared.

Design functions define meaningful information about the design. They are evaluated using the design variables and design parameters. They establish the mathematical model of the design problem. These functions can represent design objective(s) and/or constraints. Design objective drives the search for the optimal design. The satisfaction of
the constraints established the validity of the design. The designer is responsible for identifying the objectives and constraints.

General abstract mathematical model is summarized as following:

Minimize:

\[ f(x_1, x_2, \ldots, x_n) \]  \hspace{1cm} (3.1)

Subject to:

\[
\begin{align*}
    h_1(x_1, x_2, \ldots, x_n) &= 0 \\
    h_2(x_1, x_2, \ldots, x_n) &= 0 \\
    \cdots \\
    h_i(x_1, x_2, \ldots, x_n) &= 0
\end{align*}
\]

\[ x_i^l \leq x_i \leq x_i^u, i = 1, 2, \ldots, n \]  \hspace{1cm} (3.2)

Exploiting vector notation the mathematical model is:

Minimize:

\[ f(X), [X]_a \]  \hspace{1cm} (3.3)

Subject to:

\[
\begin{align*}
    [h(X)]_l &= 0 \\
    [g(X)]_m &= 0
\end{align*}
\]

\[ X^\text{low} \leq X \leq X^\text{up} \]  \hspace{1cm} (3.4)

The above mathematical model expressed the following standard format of the optimization problem expressed in natural language:

Minimize the objective function \( f \), subject to \( l \) equality constraints, \( m \) inequality constraints, with the \( n \) design variables lying between prescribed lower and upper limits.

The search of the optimal solution will depend on the nature of the problem being solved.
3.2.3 Category of Optimization Problem

The category of Optimization Problem is shown in Figure 3-4 (Edwin 1996, Boris 1987).

![Figure 3-4 Category of Optimization Problem]

3.2.4 Basic Mathematical Concepts

The basic mathematical elements in the discussion of Nonlinear Programming (NLP) are derivatives, partial derivatives, vectors, matrices, Jacobian, and Hessian (David 1978).

Numerical derivative computation: Many numerical techniques in NLP require the computation of derivatives. The derivative for the single-variable function at any value $x$ is also called slope or the gradient of the function at that point. To two or more independent variables the equivalent concept is the partial derivative. Partial derivative is defined for each independent variable.
\[
\frac{\partial f}{\partial x} \bigg|_{(x,y)} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h} \tag{3.5}
\]

The gradient of the function is a vector, and at any point represents the direction in which the function will increase most rapidly. Examining the conventional objective of NLP, minimization of objective function, the gradients has a natural part to play in the development of methods to solve the problem. The gradient is composed of the partial derivatives organized as a vector.

The Jacobian \([J]\) defines a useful way to organize the gradients of several functions. Using three variables and two functions \(f(x,y,z)\) and \(g(x,y,z)\), the definition of the Jacobian is:

\[
\begin{bmatrix}
\frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial z} \\
\frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} & \frac{\partial g}{\partial z}
\end{bmatrix}
\tag{3.6}
\]

The first row is the gradient of \([df, dg]^T\) while the second row is the gradient of \(g\). If the two functions are collected into a column vector, the differential \([df, dg]^T\) in the functions, due to the differential change in the variables\([dx, dy, dz]\), can be expressed as a matrix multiplication using the Jacobian

\[
\begin{bmatrix} df \\ dg \end{bmatrix} = [J] \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \tag{3.7}
\]

Hessian matrix \([H]\) is the same as the matrix of second derivative of a function of several variables. For \(f(x,y)\):

\[
\frac{\partial^2 f}{\partial x^2} = \frac{\partial^2 f}{\partial y^2} = \frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}
\]
\[
[H] = \begin{bmatrix}
\frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\
\frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2}
\end{bmatrix}
\] (3.8)

For a function of \( n \) variables \( f(X) \), where \( X = [x_1, x_2, \ldots, x_n]^{T} \), the gradient is

\[
\nabla f = \begin{bmatrix}
\frac{\partial f}{\partial x_1} \\
\frac{\partial f}{\partial x_2} \\
\vdots \\
\frac{\partial f}{\partial x_n}
\end{bmatrix}^{T}
\] (3.9)

The Hessian is

\[
[H] = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2}
\end{bmatrix}
\] (3.10)

Taylor’s Theorem/Series is a useful mechanism to approximate the value of the function \( f(X) \) at the point \( (X_p + \Delta X) \) if the function is completely known at current point \( X_p \) (\( \Delta X \) is the displacement vector),

\[
f(X_p + \Delta X) = f(X_p) + \nabla f(X_p)^T \Delta X + \frac{1}{2} \Delta X^{T} H(X_p) \Delta X + \cdots
\] (3.11)

3.2.4.1 Analytical Conditions for General Optimization Problem:

Optimization problems whose mathematical models are characterized by nonlinear equations are called Nonlinear Programming (NLP) problems. Engineering design problems are mostly nonlinear. Here, a traditionally bottom-up presentation of knowledge for nonlinear optimization will be given as follows (David 1978 and Donald 1986). Unconstrained problems are discussed first followed by constrained problems.
Unconstrained Problem: First-Order Conditions (FOC): FOC are usually regarded as necessary conditions.

\[ \nabla f(X^*) = 0 \quad (3.12) \]

Second-Order Conditions (SOC): SOC are usually regarded as sufficient conditions. It can be inferred that these conditions will involve second derivatives of the function. The SOC is often obtained through the Taylor expansion of the function to second order. If \( X^* \) is the solution, and \( \Delta X \) represents the change of the variables from the optimal value which will yield a change \( \Delta f \), then

\[ \Delta f = f(X^* + \Delta X) - f(X^*) = \nabla f(X^*)^T \Delta X + \frac{1}{2} \Delta X^T H(X^*) \Delta X \quad (3.13) \]

Since \( \Delta f \geq 0 \) and Equation (3.12), then, \( \Delta f = \frac{1}{2} \Delta X^T H(X^*) \Delta X > 0 \)

Where \( H(X^*) \) is the Hessian matrix (the matrix of second derivatives) of the function \( f \) at the possible optimum value \( X^* \). For the relation in equation to hold, the matrix \( H(X^*) \) must be positive definite.

Equality Constrained Problem:

Minimize \[ f(X), X \]

Subject to:

\[ \begin{align*}
[h(X)]_l &= 0 \\
X^\text{low} &\leq X \leq X^\text{up}
\end{align*} \quad (3.15) \]

Lagrange Multipliers method is an elegant formulation to obtain the solution to a constrained problem.
Inequality Constrained Optimization: An inequality constrained problem can be transformed to an equality constrained problem by introducing a slack variable for each inequality constraint, then the solution can be found.

Scaling: Numerical calculations are driven by larger magnitudes. The standard approach to minimize the impact of large variations in magnitudes among different equations is to normalize the relations. In practice this is also extended to the variables. This is referred to as scaling the variables and scaling the functions. Many current software will scale the problem without user intervention.

The presence of side constraints in problem formulation allows a natural definition of scaled variables. The user-defined upper and lower bounds are used to scale each variable between 1 and 0. Therefore,

$$ x_i = \frac{x_i - x_i^l}{x_i^u - x_i^l}; \quad x_i^s = scaled \; i^{th} \; variable $$

$$ x_i = x_i^s (x_i^u - x_i^l) + x_i^l $$

3.2.5 Numerical Techniques for Nonlinear Programming Problem

Following sections will illustrate several numerical techniques for multivariable optimization problems which are implemented in our developed software system (Donald 1986, User Guide: Optimization Toolbox for Use With MATLAB 2000). While the unconstrained optimization is not a common occurrence in engineering design, nevertheless the numerical techniques included here demonstrate interesting ideas and also provide the means to solve constrained problems after they have been transformed into an unconstrained one.
3.2.5.1 Unconstrained Optimization

All the algorithms which can be used to solve unconstrained optimization problems are iterative; the iteration takes place by moving along a search direction vector during an iteration. These search directions can be determined in many ways. The different techniques that are presented here primarily differ in how the search direction is established. The remaining elements of the algorithm, except for the convergence/stopping criteria, are the same for almost all of the methods; the generic algorithm can be expressed as shown in Figure 3-5.

![Flowchart](image)

**Figure 3-5 Generic Algorithm for Unconstrained Optimization**
Random Walk Method: Random Walk is called a zero-order method. In this method, the search direction from each iteration is a random direction (Please refer to Figure 6-20, Figure 6-21, Figure 6-25, and Figure 6-26 for examples of using this method).

Conjugate Gradient (Fletcher-Reeves) Method: Originally due to Fletcher and Reeves, this method has the property of quadratic convergence because the search directions are conjugate with respect the Hessian matrix at the solution. A quadratic problem in n variables will converge in no more than n variables. Figure 3-7 shows the algorithm for Conjugate Gradient method (Please refer Figure 6-23 and Figure 6-24 for examples of using this method). Davidon-Fletcher-Powell Method (DFP): DFP method belongs to the
family of Variable Metric Methods (VMM). It was first introduced by Davidon and several years later was developed in its current form by Fletcher and Powell. The Conjugate Gradient method’s improvement over the Steepest Descent method was possible because of the inclusion of the history from the previous iteration. In DFP method the history from all previous iterations is available. This information is collected in an $n \times n$ matrix called the metric. The metric is updated with each iteration and is used to establish the search direction. An initial choice for the metric is also required. It must be a symmetric positive definite matrix. For the method to converge, the metric must hold on to its positive definite property through the iterations. In the DFP method, the metric approaches the inverse of the Hessian at the solution.

Figure 3-8 shows the algorithm for this method (Please refer Figure 6-28 to see an example of using this method).

3.2.5.2 Constrained Optimization:

The algorithms to Constrained Optimization have two outcomes that the algorithms seek to accomplish. The first is to ensure that the design is feasible (satisfies all constraints) and the second that it is optimal (satisfies the Kuhn-Tucker conditions). Two distinct approaches will be used to handle the constrained optimization problem. The first approach is termed the indirect approach and solves the problem by transforming it into an unconstrained problem. The second approach is to handle the constraints without transformation – the direct approach. So far, the indirect approach is implemented in TRPSEMPro.
Choose $X_1, N$(number of iterations)

$f_s(i) = f(X_i); X_s(i) = X_i$(store values)

$\epsilon_1, \epsilon_2, \epsilon_3$: (tolerence for stopping criteria)

\[ S_i = -\nabla f(X_i) \]

\[ \beta = \frac{\nabla f(X_i) \cdot \nabla f(X_{i-1})}{\nabla f(X_{i-1}) \cdot \nabla f(X_{i-1})} \]

\[ S_i = -\nabla f(X_i) + \beta S_{i-1} \]

$X_{i+1} = X_i + \alpha_i S_i$

$\alpha_i$ is determined by minimizing $f(X_{i+1})$

$X_s(i+1) \leftarrow X_{i+1}; f_s(i+1) = f(X_{i+1})$ (Store values)

\[ \Delta f = f_s(i+1) - f_s(i) ; \Delta X = X_s(i+1) - X_s(i) \]

\[ |\Delta f| \leq \epsilon_1 ? \]

\[ \nabla f(X_{i+1}) \cdot \nabla f(X_{i+1}) \leq \epsilon_1 ? \]

\[ \Delta X^T \Delta X \leq \epsilon_1 ? \]

\[ i+1 = N \]

\[ \nabla f(X_{i+1}) \cdot \nabla f(X_{i+1}) \leq \epsilon_1 ? \]

Figure 3-7 Algorithm - Conjugate Gradient (Fletcher-Reeves) Method
Choose $X_{i,1}, [A]$ (initial metric), $N$

$\varepsilon_1, \varepsilon_2, \varepsilon_3$ : (tolerance for stopping criteria)

Set $i = 1$ (initialize iteration counter)

$S_i = -[A] \nabla f(X_i)$

$X_{i+1} = X_i + \alpha_i S_i$; $\Delta X = \alpha_i S_i$

$\alpha_i$ is determined by minimizing $f(X_{i+1})$

$\nabla f(X_{i+1})^T \nabla f(X_{i+1}) \leq \varepsilon_3$?

YES (Converged) NO

YES (function not changing) NO

$|f(X_{i+1}) - f(X_i)| \leq \varepsilon_3$?

YES (design not changing) NO

$\Delta X^T \Delta X \leq \varepsilon_2$?

YES (iteration limit) NO

$i + 1 \leq N$?

YES NO

$Y = \nabla f(X_{i+1}) - \nabla f(X_i)$

$Z = [A] Y$

$[B] = \frac{\Delta X \Delta X^T}{\Delta X^T Y}$

$[C] = -\frac{ZZ^T}{Y^T Z}$

$[A_{i+1}] = [A] + [B] + [C]$

$i = i + 1$

Stop

Figure 3-8 Algorithm - Davidon-Fletcher-Powell Method (DFP)
Augmented Lagrange Multiplier (ALM) method: The ALM method is the most robust of the penalty function methods. More importantly it also provides information on the Lagrange multipliers at the solution. This is achieved by not solving for the multipliers but merely updating successive SUMT (Sequential Unconstrained Minimization Techniques) iterations. Figure 3-9 shows the algorithm and Figure 6-28 shows an example of using this method.

The general optimization problem is transformed to unconstrained problem as shown in Equation (3.18) – (3.19).

Minimize:

\[
F(X, \lambda, \beta, r_h, r_g) = f(X) + r_h \sum_{k=1}^{l} h_k(X)^2 + \\
\quad + r_g \sum_{j=1}^{m} \max \left[ g_j(X), -\frac{\beta_j}{2r_g} \right]^2 + \\
\quad + \sum_{k=1}^{l} \lambda_k h_k + \sum_{j=1}^{m} \beta_j \left\{ \max \left[ g_j(X), -\frac{\beta_j}{2r_g} \right] \right\}
\]

Where:

\[
x_i \leq x_i^* \quad i = 1, 2, \cdots, n
\]

Here \( \lambda \) is the multiplier vector tied to the equality constraints, \( \beta \) is the multiplier vector associated with the inequality constraints; and \( r_h \) and \( r_g \) are the penalty multipliers.

\( F \) is solved as an unconstrained function for predetermined values of \( \lambda, \beta, r_h \) and \( r_g \).

Therefore, the solution for each SUMT iteration can be shown as Equation 3.20.

\[
X^* = X^*(\lambda, \beta, r_h, r_g)
\]
Choose $X^0$, $N$, (no of SUMT iterations)
$N_e$ (no of DFP iterations)
$\varepsilon,\alpha$ (for convergence and stopping)
$r^e, r^g$ (initial penalty multipliers)
$c_\varepsilon, c_\alpha$ (scaling value for multipliers)
$\lambda^0, \beta^0$ (initial multiplier vectors)
$q = 1$ (SUMT iteration counter)

Call DFP to minimize $F(X^0, \lambda^0, \beta^0, r^e, r^g)$
Output: $X^0$

If:
1. $h_k = 0$ (for $k = 1, \cdots, I$) 
2. $g_j \leq 0$ (for $j = 1, \cdots, m$) 
3. $\beta_j > 0$ (for $g_j = 0$) 
4. all side constraints are satisfied
Then: Converged = true
Else: Converged = false

Converged = True?

YES

NO

YES

$(\Delta F)^2 \leq \varepsilon^2$?

NO

YES

$\Delta X = X^q - X^{(q-1)}$

NO

YES

$\Delta X^T \Delta X \leq \varepsilon^2$?

NO

YES

$q = N_e$?

NO

NO

$\Delta F = F_q - F_{q-1}$

$q \leftarrow q + 1$

$\lambda^q \leftarrow \lambda^q + 2r^e_k h(X^q)$

$\beta^q \leftarrow \beta^q + 2r^\alpha_k (\max\{g(X^q), -\beta^q / 2r^\alpha_k\})$

$r^e_q \leftarrow r^e_{q-1} + r^q \leftarrow r^g_q C_{\varepsilon}$

$q \leftarrow q + 1$

$X^q \leftarrow X^q$

Stop

Figure 3-9 Algorithm - Augmented Lagrange Multiplier (ALM) method
CHAPTER 4

SOFTWARE DESIGN FOR DEVELOPING ENGINEERING MODEL

Main issues about software engineering techniques and the concept of software design patterns are introduced. Since the software tool is based on the Object-Oriented Analysis and Design (OOA&D), detailed OO approach of designing TRPSEMPro is discussed. Design models are expressed by using Unified Modeling Language (UML).

4.1 Introduction to Software Engineering

4.1.1 System and Software Engineering Relationship

Computers are now an integral part of almost any complex system and generally involve software of some description. Software is an unpredictable compartment that needs to be contained by applying software engineering techniques that increase the chance of project success enormously. Software engineering is a subset of system engineering that limits itself to a single discipline. Clearly, there are many similarities between the two disciplines and it is necessary to identify their importance through any available sources.

4.1.2 The Phase of a Software Process

The main phases of a software process are listed in Figure 4-1 (Hans 2000). The Requirements analysis specifies what the application must perform, and answer the question “WHAT?”; Design specifies what the parts will be, and how they will fit
together; while Implementation means code writing; and Test and Maintenance are a continuous process for repairing defects and adding/removing components.

![Figure 4-1 Phases of Software Process](image)

This software process in Figure 4-1 is referred to as the waterfall. And it is more of an ideal or baseline than a realistic process. The spiral process, in which the waterfall is traversed several times, is commonly used in many forms. Each pass through the waterfall produces an intermediate product more capable than the previous one, until the derivable product is produced.

4.1.3 Requirements Analysis

Requirements analysis is the process of understanding, and putting in writing, a statement of what the application is intended to do once it has been built. The difficulties in creating a requirements document are: using appropriate ways to express the requirements, organizing them, and managing them over time. The requirement is expressed in ordinary, clear, non-technical English, from the user’s perspective; organize the requirements into logical grouping, make it easy to access and change (Jeffrey 1993).

4.1.4 Software Design

A “software design” is a set of documents on whose basis a software application can be fully programmed. In other words, a complete software design should be so explicit that a programmer could code the application from it without the need for any other
documents. Software designs are like the blueprints of a building, which are sufficient for a contractor to build the required building.

The main goals of a software design are sufficiency, robustness, flexibility, reusability and efficiency (Hans 2000 and Cay 2004).

**Sufficiency** is the goal to satisfy the requirements for the application. **Modularization** is a key way to assess the design sufficiency: The more specific a question, the more precisely verification of the correctness from the design can be answered; **robustness** makes the application to be able to handle miscellaneous and unusual conditions such as bad data, user error, programmer error, and environmental conditions; the requirements of an application can change in many ways. For example, obtaining more or less of what’s already present, adding new kinds of functionality, changing functionality and so on; **flexibility** is to satisfy those changing with minimum efforts of code revision.

Engineering consists of the creation of useful products with given standards of quality at minimal cost; the trend in software is to reuse parts among applications – the goal of reusability; **efficiency** means to create designs and implementations that are as fast as required, and which make use of no more than the available memory.

To achieve the sufficiency goal, modularizing design and reusing trusted parts need to be done; to achieve robustness goal, identification of reliable designs and robust parts is required; to achieve reusability and flexibility goal, keeping code at a general level and minimize dependency on other class are important.

4.1.5 Design Patterns in Software Development

Experienced object-oriented developers (and other software developers) build up a repertoire of both general principles and idiomatic solutions that guide them in the
creation of software. This repertoire is the design pattern – class combinations and accompanying algorithms that fulfill common design purposes. A design pattern expressed an idea rather than a fixed class combination. Accompanying algorithms express the pattern’s basic operation.

Gamma et al. (Gamma 1999) have classified each of the design patterns in one of three categories as listed follows.

- Structural patterns. Representing a collection of related objects.
- Behavioral patterns. Capturing behavior among a collection of objects.

4.2 Object-Oriented Analysis and Design (OOA&D)

OOA&D is a way of specifying and designing applications, which exploits OO technology. Its main characteristic is to approach the project under terms that occur naturally in the application. These terms are, in effect, the ingredients.

Whereas pre-OO approaches to application design and development emphasized functionality, the OOA&D approach emphasizes ingredients. In other words, “what is this application about?” The individual words (usually nouns) that answer this question are called the domain\(^2\) classes of the application, and they appear as classes in the designs and implementation.

Usually, the use cases may not be sufficient for arriving at all of the domain classes. A brain storming process maybe required to complete the list (Cay 2004).

---

\(^2\) There are two kinds of classes used in designs: domain classes, which pertain to the specific application under design, and non domain classes, which are not special to this application but may be useful for this application.
Figure 4-2 shows a standard way to go about effecting Object-Oriented Analysis and Design. The process realizes a major benefit of Object-Orientation. It creates a clean mapping from the very beginning between the real world, in the form of requirements, and the design, in the forms of classes. The domain classes are keys in realizing this mapping.

The rest of this chapter will discuss the build of the whole application with Object-Oriented technology.

![OOA&D Roadmap](image)
4.3 Requirement Analysis and Domain Classes Obtainment

The process to determine the domain classes is started with use cases, converting them to sequence diagrams, and selecting the classes that appear at the top of the sequence diagrams as shown in Figure 4-3.

![Diagram showing the process of obtaining domain classes]

Figure 4-3 OOA&D Approach for Obtaining Domain Classes

User can integrate any number, any kind of chemical separation models or some utility model into the whole system model as far as these models satisfy some conditions, like: they implement the defined interface, or it is some kind of simucode program which have input file, output file, executable program file. To customize user’s system model, different system analysis strategies can be applied. Users can view the system analysis process both in graphical and/or tabular ways. Users can monitor the generated results in real time fashion. User can specify their parameters of interest from designed interface. The following are key use cases for the development of a systems engineering model.

4.3.1 High Level Use Case

In the high level, there are five use cases as shown in Figure 4-4.

- System initializes the system engineering model.
- User integrates all the required models
- User builds the study plans
- User builds the solution views
- User executes the study plans which have been built.

According the use cases in Figure 4-4, following sequence diagram shows detailed information for each use case, as shown in Figure 4-5.
4.3.2 Middle Level Use Case

Each use case in last section can be further divided into more detailed sub use cases.
4.3.2.1 Use Cases for ModelIntegration

Use cases for model integration include four cases as shown in Figure 4-6.

- System initializes the environment;
- User Adds/Removes models;
- User configures model properties;
- User configures parameter properties for each model, including changing values and parameter mapping.

Figure 4-6 Use Case for Model Integration

From the use cases shown in Figure 4-6, the corresponding sequence diagram shows detailed information for each use case in Figure 4-6. The sequence diagram is shown in Figure 4-7.
Figure 4-7 Sequence Diagram for ModelIntegration
4.3.2.2 Use case for StudyPlanBuilder:

There are four use cases for StudyPlanBuilder, as shown in Figure 4-8.

- User configures objective and constraint
- User configures study plan. Study plan can be extended to two use cases, Design of Experiment (DOE) and Optimization study plans.

![Figure 4-8 Use case for StudyPlanBuilder](image)

From the above use cases, the sequence diagram for StudyPlanBuilder shows the detailed information in Figure 4-9.
Figure 4-9 Sequence Diagram for StudyPlanBuilder

4.3.2.3 Use case for SolutionViewer

There are five use cases for SolutionViewer as shown in Figure 4-10.

- System initializes SolutionViewer
- User configures graph page, including adding/removing/modifying graph page
- User configures TableGraph, which can be extended to two use cases, Configure Graph and Configure Table.
From the above use cases, the corresponding sequence diagram shows the detail information of each of use cases in SolutionViewer. Figure 4-11 shows this sequence diagram.
4.3.3 Other Issues about Obtaining Domain Class

To get the domain class, sequence diagrams are not the only sources, because some requirements do not fit within use cases; the most important source for domain classes is the set of nouns mentioned in background documents for the application.
Once candidate domain classes have been selected, they should be cut back to just the necessary ones. In fact, it is better to have too few classes at the end of this process than too many. The reason is that it’s much easier to add a class when needed than to remove a class that has become embedded in a design and implementation.

4.3.4 List of Domain Class

For the list of domain class, reader can refer to the detailed design part in section 4.5.

4.4 System Architectures

While architecture is design at a high level, the sufficiency and flexibility can be better dealt with than robustness and efficiency can. Robustness and efficiency are usually better handled at lower levels of design.

With enough practice, it is not hard to write small programs: large applications, however, present very different problems, and are difficult to create. The principle problem of software systems is complexity – not the number of classes or lines of code per se, but their interrelationship. A very good weapon against complexity is decomposing the problem so that the result has the characteristics of small programs. For this reason, decomposing (or modularizing) the problem is of critical importance, and one of the exciting design challenges. The designer should form a clear mental model of how the application will work at a high level, and then develop a decomposition to match this mental model. This process is sometimes called recursive design because it repeats the design process at successive scales.

Effective modularization is accomplished by maximizing cohesion and minimizing coupling. This principle helps to decompose complex tasks into simpler ones. Cohesion within a module is the degree to which communication takes place among the module’s
elements; coupling describes the degree to which modules depend directly on other modules. Figure 4-12 shows the System Architecture for TRPSEMPro.

Figure 4-12  System Architecture

4.5 Detailed Design for TRPSEMPro

In this section, detailed design for TRPSEMPro is discussed. Each package shown in Figure 4-12 will be given a detailed static class diagram. The application of design pattern will be discussed here also.

4.5.1 Design for System Engineering DataModel

Figure 4-13 shows the class diagram for main data model of TRPSEMPro. In package DataModel, in order to incorporate more than one simulation program, to represent trees of object, composite design pattern are used (Robert 2003, Mark 1998). A recursive form
to represent objects in which the tree class aggregates and inherits from the base class of the objects, is used. TaskBase is the base class while class ChemicalProcess, Calculation, and Simcode are LeafNode which don’t aggregate other objects. Class TaskProcess is NonLeafNode that can aggregate other objects.

Client of DataModel, such as SystemManager, can reference any of the objects in the tree, and cause the object and all of its subsidiaries to perform some common actions, such as singleRun().

Figure 4-13 Static Diagram for System Engineering Data Model

From the above static diagram, Figure 4-14 shows this sequence diagram for

Composite Design Pattern in DataModel
4.5.2 Design for ModelIntegration

Figure 4-15 shows the static class diagram for graph drawing in ModelIntegration.

Figure 4-14 Sequence Diagram for Composite Design Pattern in DataModel

Figure 4-15 Static Diagram for ModelIntegration
4.5.3 Design for StudyPlan Module

Figure 4-16 shows the static class diagram for StudyPlan Module.

4.5.4 Design for SolutionViewer Module

Figure 4-17 shows the static class diagram for SolutionViewer module.
CHAPTER 5

AMUSE SIMULATOR AND UREX PROCESS

This chapter first introduces the design structure of AMUSE and the role of AMUSESimulator in the whole system; then, elaborates the software design about AMUSESimulator, Finally, the key operations inside AMUSESimulator are overviewed.

5.1 AMUSE Code

The AMUSE (Argonne Model for Universal Solvent Extraction) code is a software package developed by Argonne National Laboratory for the analysis of a Generic UREX process which is a sub process of the whole chemical separation process as shown in Figure 1-1.

The UREX process is a solvent extraction process capable of separating small quantities of transuranic elements (for example; Np, Am, Pu, and Cm) from aqueous nitrate and chloride solutions. These types of chemical streams are typically generated in reprocessing plant operations or in plutonium production and purification processes.

To incorporate the AMUSE Code into the systems engineering model, AMUSESimulator program is introduced as a communication layer that allows the user to quickly and easily define the UREX process. More straightforward ways to examine different process designs will be useful for the engineering analyst.
The relationships of AMUSESimulator with systems engineering model and AMUSE macro are shown in Figure 5-1, all calculations related to uranium solvent-extraction process are made by the interaction with the MS Excel macros, defined in the ANL AMUSE codes. Also, it is not required to run AMUSESimulator with System Engineering Program TRPSEMPro together, AMUSESimulator can work with AMUSE Macro as a stand along program.

Figure 5-1 Relationship among System Engineering Model, AMUSESimulator, and AMUSE Macro

5.2 Framework of AMUSESimulator

Figure 5-2 shows the framework of AMUSESimulator. First, each of the different steps in the UREX process is outlined on the screen and allows easy modification. After all the modifications have been made, AMUSESimulator will write all the modifications into Export File which is the input file for AMUSE Macro, and triggers the execution of AMUSE Macro; at the end of AMUSE execution, a report file is generated. To expedite the data usage, AMUSESimulator provides user with the capability that all the Export File and Report File information can be saved to database – either Microsoft SQL Server 2000 database MS ACCESS database or XML database.
5.3 Design of AMUSESimulator

The design process for the AMUSESimulator is discussed here. First, the main objects in UREX process are identified; then, according the functionality required for AMUSESimulator, the main domain classes and architecture of AMUSESimulator are obtained; finally, this section will elaborate the detailed design.

5.3.1 Objects Identification for UREX Process

Since AMUSESimulator is able to access AMUSE Macro, allow user to configure each steps in the UREX process, write this configuration to Export File, and load result from Report File, each of the processes has to be modeled to define how it is an integral part of the overall process. The process models in UREX process have been identified and the main data models in UREX process are shown in Figure 5-3.
5.3.2 Domain Class

Functions for AMUSESimulator include (Provided by ANL):

- Draw flowsheet (sections) - need each graphic shapes object to represent each object in UREX process.
- Define sections – require property dialog to define sections.
- Define feed streams – need property dialog to define streams.
- Run flowsheet – need export file parser to read Export File and export file writer to write Export File.
- View report – need a report file parser to read the report file, the correspondent object to store the information in Report file, and a way to display the information for that.

Summarizing from the above functionalities, the main domain classes include:

Flowsheet, Section, Stream, Concentration, Stage, GFlowsheet, GSection, and GStream.
5.3.3 Architecture of AMUSESimulator

AMUSESimulator includes four main packages as shown in Figure 5-4:
SimulatorManager, IOFileParser, AMUSEDataModel, and FSGraphics.

![Figure 5-4 Architecture of AMUSESimulator](image)

5.3.4 Detailed Design

5.3.4.1 Detailed Design for AMUSEDataModel

AMUSEDataModel is the main package for storing and organizing all the data used in UREX process. The static diagram for AMUSEDataModel is shown in Figure 5-5.
Figure 5-5 Static Diagram for Main Data Models for AMUSESimulator

5.3.4.2 Detailed Design for FSGraphics

Figure 5-6 shows the static diagram for FSGraphics package which includes four main parts:

- **FSCanvas** – display Flowsheet in way of graphic blocks.
- **FSStatus** – display detailed flowsheet information.
- **FSPROPERTY** – display the flowsheet properties list by way of table.
- **FSTreeView** - display the content of flowsheet by way of treeview.
5.3.4.3 Observer Design Pattern Applied to AMUSESImulator

Whenever the data source changed, all the observers must be updated. The observer design pattern, as discussed in [37] and [47], is applied here to satisfy this kind of requirements. The static diagram about the design is shown in Figure 5-7.
5.4 Database Design

Database design for AMUSESimulator is shown in Figure 5-8, and XML database structure can be found in Appendix A.

Figure 5-8 Diagram for the Structure of Database Design
5.5 Key Operations Overview for AMUSESimulator

5.5.1 Load Flowsheet from Database

AMUSESimulator allows user to load an initial Flowsheet from either existing Export File or Database. Figure 5-9 shows the case of opening a Flowsheet from Database.

![Figure 5-9 Dialog for Loading Flowsheet from Database](image)

Figure 5-9 Dialog for Loading Flowsheet from Database

5.5.2 The Main GUI for AMUSESimulator

The main GUI for AMUSESimulator includes five parts as shown in Figure 5-10.

- **Menu and toolbar (Top)** – where user can invoke all the available commands.
- **Flowsheet contents displayed in tree view format (Left)** – where user can select different section, stream.
- **Flowsheet contents displayed in way of drawing blocks (Middle)** – where user can select different sections, streams by clicking mouse at corresponding area.

5-9
5.5.3 Property Dialog for Flowsheet

Figure 5-11 shows the property dialog for flowsheet, where the user can see and modify the properties of the flowsheet.
5.5.4 Select One Section.

Figure 5-12 shows the case that a section is selected; both the property window (right part) and status window (lower part) show the information about this selected section.
Figure 5-12 the selected Section 1 (Extraction) is highlighted with a dark blue color.

5.5.5 Property Dialog for Section

Figure 5-13 shows the property dialog for section, where user can see and modify the properties of the selected section.
5.5.6 Select One Stream

Figure 5-14 shows the case that a stream is selected; both the property window (right part) and status window (lower part) show the information about this selected stream.
5.5.7 Property Dialog for Stream

Figure 5-15 shows the property dialog for stream, where user can see and modify the properties of the selected stream.

![Property Dialog for Stream](image)

Figure 5-15 Property Dialog for Stream
5.5.8 Utility Program: Database Viewer

Database Viewer is a utility program from which the user can look inside the database, and get all the stored run case, flowsheet, section, stream information. Figure 5-16 is a screenshot for Database Viewer.

Figure 5-16 Utility Module - Database Viewer
CHAPTER 6

CASE STUDY FOR THE DEVELOPED SYSTEMS

ENGINEERING MODEL

This chapter will give three test cases to show how the system works. Since the complex chemical separation process is still under construction in Argonne National Laboratory, here the test cases is given that identify the capability of the system.

6.1 Test Case 1 – Solving a Simple Optimization Problem

6.1.1 What is the Problem to Be Solved?

Minimize:

\[ f(X) = f(x_1, x_2) = 3 + (x_1 - 1.5x_2)^2 + (x_2 - 2)^2 \]  

Subject to:

\[ 0 \leq x_1 \leq 5; \quad 0 \leq x_2 \leq 5 \]  

By using TRPSEMPro to solve this problem, there are five steps required: SystemManager initialization, model integration, study plan definition, solution viewer definition, and study plan execution.

6.1.2 SystemManager Initialization

When user starts the program TRPSEMPro, the main GUI of SystemManager is the default user interface, as shown in Figure 6-1, where user can access any other three main modules (see Figure 2-4): ModelIntegration, StudyPlan, and SolutionViewer.
6.1.3 Model Integration

When the user clicks button “ModelIntegration” in the tool bar as shown in Figure 6-1, the interface for the module, ModelIntegration, pops up as shown in Figure 6-2. By using ModelIntegration, the user can add individual models into the system engineering model. These individual models can be the calculation model, the simcode model or the chemical separation models. Figure 6-2 shows the case that a calculation model is added;

Then, the user can specify the input, output parameters for each individual model and the parameter mapping information by which the data flow among these individual models can be specified. Figure 6-3 shows parameter definition dialog, here, 2 input parameters x1 and x2, and 1 output parameter f are defined; Data flow information with parent task, TaskProcess, is also defined here.

To the calculation model, the user must add corresponding formula(s). The user can add a formula either by a graphic way as shown in Figure 6-4, or in a tabular way as
shown in Figure 6-5. After finishing the model integration, the user can go back the SystemManager interface, the newly added calculation model can be seen here, as shown in Figure 6-6; During these operations, the user can save his/her work to a XML file at any time.

Figure 6-2 Insert a Calculation Model

Figure 6-3 Define the Input and Output Parameters
Figure 6-4 Input the Formula in the Graphical Way

Figure 6-5 Input the Formula in the Tabular Way
6.1.4 Problem Definition and Study Plan Definition

After user has integrated all the individual models into our systems engineering model and specified all the data flow among each of these models, the next step would be the problem’s definition and study plan definition. The problem definition includes equal and in-equal constraints definition, objective and design variables definition. As shown in Figure 6-7, user can specify x1 and x2 to be design variable, the upper bound for both x1 and x2 to be 5 and lower bound to be 0; objective of f needs to be minimized (downward arrow).

Users can access the StudyPlan module by clicking the button “StudyPlan” in the toolbar as shown in Figure 6-6. Figure 6-8 shows the main interface for the StudyPlan module. By using the interface, user can define two kinds of study plans: DOE study plan and optimization plan. Reader can reference Chapter 3 for detailed information about these techniques.
Initially, after user selecting “Design of Experiment” in the sampling options (Figure 6-8) and click the button “New”, the DOE definition dialog pops up, as shown in Figure 6-9, where the user can define the detailed information for one specific DOE study plan. Figure 6-9 shows the specification of all factors of interest while Figure 6-10 shows the specification of factors interaction. Figure 6-11 and Figure 6-12 show the content of the design matrix of which each row represents the configuration of design variables for each design run. After specifying the DOE study plan, optimization study plan can be setup as well. The optimization study plan definition dialog is shown in Figure 6-12. Figure 6-13 displays the pop-up window where the user can choose the optimization technique from the four available techniques (Reference to Figures 3-6, 3-7, and 3-8 for detailed algorithms). More detailed configuration can be found through the interface shown in Figure 6-14.

The defined DOE study plan and optimization can be found in the main interface of StudyPlan (Figure 6-14). User can choose a single study plan or the combination of multiple study plans for the study plan. An optimization study plan of “Optimization Random Walk” is selected in Figure 6-14.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Var</th>
<th>Obj</th>
<th>Type</th>
<th>LBound</th>
<th>Current V</th>
<th>UBoun d</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x1</td>
<td></td>
<td>real</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>Design Variable x1</td>
</tr>
<tr>
<td>2</td>
<td>x2</td>
<td></td>
<td>real</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>Design Variable x2</td>
</tr>
<tr>
<td>3</td>
<td>f</td>
<td></td>
<td>real</td>
<td></td>
<td>0</td>
<td></td>
<td>Objective Function</td>
</tr>
</tbody>
</table>

Figure 6-7 Problem Definition - Definition of Constraint, Objective and Design Variable

Figure 6-8 Specify Study Plan – Optimization with RandomWalk Technique
Figure 6-9 DOE Study Plan: Factor Specification

Figure 6-10 DOE Study Plan: Factor Interaction Specification
Figure 6-11 DOE Study: Design Matrix Display

Figure 6-12 Optimization Study Plan: Choose Optimization Techniques
Figure 6-13 Optimization Study Plan: Specification of Detailed Information

Figure 6-14 Add Defined Study Plan to the Calculation Model
6.1.5 Configure the SolutionViewer

The module SolutionViewer provides a convenient way for the user to configure how the design run result can be display. The user can access SolutionViewer by clicking button “SolutionViewer” in the toolbar shown in Figure 6-1.

The user can display the design run data in tabular form and/or chart form. Both the table and chart must be located in the graph page, as defined in TRPSEMPro. The user can add one or more graph pages, and within one graph page, one or more chart and/or table can be added.

Figure 6-15 shows the configuration dialog of graph page, user can configure how many charts and tables can be display in one specific graph page; Figure 6-16 show the chart configuration dialog. Four types of charts are provided: history chart, multi-line chart, scatter chart, and bar chart; After the chart type is selected, the parameter(s) of interest, which user want to be displayed, can be added to this chart, As shown in Figure 6-17, parameter x1, x2 and f are added to this chart. The detailed chart configuration, like position of legend, color of line, weight of line, also can be specified, as shown in Figure 6-18. Figure 6-19 shows the table configuration dialog. Most the operations are very similar to the case of chart.
Figure 6-15 Configure Graph Page

Figure 6-16 Configuration of Chart: Select Chart Type
Figure 6-17 Configuration of Chart: Select Parameter of Interest

Figure 6-18 Configuration of Table: Select Parameter of Interest
6.1.6 Execution of Study Plan

Up to this point, individual models have been integrated, study plans have been defined, and solution viewer also has been defined. The user are ready to execute the design run. To execute the design run, the user can first select the current study plan from all the defined study plans. These plans can be single-run, DOE study plan, optimization study plan, or the combination of these different plan with specified execution order. As shown in Figure 6-20, the user chooses the defined optimization study plan, optimRandomWalk, as the current study plan.

After choosing the current study plan, the user can click the button of “Execute” in toolbar as shown in Figure 6-1.

Figure 6-20 shows the design run results displayed in both the chart and tabular way. The user also can display the chart and table with another separate window to get an enlarged view, as shown in Figure 6-21.
Figure 6-20 Result Display: By Way of Graph Page

Figure 6-21 Result Display: In a Separate Window
6.1.7 Solution to this Problem

The selected start point is shown in Equation (6.3).

\[
\begin{align*}
  x_1 &= 5.0 \\
  x_2 &= 5.0 
\end{align*}
\]  

(6.3)

The solution for this problem is shown in Equation (6.3).

\[
\begin{align*}
  x_1 &= 2.99921648853126 \\
  x_2 &= 1.99943718878818 \\
  f &= 3.0000003204416 
\end{align*}
\]  

(6.4)

6.2 Test Case 2 - Minimization of the “Banana Function”

6.2.1 The Rosenbrock function

Equation (6.5) is called the Rosenbrock function which is often used as a test problem for optimization algorithms. It is also called the banana function because of the way the curvature bends around the origin as shown in Figure 6-22. It is notorious in optimization examples because of the slow convergence which most methods exhibit when trying to solve this problem.

\[
f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2
\]  

(6.5)
6.2.2 What is the Problem: Minimization of the “Banana Function”

The problem presented here is minimization of the “Banana Function” as described in Equation (6.6) and Equation (6.7).

Minimize:

\[ f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \]  

(6.6)

Subject to:

\[-2 \leq x_1 \leq 5; \quad -2 \leq x_2 \leq 5 \]  

(6.7)

6.2.3 Solving the Problem by Using TRPSEMMPro

This problem was modeled by way of Calculation model. The optimization techniques here adopted are both Conjugate Gradient method and RandomWalk method, and the results for those two methods are compared.

(1). Using Gradient Method: The selected start point is shown in Equation (6.8):
\[
\begin{align*}
& x_1 = 3.0 \\
& x_2 = 2.0 \\
& \text{(6.8)}
\end{align*}
\]

Solution to this problem is shown in Equation 6-9. A screen shot of the solution is shown in Figure 6-23 and Figure 6-24.

\[
\begin{align*}
& x_1 = 0.879375192668678 \\
& x_2 = 0.773655437989038 \\
& f = 0.0145629259562806 \\
& \text{(6.9)}
\end{align*}
\]

Figure 6-23 Conjugate Gradient Method: Solution Display in Chart Form

Figure 6-24 Conjugate Gradient Method: Solution Display in Tabular Form
(2). Using RandomWalk Method: The selected start point is shown in Equation (6.10):

\[
\begin{align*}
  x_1 &= 3.0 \\
  x_2 &= 2.0
\end{align*}
\]  

(6.10)

Solution to this problem is shown in Equation 6-11. Screen shot of solution is shown in Figure 6-25, Figure 6-26.

\[
\begin{align*}
  x_1 &= 0.981806441867034 \\
  x_2 &= 0.963611644121185 \\
  f &= 0.0003420442428644
\end{align*}
\]  

(6.11)

Figure 6-25 RandomWalk Method: Solution Display in Tabular Form
6.3 Test Case 3 – Application of ALM Method

6.3.1 What is the Problem to Be Solved?

Minimize:

\[ f(x_1, x_2) = x_1^4 - 2x_1^2x_2 + x_1^2 + x_1x_2^2 - 2x_1 + 4 \]  \hspace{1cm} (6.12)

Subject to:

\[
\begin{align*}
    h(x_1, x_2) & : x_1^2 + x_2^2 - 2 = 0 \\
g(x_1, x_2) & : 0.25x_1^3 + 0.75x_2^3 - 1 \leq 0 \\
    0 & \leq x_1 \leq 5; 0 \leq x_2 \leq 5
\end{align*}
\]  \hspace{1cm} (6.13)

6.3.2 Solving Problem by Using TRPSEMPro

Problem definition dialog is shown in Figure 6-27, \( x_1 \) and \( x_4 \) are design variables, \( f \) is the design function, \( h \) is an in-equality constraint and \( g \) is an in-equality constraint.
This problem was modeled by way of Calculation model. The optimization techniques here adopted are Augmented Lagrange Multiplier (ALM) method.

The selected start point is shown in Equation (6.14):

\[
\begin{align*}
  x_1 &= 3.0 \\
  x_2 &= 2.0
\end{align*}
\]  

(6.14)

The solution to this problem is shown in Equation 6.15. Screen shot of solution is shown in Figure 6-28.

\[
\begin{align*}
  x_1 &= 0.9994005161742 \\
  x_2 &= 1.00020313907996 \\
  f &= 2.99940259686877
\end{align*}
\]  

(6.15)
6.4 Integration of Chemical Separation with the System Model

As mentioned earlier, the complex chemical separation process is still under construction at Argonne National Laboratory; here one example is given to show how to integrate chemical separation process by using TRPSEMPro.

Figure 6-29 shows a flowsheet for the Glovebox phase of UREX process (see Figure 1-1 for the whole chemical separation process). The user can integrate all individual models into the flowsheet in the system model; Figure 6-30 shows the integration result. Currently the dataflow definition between individual models was complete, but most individual blocks are still under construction. Once all the models are implemented, the user can start the processing from defining constraints and objectives, determining study plan, selecting solution viewer to executing design runs for system analysis.
Figure 6-29 Glovebox Phase of UREX Process

Figure 6-30 Integration of Chemical Separation Process:

Glovebox Phase of UREX Process
CHAPTER 7

CONCLUSION

The whole chemical separation process, as shown in Figure 1-1, is complex to the point that definitely requires certain level of systematic coordination. To perform smoothly and meet the target extraction rates among those processes, this research proposes a general-purpose systems engineering model. Since constructing a system model is generally complex, requiring intensive communication and in-depth understanding, a carefully designed model can be more flexible and useful in the long term. This research project considers design concepts from requirements definition and conceptual design to system partitioning, and finally system validation. Lengthy pre-coding of the design process and recursive modification provides a system with high degree of flexibility and robustness.

A general purposed systems engineering model, Transmutation Research Program System Engineering Model Project (TRPSEMPro), was developed based on the above design concept. The system model includes four main parts: System Manager, Model Integration, Study Plan, and Solution Viewer. System Manager supervises all the case (problem) creation, and functionality definition. Model Integration identifies chemical extraction processes and their execution sequence. Study Plan is the key to define modeling scenarios, such as Optimization, Design of Experiments, single-set parameter and multiple parameter-set. No system can be completed without a visualization tool.
Solution Viewer provides a visual means to monitor the optimization process during and after model execution. TRPSEMPro can apply not only to chemical separation process, but also a general system model. TRPSEMPro allows industries to model their process quantitatively and to study the interactions between subsystems and performance of the model under the influence of various design parameters.

Software engineering and Object Oriented Analysis and Design (OOA&D) play a critical role during our software development. Through the application of OOA&D, the user can define objects and concepts from our problem domain that is quantitatively described by Unified Modeling Language (UML). The logical software objects were created from the previous definition. Meanwhile, different design patterns were also applied during the detailed design phase. Finally, those designed components were implemented by using Microsoft.Net, the most up-to-date object-oriented programming language framework from Microsoft.

Currently, only the UREX process module is available and ready to be implemented. Since extraction modules can be developed from various agencies with different development concepts and programming conventions, an intermediate bridge or interpreter is generally required. The system connects the only available process, UREX and with the TRPSEMPPro system model from the AMUSESimulator interface. The AMUSESimulator communicates with the calculation engine AMUSE macros designed for the UREX process. A user-friendly GUI in AMUSESimulator allows the user to efficiently define the UREX process – flowsheet, input streams, sections, and stages.

The combination of several up-to-date techniques makes this research unique and robust. Those include Microsoft.NET framework, MS SQL Server database, MS
ACCESS, eXtensible Markup Language (XML), Design of Experiment and System Optimization. The design and implementation of database provide an effective way to manipulate and store all the input and output data generated during the system design run. To analysis the system in an optimum way, several optimization technologies have been studied and correspondent algorithms have been developed and implemented into the system model. A universal XML file format is applied for data storage and transport among modules. While waiting more chemical processes to be developed, more improvement will be made to enhance the TRPSEMPro package for solving complex chemical separation process.
APPENDIX A

XML SCHEMA FOR UREX PROCESS

```xml
<?xml version="1.0" %tandalone="yes"?>
<xs:schema id="AmuseDataSet" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="AmuseDataSet" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="tblRunCase">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="case_ID">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:maxLength value="20" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="gen_time" type="xs:dateTime" />
              <xs:element name="description" minOccurs="0">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:maxLength value="500" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="tblFlowsheet">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="case_ID">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:maxLength value="20" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:element name="FS_ID">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="20" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="file_suffix">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="50" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="folder_name">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="50" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="num_sections" type="xs:int" />

<xs:element name="process_temp" type="xs:double" />

<xs:element name="diluent">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="20" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="solvent_extraction_unit">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="10" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="recycle">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="10" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="TBP" type="xs:double" minOccurs="0" />

<xs:element name="CMPO" type="xs:double" minOccurs="0" />

<xs:element name="CROWN" type="xs:double" minOccurs="0" />

<xs:element name="save_DValue">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="10" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="tblSection">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FS_ID">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="20" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="section_no" type="xs:int" />
      <xs:element name="section_name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="20" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="num_stages" type="xs:int" />
      <xs:element name="frac_efficiency" type="xs:double" />
      <xs:element name="QFAI" type="xs:double" minOccurs="0" />
      <xs:element name="aq_feed_cl" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="1" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="QFOI" type="xs:double" minOccurs="0" />
      <xs:element name="org_feed_cl" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="1" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="FEAI_input" type="xs:double" minOccurs="0" />
    </xs:sequence>
    <xs:element name="gen_time" type="xs:dateTime" />
  </xs:complexType>
</xs:element>
<xs:element name="aq_sample_name" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="2" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="org_sample_name" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="2" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="QSAI" type="xs:double" minOccurs="0" />
<xs:element name="QSOI" type="xs:double" minOccurs="0" />
<xs:element name="proc_temp" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="10" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="stage_efficiency" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="10" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="tblConcInitD">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FS_ID" type="xs:string" minOccurs="0" />
      <xs:element name="section_no" type="xs:int" minOccurs="0" />
      <xs:element name="comp_name" type="xs:string" minOccurs="0" />
      <xs:element name="aq_feed_conc" type="xs:double" minOccurs="0" />
      <xs:element name="org_feed_conc" type="xs:double" minOccurs="0" />
      <xs:element name="init_DValue" type="xs:double" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>

<xs:unique name="Constraint1" msdata:PrimaryKey="true">
  <xs:selector xpath="/tblRunCase" />
  <xs:field xpath="case_ID" />
</xs:unique>
APPENDIX B

XML SYSTEM ENGINEERING MODEL DESCRIPTION FILE

<?xml version="1.0"?>
<schema xmlns="http://www.unlv.edu/ncacm/TRPSEMPro/XMLSchema">

<!-- Task XML File - Author: Lijian Sun-->
<Task Name="TaskProcess" Type="TASK_PROCESS">
  <ParameterList>
    <Parameter Name="x1" Category="input" Type="real">
      <InitialValue>0</InitialValue>
      <VarIndicator>POTENTIAL_DESIGN_VAR</VarIndicator>
      <ObjIndicator>POTENTIAL_OBJ</ObjIndicator>
      <CurValue>0</CurValue>
      <LowerBoundValue>0</LowerBoundValue>
      <LowerBoundIndicator>No_LBound</LowerBoundIndicator>
      <UpperBoundValue>0</UpperBoundValue>
      <UpperBoundIndicator>No_LBound</UpperBoundIndicator>
      <Description></Description>
    </Parameter>
    <Parameter Name="f" Category="output" Type="real">
      <InitialValue>0</InitialValue>
      <VarIndicator>POTENTIAL_DESIGN_VAR</VarIndicator>
      <ObjIndicator>POTENTIAL_OBJ</ObjIndicator>
      <CurValue>0</CurValue>
      <LowerBoundValue>0</LowerBoundValue>
      <LowerBoundIndicator>No_LBound</LowerBoundIndicator>
      <UpperBoundValue>0</UpperBoundValue>
      <UpperBoundIndicator>No_LBound</UpperBoundIndicator>
      <Description></Description>
    </Parameter>
    <Parameter Name="x2" Category="input" Type="real">
      <InitialValue>0</InitialValue>
      <VarIndicator>POTENTIAL_DESIGN_VAR</VarIndicator>
      <ObjIndicator>POTENTIAL_OBJ</ObjIndicator>
      <CurValue>0</CurValue>
      <LowerBoundValue>0</LowerBoundValue>
      <LowerBoundIndicator>No_LBound</LowerBoundIndicator>
      <UpperBoundValue>0</UpperBoundValue>
      <UpperBoundIndicator>No_LBound</UpperBoundIndicator>
      <Description></Description>
    </Parameter>
  </ParameterList>
</Task>
</schema>
<UpperBoundIndicator>No_LBound</UpperBoundIndicator>
</Description>
</Parameter>
</ParameterList>
<TaskPlansTemplate />
<TaskPlans />
<Task Name="rexCaculator" Type="CALCULATION">
<ParameterList>
  <Parameter Name="x1" Category="input" Type="real">
    <InitialValue>0</InitialValue>
    <VarIndicator>DESIGN_VAR</VarIndicator>
    <ObjIndicator>POTENTIAL_OBJ</ObjIndicator>
    <CurValue>4</CurValue>
    <LowerBoundValue>0</LowerBoundValue>
    <LowerBoundIndicator>Inequality</LowerBoundIndicator>
    <UpperBoundValue>5</UpperBoundValue>
    <UpperBoundIndicator>Inequality</UpperBoundIndicator>
    <Description>Design Variable x1</Description>
    <MapInformation>
      <ParameterName>x1</ParameterName>
      <DataFlow>from_to</DataFlow>
    </MapInformation>
  </Parameter>
  <Parameter Name="f" Category="output" Type="real">
    <InitialValue>0</InitialValue>
    <VarIndicator>POTENTIAL_DESIGN_VAR</VarIndicator>
    <ObjIndicator>MINIMIZE</ObjIndicator>
    <CurValue>3.0000000025278</CurValue>
    <LowerBoundValue>0</LowerBoundValue>
    <LowerBoundIndicator>No_LBound</LowerBoundIndicator>
    <UpperBoundValue>0</UpperBoundValue>
    <UpperBoundIndicator>No_LBound</UpperBoundIndicator>
    <Description>Objective Function</Description>
    <MapInformation>
      <ParameterName>f</ParameterName>
      <DataFlow>from_to</DataFlow>
    </MapInformation>
  </Parameter>
  <Parameter Name="x2" Category="input" Type="real">
    <InitialValue>0</InitialValue>
    <VarIndicator>DESIGN_VAR</VarIndicator>
    <ObjIndicator>POTENTIAL_OBJ</ObjIndicator>
    <CurValue>4</CurValue>
    <LowerBoundValue>0</LowerBoundValue>
    <LowerBoundIndicator>Inequality</LowerBoundIndicator>
    <UpperBoundValue>5</UpperBoundValue>
    <UpperBoundIndicator>Inequality</UpperBoundIndicator>
    <Description>Design Variable x2</Description>
    <MapInformation>
      <ParameterName>x2</ParameterName>
    </MapInformation>
  </Parameter>
</ParameterList>
<ParameterDOE>
    <Name>f</Name>
    <Category>output</Category>
    <IsFactor>False</IsFactor>
    <LevelTypeIndicator>%</LevelTypeIndicator>
    <Values>-25 25</Values>
    <BaseLine>0</BaseLine>
</ParameterDOE>

<ParameterDOE>
    <Name>x2</Name>
    <Category>input</Category>
    <IsFactor>True</IsFactor>
    <LevelTypeIndicator>%</LevelTypeIndicator>
    <Values>-25 25</Values>
    <BaseLine>0</BaseLine>
</ParameterDOE>

<TaskPlansTemplate>
    <Plan>
        <TaskPlans>
            <Plan Name="optimRandomWalk" Category="Optimization">
                <OptimTechList>
                    <OptimTech>
                        <Type>RandomWalk</Type>
                        <NumIterations>50</NumIterations>
                        <Tolerance>0.0001</Tolerance>
                        <InitValueStepSize>0</InitValueStepSize>
                        <IncrementalValue>1</IncrementalValue>
                        <NumScanSteps>20</NumScanSteps>
                    </OptimTech>
                </OptimTechList>
            </Plan>
        </TaskPlans>
    </Plan>
</TaskPlansTemplate>

<FormulaList>
    <Formula>f = 3 + (x1-1.5*x2)^2 + (x2-2)^2</Formula>
</FormulaList>
REFERENCES


Gamma E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley, 1999.


Deitel, Deitel, Listfield, Nieto, Yaeger, Zlatkina, C# how to program, Prentice Hall, 2002.


Michael J. Laszlo, Object-Oriented Programming Featuring Graphical Applications in Java, Addison-Wesley, 2002.


R. Allen Wyke, Sultan Rehman, Bradley Leupen, XML Programming (Core Reference), Microsoft Press, c2002


The XML industry portal. XML.ORG, URL: http://www.xml.org/xml/resources_cover.shtml (viewed 2003)


W3C. Extensible markup language (XML) 1.0 (second edition). URL: [http://www.w3.org/TR/REC-xml](http://www.w3.org/TR/REC-xml) (viewed 2003)


