

WORKBENCH FOR MODELING AND OPTIMIZATION  
OF DIVERSE NETWORKS

**WORKBENCH FOR MODELING AND OPTIMIZATION  
OF DIVERSE NETWORKS**

**By JUNAID AZIZ, M.Sc.**

A Thesis Submitted to the School of Graduate Studies in Partial  
Fulfilment of the Requirements for the Degree Master of Science

McMaster University © Copyright by Junaid Aziz, April 2012

MASTER OF SCIENCE (2012)  
(Computing and Software)

McMaster University  
Hamilton, ON

TITLE: Workbench for modeling and optimization of diverse networks

AUTHOR: Junaid Aziz, M.Sc. (Mohammad Ali Jinnah University, Pakistan)

SUPERVISOR: Dr. Vladimir Mahalec

NUMBER OF PAGES: ix, 71

---

# Abstract

---

This work describes an architecture which enables experiments in optimization of networks that represent systems in diverse application domains, e.g. multi-product food production plants, gasoline blending and shipment, heat exchanger networks in refineries, etc. The prototype implementation is a web-based workbench (NOPT). Design of the workbench enables instantiation of different application domains via attributes describing entities (materials, energy) flowing through network arcs, and via node models relevant to the domain. From data describing the network attributes, NOPT generates a mathematical model described by a set of linear equations and provides a user with abilities to select appropriate solution algorithms. Multi-step composite algorithms, each solving a subnetwork or an entire network for specific time periods can be constructed with input from the user. Some of the steps in the algorithm can be non-linear procedures which compute specific model parameters. Hence, the architecture enables solution of bilinear systems of type “ $x*y$ ” (e.g. energy balances) by first solving for “ $x$ ” (e.g. mass flows) from some other set of equations (e.g. mass balances) and then solve for “ $y$ ” since “ $x$ ” is known. Current architecture of NOPT also supports the inclusion of external node models that helps user to import his customized node models into the workbench via the feature called User Node.

---

# Acknowledgements

---

First of all I would like to thank Almighty Allah for giving me courage and skills in order to be what I am today. I like to thank my supervisor Dr. Vladimir Mahalec for supporting me and guiding me throughout the completion of this thesis. I would like to thank my team members (names in alphabetical order) Erik Rafael and Shefali Kulkarni for being part of developing this software. Also, I like to thank Pedro Castillo for testing NOPT interface by solving different problems and providing his feedback. Lastly, I like to thank my parents, brother and sister for bearing me and encouraging me through tough times.

# Contents

---

## Contents

Introduction .....	1
1.1 NOPT: Network Optimization Tool .....	3
1.2 Node Libraries .....	4
1.3 User Node .....	5
1.4 Stream Library.....	6
1.5 Results Analysis .....	6
Architecture View .....	7
2.1 Architecture Constraints and Principles .....	10
2.2 Architecture Representation .....	12
2.3 Functional View .....	13
2.4 Process View .....	14
2.5 Logical View .....	15
2.6 Physical View.....	16
Graphical User Interface .....	18
3.1 Workflow .....	20
3.2 Design Rationales .....	21
NOPT Controller.....	24
4.1 Data Handling .....	25
4.2 Controller Execution Flow (Pseudo Code) .....	27
4.3 Case Study: Gasoline Blending Problem .....	29
4.3.1 Network Topology .....	30
4.3.2 Stream Properties .....	33
4.3.3 Calculation Phases .....	34
4.3.4 Set Parameter Data.....	35

4.3.5	Computational Sequence & Solver Parameters .....	37
4.3.6	Numerical Results .....	39
	Conclusion and Future Work .....	42
	Documentation .....	43
A.1	Doxygen .....	43
A.2	Star UML .....	45
A.3	NOPT Documentation Screenshots .....	45
	Use Case Details .....	48
	Node Model Design .....	68
	Bibliography .....	70

# List of Figures and Tables

---

Figure 1: Icons Representing Node Models .....	5
Figure 2: Top Level Architecture Diagram .....	8
Figure 3: Architecture Standards .....	11
Figure 4: Use Case Scenarios for Application Interface .....	13
Figure 5: Use Case Scenarios for Model Building.....	14
Figure 6: Interface Diagram .....	15
Figure 7: Activity Diagram .....	16
Figure 8: Deployment Diagram .....	17
Figure 9: GUI Work Flow .....	21
Figure 10: Controller Work Flow .....	25
Figure 11: Enter New Model Information.....	30
Figure 12: Set Node Title and Template Information .....	31
Figure 13: Set Stream Information and Port Selection.....	32
Figure 14: Complete Model.....	32
Figure 15: Properties related to Model.....	33
Figure 16: Set Stream Properties for Model .....	34
Figure 17: Set Calculation Phases.....	35
Figure 18: Parameter Data For Multiple Time Periods .....	36
Figure 19: Parameter Data For Single Time Period.....	37
Figure 20: Adding New Computational Sequence.....	38
Figure 21: Setting Solver Parameters.....	39



Figure 22: Run Validation Checks .....	40
Figure 23: Find A Solution.....	41
Figure 24: Showing documents directory structure for all the components .....	46
Figure 25: Showing links to UML diagrams .....	46
Figure 26: Partial Collaboration Diagram for Main Controller .....	47
Figure 27: Partial list of class members for Main Controller class .....	47
Figure 28: Designing Cost Node .....	69
Figure 29: Setting Properties for Cost Node.....	69

# List of Abbreviations and Symbols

---

NOPT	Network Optimization Tool
UML	Unified Modeling Language
.NET Framework	Microsoft Framework used to run applications
ASP.NET	Programming language used for web development
ASCII	American Standard Code for Information Interchange
ADO.NET	ActiveX Data Objects for .NET

# Chapter 1

---

## Introduction

---

Modeling, scheduling and production planning are major research topics in process industries. Researchers have solved many supply chain problems (e.g. gasoline blending, production of food bars, shipping of multiple liquid products vi a pipeline, etc.) using network modeling technique. In this technique all the components of a particular problem are termed as nodes and connections between components are termed as edges. For example in a gasoline blending problem Tanks and Splitters will be represented by nodes while streams coming in and going out from each of these nodes will be represented by edges. Due to high efficiency and popularity of this technique, tools for modeling network and solving problems are available today from software vendors, such a AspenTech, i2 Technologies, Oracle,SAP etc. Production planning and scheduling in various industries are typically solved either by special purpose tools developed for such industries or (e.g. refinery planning via AspenTech’s PIMS) or by generic tools (e.g. LINDO or AMPL). Special purpose tools incorporate domain knowledge and user interfaces that make it easy to solve problems in a given domain. Generic tool can model any kind of problems, but require that a user works at the level of individual equations. Among researchers, use of Microsoft Excel in combination with Microsoft Access and A

Mathematical Programming Language (AMPL) has become common [1]. Some researchers have also proposed tools based on simulation modeling for solving such problems [2] [3]. A model used in simulation can capture a lot of detail about a specific system, but the complexity of the model is dependent upon the purpose of the simulation that will be “run” using the model. However, it is known that accuracy of the results generated by using these simulation tools is highly dependent on accuracy of the simulation representing a particular physical system [4]. Also, so far such efforts have typically led to building network models to solve particular problems fulfilling specific scenarios only [5][6]. Most of these tools are either employed by a single user or by multiple users with no collaboration among the users. User interface of a network optimization tool provides capabilities for building models and manipulating values. Also, it controls how the user view results, influences how the user understands results and hence influences user choices.

The goal of this work is to design and develop a multi user web based network optimization tool that:

- Can be used to model networks belonging to different, diverse business domains.
- Is able to solve subsets of equations describing the network, e.g.
  - Solve only material balance equations, or
  - Solve volumetric flows and (quality\*flow) equations

- Enables a user to configure a network model for a given domain by using predefined models of nodes and network arcs (streams) , corresponding to the specific domain.
- Can solve various applications on the network (e.g. planning of material flows through the network) or its subnetworks (e.g. energy use optimization in a plant that is a subset of the total network).

Such a software will not only make it possible to model different types of networks rapidly but also be handy in constructing solution algorithms that utilize either mathematical programming or evolutionary optimization algorithms or their combinations.

The remainder for this chapter describes at a high level key functional capabilities of the NOPT system. This will provide the reader with an understanding of the functional requirements that are the basis for architectural design.

## **1.1 NOPT: Network Optimization Tool**

NOPT is a web based network optimization workbench where multiple users, regardless of their geographic locations can build network models, schedule production or optimize production plans. Users can build new models, persist them in their individual repositories for later editing, build custom reports and share their models with other users by changing the “view” property of the models to “public”. Public models grant read only access to the users other than the owner of these models. Access to the workbench is

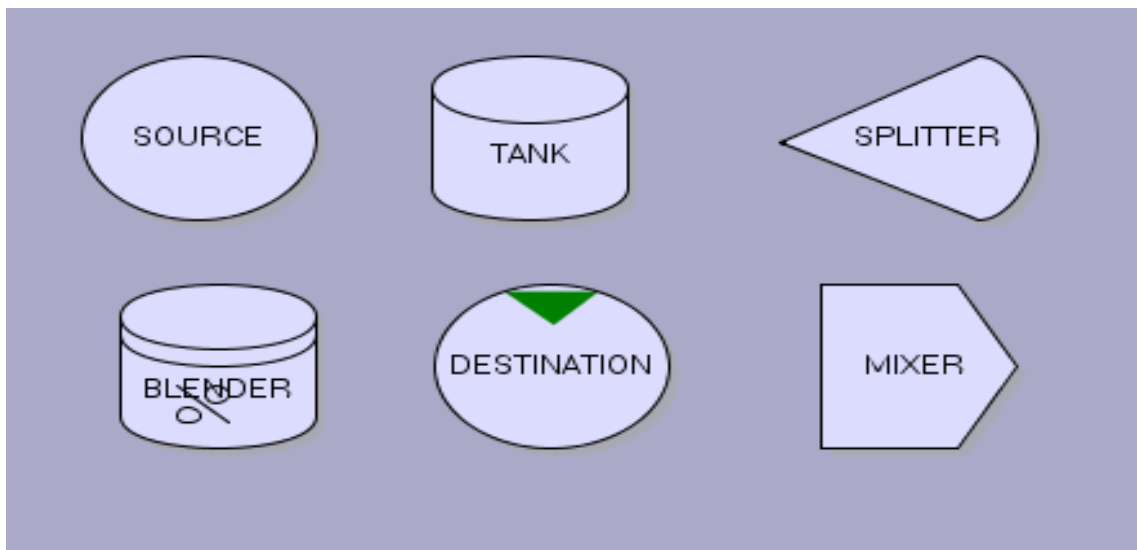
granted based on two type of roles “Administrator” and “User”. Administrators have full access to the workbench that includes updating node libraries for applications and setting appropriate icon representation for each of them, updating model templates, updating stream properties, etc. The users have limited access to the workbench that allows them to build new models and solve them, update or remove their models, view models built by other users and import them to their repositories, manage their model repositories, build charts and reports, etc. Each user selects an application domain, which causes NOPT to provide him access to the libraries of node models and material properties specific to that domain. Only Administrator of the workbench can attach such libraries and material properties to a particular domain.

## **1.2 Node Libraries**

Any supply chain or manufacturing system can be represented as a network consisting of nodes that process or store materials. Each node connects to the incident streams via ports. In real plants, ports can be placed where pipes connect to the process equipment or ports can be doors on the warehouse storing material that is being delivered via some transportation devices. In NOPT, node is described by equations that model the transformation taking place in the node. Our convention is that every entity appearing in the node equation is a variable. This means that vector of variables describing the network contains individual entries for all coefficients and all variables. Such approach

allows us to pose problems where coefficients of the equations are computed if the variables are set to some specific values.

NOPT workbench contains a library of node models representing different network types such as Gasoline Blending, Pipeline, and Heat Exchanger etc. Each node model is a generic representation of equations for a given model type. Some of these node models along with their iconic representation are shown below:-



**Figure 1: Icons Representing Node Models**

## 1.3 User Node

NOPT also allows user to add his equations generated outside the NOPT workbench environment. User can embed these equations with the help of User Node which is a special type of node that can be made part of the model. Currently, this type of node is representing two types of templates one that takes single input and multiple output and

other one that takes multiple input and single output. More templates will be attached to this node in future.

## **1.4 Stream Library**

Streams are used to transfer material from one point to another point in the Network model. These streams differ from one application area to another such as streams describing gasoline blending components are described by Volumetric Flow, RON, MON etc. Therefore, modeling of a network for specific application area requires that the streams in the network be described by a stream class containing the required attributes. Since NOPT is able to handle multiple subnetworks therefore more than one stream class can be used to describe the streams in the whole model.

## **1.5 Results Analysis**

Currently all result analysis are being done using Microsoft Excel. Once the solution to the model is found then results are stored into a database table by the Controller. Operator can then generate different charts based on that data. At present this is a manual process. However, in future NOPT workbench will have reporting and analysis component integrated to it which will let operators to generate reports and charts online using Microsoft SQL Server Reporting Services.



## Chapter 2

---

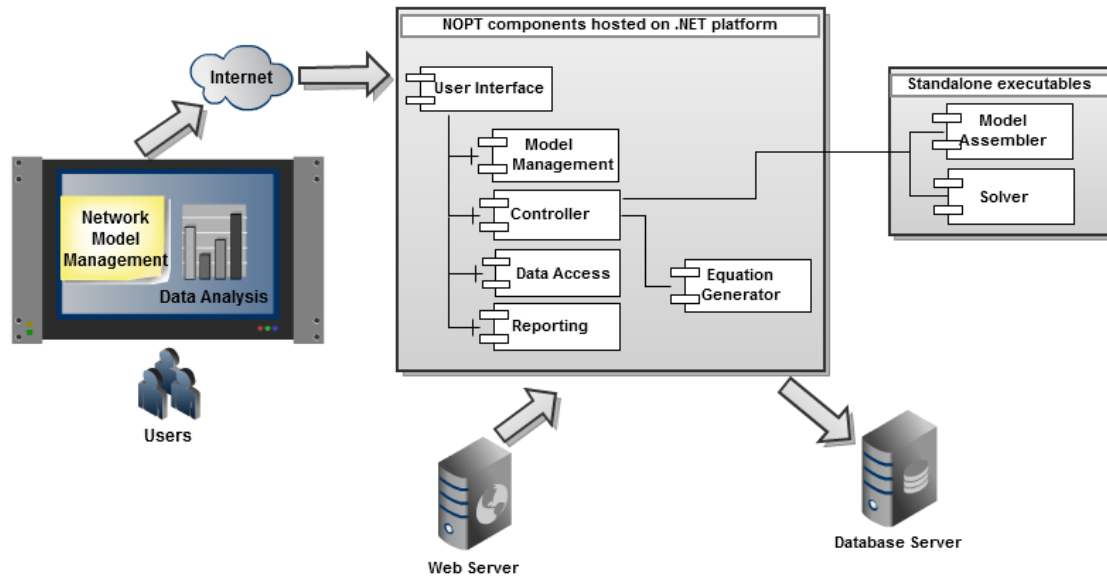
# Architecture View

---

The technical architecture of NOPT is described in the following terms:-

- Architecture Constraints and Principles: decisions about implementation, messaging between different components, persistency, data transfer and error handling
- Functional and Non-functional requirements: represented by use cases that describe the main functionality provided by workbench
- Domain concerns focusing on commonly agreed templates representing key information
- Design of required components and interfaces provided by these components
- Information display, data security, authentication of registered users and workbench administration
- Deployment standards for the whole workbench

Currently architecture of NOPT consists of four major components which are User Interface, Model Management, Controller and Data Access.



**Figure 2: Top Level Architecture Diagram**

### A. User Interface

NOPT interface has been implemented on .NET Framework. Model of a network is built via drag & drop of the generic node model icons on the canvas and connecting them by arcs (streams). User selects a class of attributes describing the material in the streams and also specifies node parameters (e.g. max. storage capacity). Specification of node parameters instantiates the generic node model and makes it specific to the given node. The user interface, written in ASP.NET, provides various forms and editing tools to specify “applications”. An application is a subnetwork for which computations will be carried out. In other words, multiple computational subnetworks can be defined for the same network topology. A subnetwork is described by its nodes, stream properties, starting and ending time period. Computational sequence for multiple subnetworks can

also be provided, including invocation of operator defined procedures at any computational step. Operators can also configure the look and feel of the interface according to their needs and likeness or dislikeness. Since user interface is compatible with all commonly used web browsers such as Firefox, Internet Explorer and Chrome etc. therefore this workbench is accessible from any location that has an Internet access.

## **B. Model Management**

This component enables operators to edit their existing models, change view properties of models by either setting them to public or private, persist models in their model repository and delete models from the model repository etc. Model ownership is managed via model view properties. An existing network model can be edited or copied into a new model and then edited. This facilitates network model sharing among operators.

## **C. Controller**

Once operator is finished building the model then the Controller starts generating the topology information, stream properties, node model information, solver parameters and proceeds towards finding the solution of the problem. Controller determines the execution flow starting from the model definition until it is solved. The first step is to generate all equations describing each node in the network. This is followed by assembling equations and variables for each subnetwork (based on the nodes included in each subnetwork and the attributes that describe the streams in that subnetwork) for as many time periods as specified by the operator. Supply and demand for each time period are computed from their respective data for the time horizon that is being modeled. Finally, the Controller

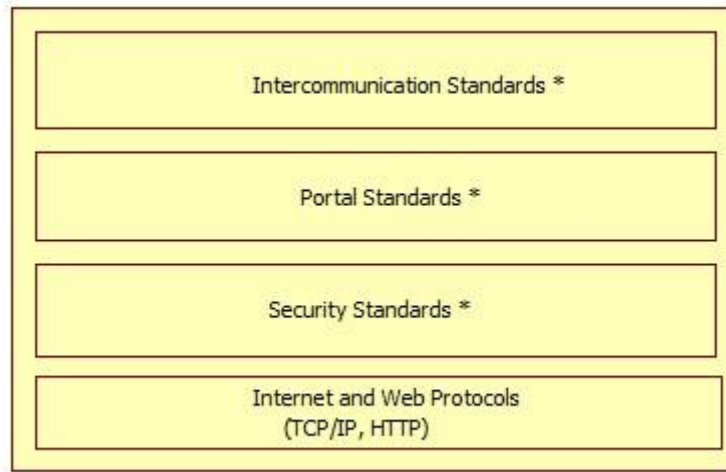
invokes the solver specified by the operator and persists the results in the SQL Server database.

#### D. Data Access

Topology information of all network models built by an operator, generic node models (model templates), stream properties that can be “processed” by the node models, demand and supply data, and results (intermediate and final) are persisted in the Microsoft SQL Server database. This component is responsible for performing add, remove and update related operations in the database.

## **2.1 Architecture Constraints and Principles**

Now a days focus has been shifted from standalone desktop applications to online multi-tier applications so that users can access and operate them from anywhere in the world. This also allows different users to collaborate, share their knowledge and experiences based on the usage of such online applications. Therefore, considering the current trend and standards, the architecture of NOPT has been designed using web based protocols. This enables the tool to handle requests made by the users online. All these requests are first handled by the web server and then forwarded to the application modules for processing. Some custom standards related to the NOPT tool have also been set as denoted by \* in Figure 3.



**Figure 3: Architecture Standards**

Security standards denote the user authentication, data privacy and data management. Access to the online workbench is granted to the registered users only through a unique user id and password. User accounts can be created or modified by the administrator only. Every user has his own view of the workspace which prohibits him to view other user's workspace. Some portal standards related to the work flow and presentations of interface have also been set. This web portal has been designed and made functional in such a way that user is aware of every step it takes to build a new model until finding the optimized solution for that model. Through the interface user is given different entry points during the process of solving a model. Using these entry points users can make appropriate changes at any point of time and those changes will be reflected in the solution instantly. Since the architecture of NOPT is based on different components therefore some message standards in form of ASCII files have been set to help components exchange information

with each other. Examples of such files are NET\_DEFINITION which holds the information of subnetworks belonging to a network model along with phases and equations types, SOLVING\_PARAM which holds solver information including solver name and parameters for each subnetwork, etc.

## 2.2 Architecture Representation

UML specification of the system has been divided into following four views:-

- **Functional View:** Describes the actors and use cases for the system, this view presents the need of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail.
- **Process View:** Describes interfaces provided by components of the system. Also, shows interaction of the components.
- **Logical View:** Shows complete activity detail of building model until finding an optimized solution.
- **Physical View:** Describes potential deployment structure for the system including Operating System, Database vendor, Platform information to host components and hardware information.

## 2.3 Functional View

Figure 4, 5 and 6 present the user perception of the functionality provided by NOPT. These use cases were synthesized while considering all the requirements but do not include descriptive text. Putting aside overriding architectural constraints outlined above, all development (in terms of design and content documentation) has been done in support of one or more of the following use cases. Details of all the use cases are provided in Appendix B.

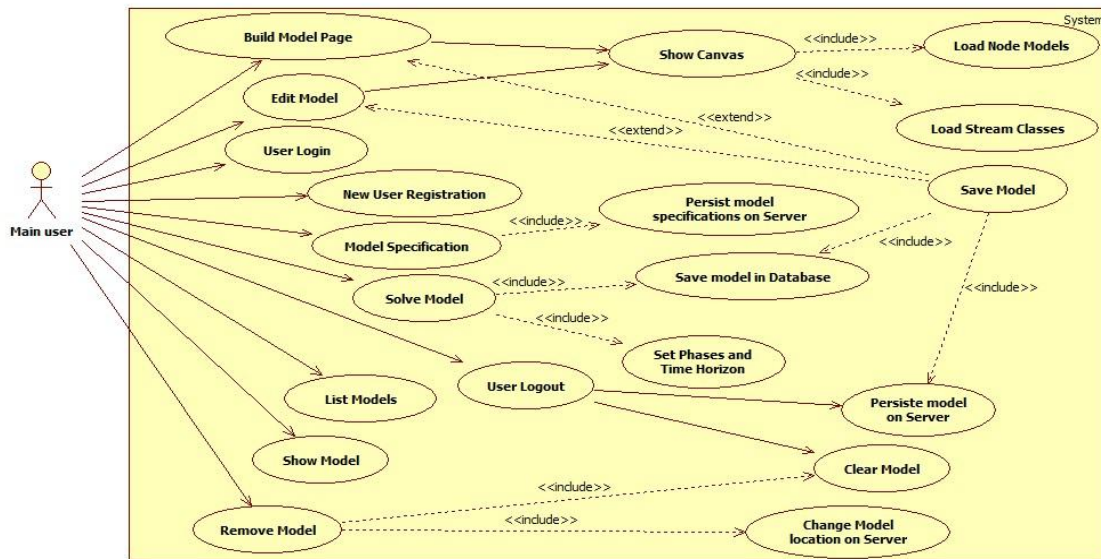


Figure 4: Use Case Scenarios for Application Interface

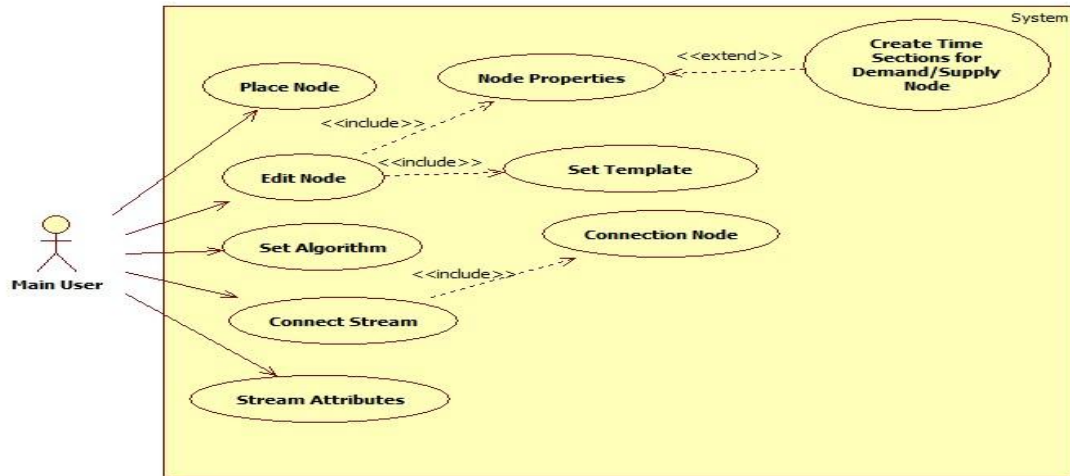


Figure 5: Use Case Scenarios for Model Building

## 2.4 Process View

The diagram below demonstrates how to present NOPT components in the UML [8] which expose set of interfaces. These components include NOPT Site, MainController, Assembler, NOPTGenerator and Solver. Assembler component makes use of another library component called NOPT++. By separating the notion of a component which is an actual software entity that exists at some identified end point from the interfaces which are a specification of some set of behavior we gain the flexibility to reuse common interfaces across number of components. In this way an interface can be identified as a stand-alone entity, versioned and managed independently from any component that implements it (in UML terms the component realizes the interfaces).



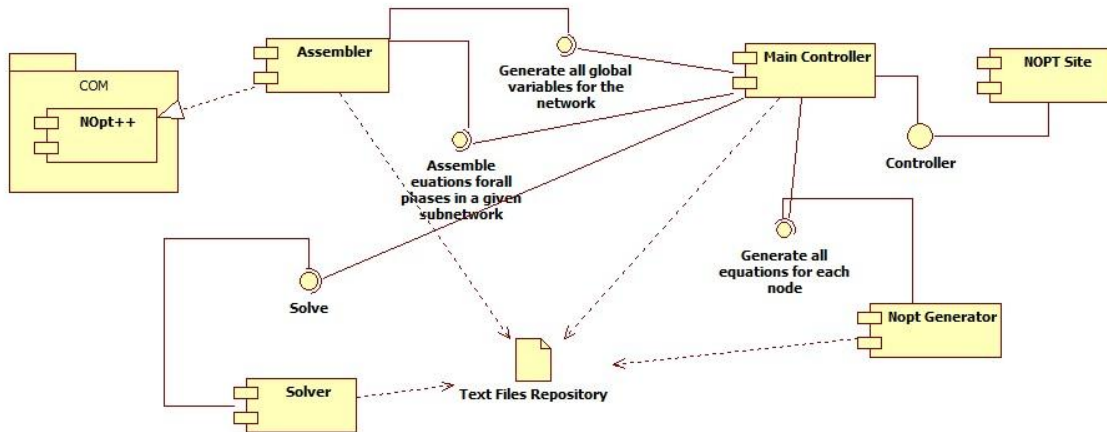


Figure 6: Interface Diagram

## 2.5 Logical View

Activity diagram below gives the detail view of NOPT work flow along with an appropriate description. Steps to find an optimized solution of a model are identified by step number. Controller is the main driver which supervises the whole process of finding the solution and make sure all the pre-requisites have been met before moving to the next step by performing different validation checks on the results obtained in the previous step. View of all the models are restricted based on user access level. Owner of the model has full access on that model however other users have just view level access to that model. Once the solution is found then the results are saved into the Database from where appropriate reports and graphs are generated for analysis purposes.

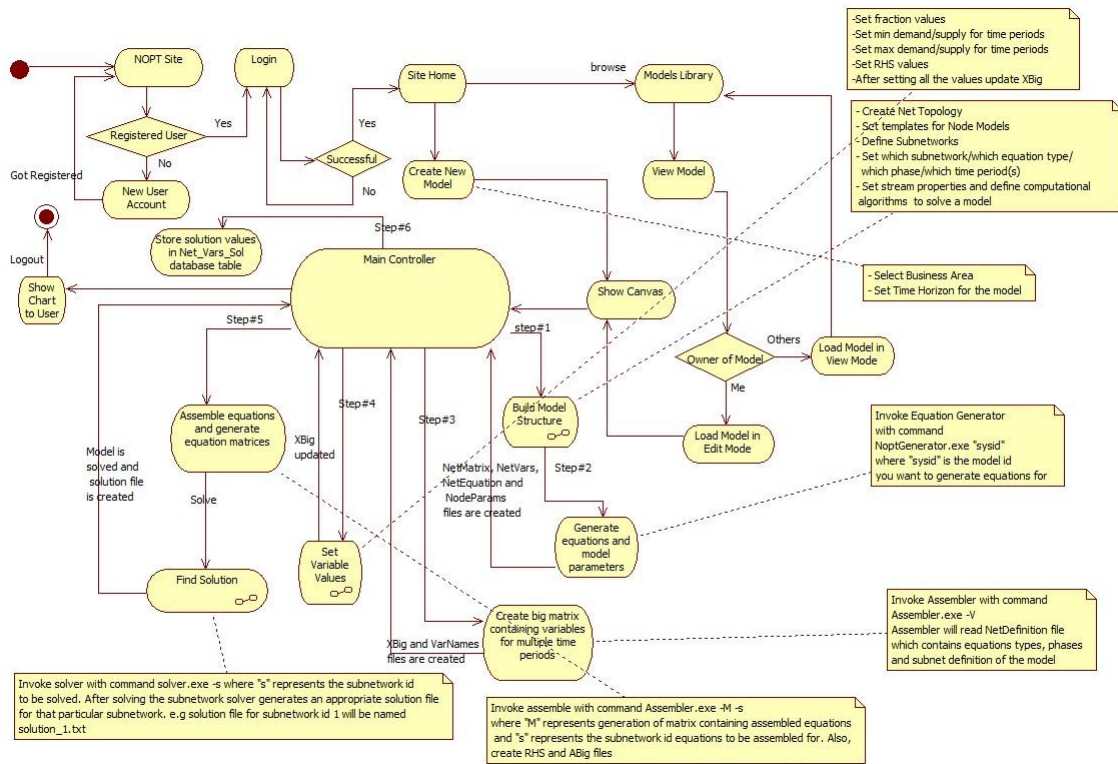


Figure 7: Activity Diagram

## 2.6 Physical View

NOPT tool is deployed in .NET environment running on Windows Server. Figure9 shows the detailed deployment structure. Net Diagrammer toolkit from MindFusion[10] is used to provide a Canvas where models can be built using different model libraries and streams connecting each other. Assembler and Solver Executive components have been implemented in C++ running as standalone executables. Database operations are performed using ADO.NET control. Only components running in .NET environment access database to perform different operations.

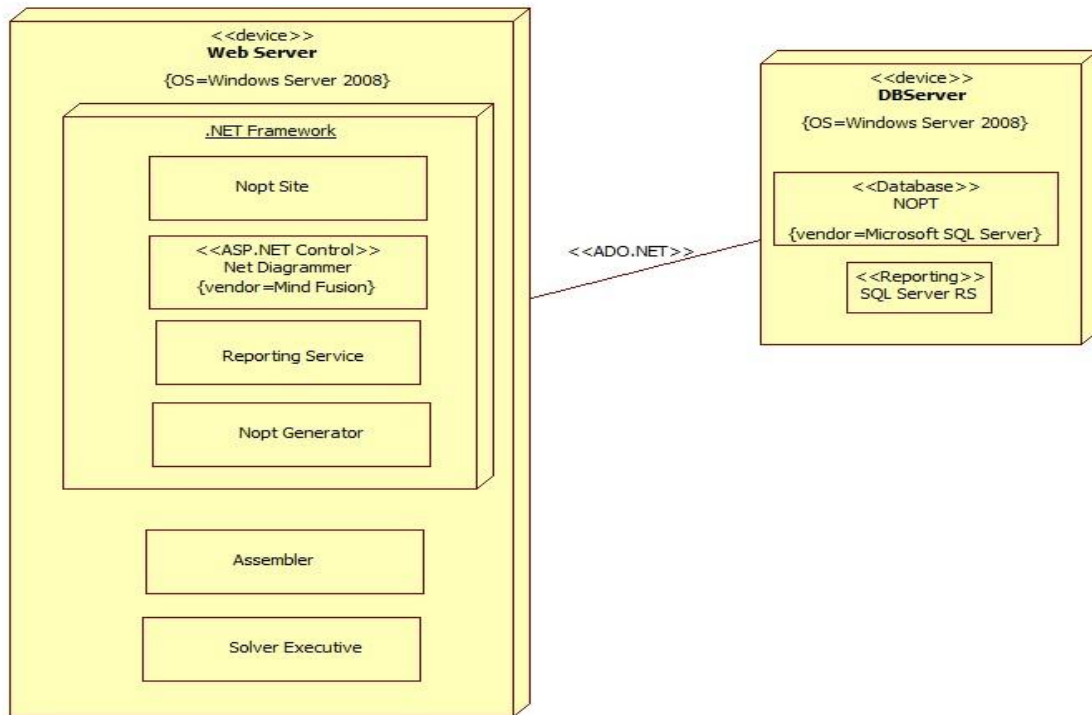


Figure 8: Deployment Diagram

## Chapter 3

---

# Graphical User Interface

---

NOPT workbench can be populated with knowledge (node models, properties of streams) corresponding to different domains. This enables an operator to select a specific domain (e.g. gasoline blending) and configure the network models with minimal effort. A model library representing models pertaining to the selected application area becomes visible in the GUI. For example, when operator selects gasoline blending (“GB”) as an application area, then all the model library containing tanks, stream splitters, blending tanks, etc. is activated in the GUI. Every library model is represented by a distinct shape and the associated generic model equations (model template). Alternative node models are available when appropriate, and an operator can select the model best suited for the application. Shapes in the libraries have been designed to convey the meaning of the underlying model, e.g. Heat Exchanger model is represented by a shape that is divided into four different color blocks where each block represents a specific port type, e.g. the red block on the left represents the hot inlet port, yellow block on the right represents the hot outlet port, green on the top represents the cold outlet and blue on the bottom represents the cold inlet port. Only Administrator of the workbench has access to add custom shape libraries. Once the network is configured, the operator specifies subnetworks that are used

for computation (by default, the entire network is one computational subnetwork). For each subnetwork, operator selects property set containing list of properties describing every stream in that subnetwork. Operator is then asked to set initial values for the equipment parameters for each of those templates used in that subnetwork. After that the entire network information (topology, parameter values and stream properties information, node models) are saved in the database. The network topology model along with its subnetworks also becomes visible in the operator's model repository. Each application area has associated default set of equations (by equation type) that describe the transformations that nodes carry out on the incident streams. Hence, the operator needs only to specify for each subnetwork the solver engine to be used and its parameters. Prior to invoking the solver, after the time periods are specified, the Controller populates the variables and constraints with the demand and supply constraints. The same process is repeated for every subnetwork in the computational sequence specified by the operator. This enables construction of complex algorithms, e.g. solve supply chain network planning, and that followed by solving problems for subnetworks. After obtaining the results if an operator wants to change parameter values, demand or supply values or solve the same model for different time periods then instead of repeating the whole process, operator is given an option to change the required values and re-invoke the solver. In this way operator can obtain the optimized results by running different tests on the subnetwork(s) and analyze them. Interface also allows user to build new model from an

existing model with different model information such as Model Name, Total Time Periods, etc.

## **3.1 Workflow**

Diagram below shows the workspace that user will see when he logs into the system which helps user guide through the process of building new model from scratch or building a new model from an existing model, updating model, setting model parameters and finding solution etc. As shown in Figure9, NOPT is basically comprised of five main sections described below:-

- a) Start: It allows user to select business area, set new model information and make changes to an existing model.
- b) Network Specification: It lets user to build new model using different node libraries, add subnetworks and setting stream properties. It also help user setting any simulation parameters for the model and building calculation phases for each subnetwork of the model.
- c) Data: It allows user to set parametric values such as demand data, supply data, Max flow, Min flow, etc.
- d) Solve: Here user will be defining computational sequence, selection of solvers and their parameters and invoke the solver to find the solution. As described above before finding a solution to the model Controller's job is to validate whether all

the pre-requisite have been met or not. That validation check will be run at this level.

- e) Analyze: Finally, once solution is found and saved in database, user is shown different charts and reports to perform analysis on those results.

Welcome back to your workspace...

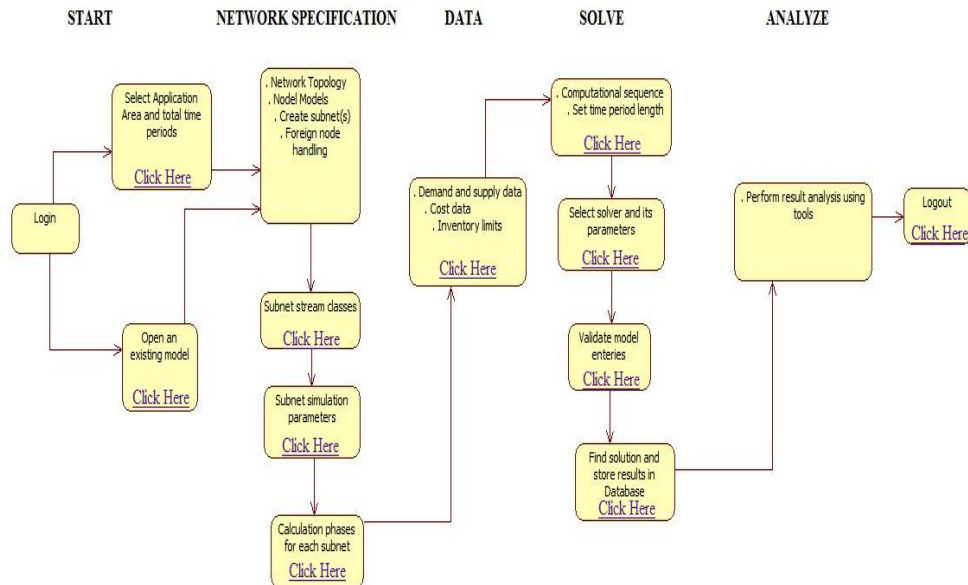


Figure 9: GUI Work Flow

## 3.2 Design Rationales

In order to keep NOPT interface simple and efficient some important design decisions have been made. With the help of these decisions it has been made possible that a novice user will not have to read manuals in order to start using this interface because workflow

diagram shown on workspace is self explanatory. These design decisions are mentioned below:-

- a) NOPT interface doesn't have single entry point rather it has multiple entry points which makes it easy for the user to jump to a particular section straight away without digging into multiple pages.
- b) Size of the interface has been kept compact by limiting the number of pages because web interfaces cluttered with lot of pages always tend to confuse users.
- c) Unnecessary input from the user is avoided. Most of the input fields or check boxes are set automatically with default values. So in order to complete a certain operation all he needs is a click of the button unless he wants to set different values.
- d) Informative messages are displayed wherever required. These messages are quite helpful to guide the user throughout the completion of a certain operation. Also, it helps user identify correct input data.
- e) Every page that requires input data from the user is equipped with Clear button to clear the entries at once. This is quite common practice especially in web development and has proven to be quite effective.
- f) Since a subnetwork can be solved either for single time period or multiple time periods and it is highly likely that model parameters will overlap for all the time



periods therefore an option is provided in the interface which allows user to set same value for all the time periods.

- g) All the messages generated by the Controller throughout the process of finding the solution are logged and displayed to the user, informing him about the outcome of every operation with the status either Success or Failure along with brief description.
- h) Solve button is used to invoke the solver for finding the solution of a problem. Initially, this button is disabled which enforces the user to click the Validate button first that eventually asks Controller to run different validation checks on the values which have been set for the model. If Controller returns with the status OK then Solve button becomes enable otherwise an informative message is displayed to the user.

## Chapter 4

---

### NOPT Controller

---

Controller is the core component of NOPT workbench software. It keeps track of all the operations necessary to perform in order to find a solution to the model. Figure10 shows the process which is followed by an operator to build a new model and find an optimized solution to it. Once the new model is built by the user then Controller generates the equations and required variables for that model. Next step is to set the values and bounds for those variables. Controller then tries to assemble the equations as per the criteria set by the operator. Finally operator is given an option to define any computational sequence using which his model is going to be solved. Before invoking the Solver to solve the model, Controller performs the check whether all the prerequisites have been met or not. At this point if it finds any property or important information for this model missing then it uses the default set of properties most appropriate for this model just to avoid any abnormality during the process of finding an optimized solution to this model. Lastly, based on the appropriate solver selection made by the operator, Controller tries to solve the model according to the computational sequence already set by the operator. After getting the results from the solver, Controller saves all the results in the Database for persistency and reporting purpose.

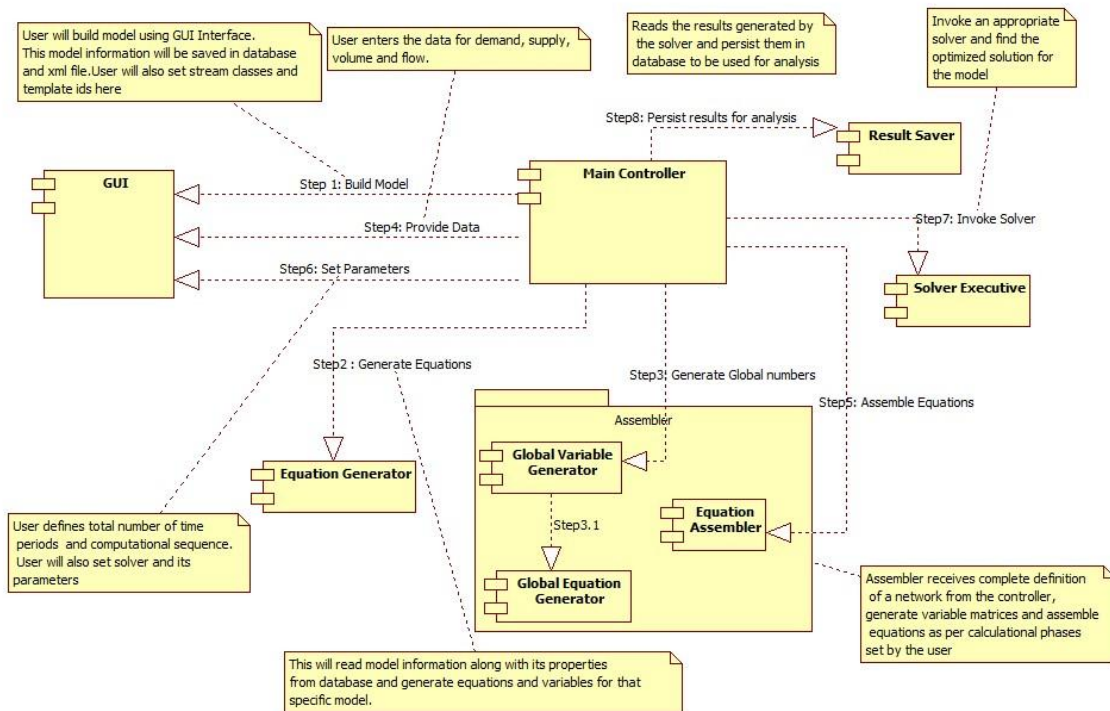


Figure 10: Controller Work Flow

## 4.1 Data Handling

Controller deals with two different types of data handling operations. One is to exchange information between components and other is data persistence. Since NOPT components have been developed in two different type of environment C++ and C# therefore information sharing between these components has been achieved using ASCII files. Currently there are 10 such files which exist in the system providing different set of information. Besides to these ASCII files, Controller also creates a separate XML file for every system id which can be used by the GUI to allow user to make changes in the model and set different parameter values. Detail of each file is mentioned below:-

- **Model\_Name.xml:** Describes the detailed information for the model including template id assigned to the node, stream class, node and stream names etc. Controller creates this file every time a new model is built by the user.
- **NET\_Definition.txt:** Holds all the information regarding the subnetworks belonging to a model. Controller creates this file.
- **Solving\_Params.txt:** Lists all parameters that must be sent to the solver engine to compute the solution of a model. Controller creates this file.
- **Solution\_subnetid.txt:** Lists all retained variables assigned to a particular subnetwork of the model with their respective values once the solver finishes the computation. Controller creates this file.
- **NET\_Matrix.txt:** Holds all information coming from the database to create coefficients of all the equations. This file is created by NOPTGenerator.
- **NET\_Vars.txt:** Holds all information coming from the database to create all variables to the model. This file is created by NOPTGenerator.
- **NET\_Equations.txt:** Holds all information coming from the database to create all equations to the model. This file is created by NOPTGenerator.
- **ABig.txt:** Specifies the coefficients of the equations, inequalities and objective function of the RHS data structure of the model. Assembler creates this file when it assembles all the equations by equation types.

- **RHS.txt:** Specifies the information related to the RHS of the equations of the model. Assembler creates this file when it assembles all the equations by equation types.
- **Xbig.txt:** Specifies the information related to the variables of the model. Assembler creates this file when it finishes generating all the model variables
- **Var\_Name.txt:** Specifies all variable names and parameters used in the mode. Assembler creates this file when it finishes generating all the model variables.

With the help of Data Access object from time to time Controller keeps performing different database related operations such as storing the data, updating the data, etc. Controller updates the database with all the information of a model from the time it is built until a solution is found by the solver. Once the solution is stored in database then it can be used for different analysis and reporting purposes. Database tables which are affected in such case will be discussed step by step when we will try to build a small gasoline problem and find an optimized solution using NOPT in a section below.

## 4.2 Controller Execution Flow (Pseudo Code)

Pseudo code below shows the execution flow of the Controller in terms of making calls to other components and data handling in result of these calls. It can also be seen in the following code that Controller also performs check on the existence of required file before moving forward. Before moving forward to the next step Controller performs a

check on the existence of these files. If it finds any of these files missing then it reports that as an error the logger and take appropriate action based on that.

ROUTINE NOPT\_CONTROLLER

```

1. (Path, Ret, Model.xml) := Call BuildModel (Path,Debug)
2. IF Ret = 1 AND (Model.xml) Exist = TRUE then
3.   (Ret, Net_Matrix.txt, Net_Vars.txt, Net_Equations.txt) := Call EGProgram
4.   (Path,Debug)
5.   IF Ret = 1 And (Net_Matrix.txt, Net_Vars.txt, Net_Equations.txt) Exist =
6.   TRUE then
7.     (Ret,Net_Definition.txt) := Call Create_Phase_Seq_TP(Path,Debug)
8.
9.   IF Ret=1 And Net_Definition.txt Exists = TRUE then
10.    (Ret, XBig.txt, Var_Name.txt) := Call GenerateVariables
11.    (Path,Debug,Net_Matrix.txt, Net_Vars.txt,
12.    Net_Equations.txt , Net_Definition.txt)
13.
14.    IF Ret=1 And (XBig.txt, Var_Name.txt) Exist = TRUE then
15.
16.    IF Ret = 1 And XBig.txt Exist = TRUE then
17.
18.    For( subnetid := I to totalsubnetworks )
19.
20.    While(Reassemble)
21.      (Ret, RHS.txt,ABig.txt) := Call
22.      AssembleEquations(Path,Debug,subnetid)
23.    END While
24.
25.    IF Ret = 1 And (RHS.txt, ABig.txt) Exist = TRUE then
26.      Ret := Call UpdateVars(Xbig.txt, Var_Name.txt,Debug)
27.      (Ret, Solving_Params.txt) := Call SolverSelection (Debug)
28.      (Ret, Solution_subnetid.txt) := Call SolverExecutive (
29.      Path,Debug, XBig.txt,RHS.txt,
30.      ABig.txt, Solving_Params.txt)

```

```
31.           END IF
32.         END For
33.
34.         For(subnetid := 1 to totalsubnetworks)
35.           IF Solution_subnetid.txt Exists = TRUE then
36.             Ret := Call StoreResults (Solution_subnetid.txt)
37.           END IF
38.         END For
39.
40.       END IF
41.     END IF
42.   END IF
43. END IF
44. END IF

END NOPT_CONTROLLER
```

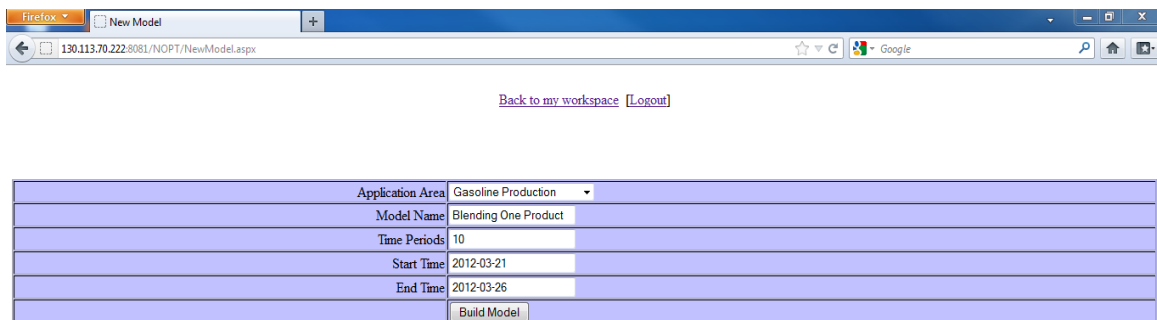
## 4.3 Case Study: Gasoline Blending Problem

Gasoline is an important and widely used refinery product. The biggest portion of a refinery's revenue comes from its production [11]. Also, its production quality provides a competitive edge to the refineries over each other [12]. NOPT workbench has been tested to build and solve problems related to Gasoline Blending, Pipelines and Heat Exchanger. In this section let us walk through the process of building a small Gasoline Blending model using NOPT interface and try to find an optimized solution. This will also help us understand the functionality of the Controller in a better way. This model consists of three supply nodes, four tanks, one blender and one demand node. Initially this model has one subnetwork attached to it. Properties attached to all the streams in the subnetwork consist

of FRACTION, MFLOW, QOPEN and VFLOW. This subnetwork is going to be solved using SPARSE and GLPK solver.

### 4.3.1 Network Topology

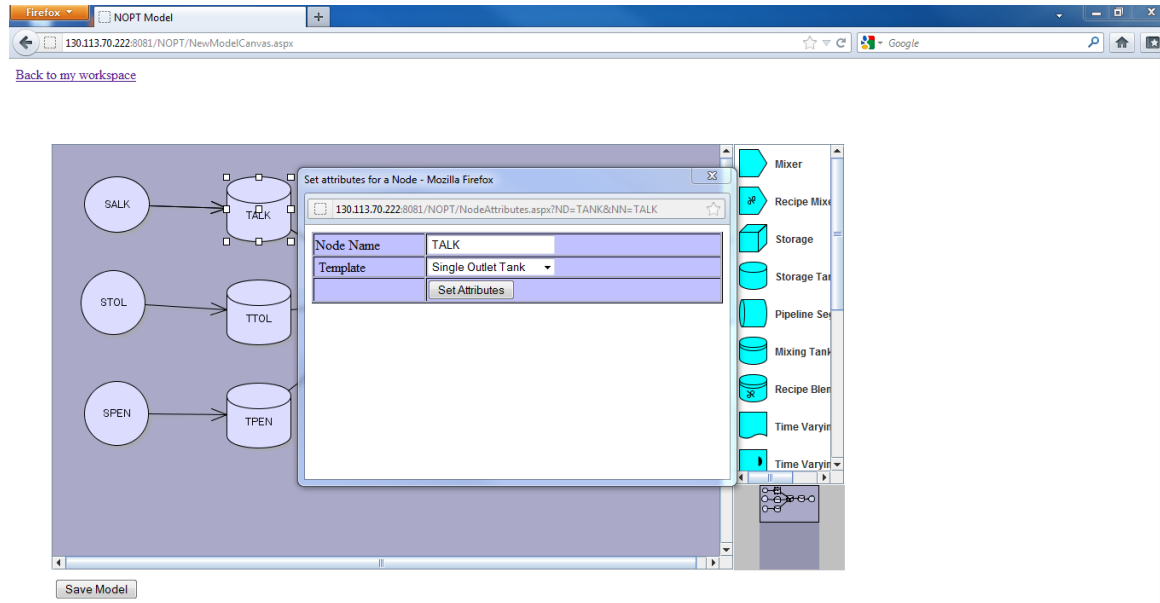
In order to build this model user will be asked to enter some model related information first such as application area, model name, maximum time periods and length of time. Since our model is a gasoline blending model therefore we select Gasoline Blending as an application area for this model and set other information as shown in Figure11. Once this information is entered then the next step is to build the model by dragging and dropping the appropriate node model icons from the list of icons visible on the right hand side of the Canvas. Initially dropped icon will have no title set to it so in order to set the title and template for this node user will have to double click on the node and enter the required information there. Figure12 shows how to set a title of a Tank to TALK and template to Single Outlet Tank. In this way we'll fill out the information for every node of this model one by one.



Application Area	Gasoline Production
Model Name	Blending One Product
Time Periods	10
Start Time	2012-03-21
End Time	2012-03-26
Build Model	

**Figure 11: Enter New Model Information**





**Figure 12: Set Node Title and Template Information**

After this we'll set the stream information for every link that connects one node to the other by just double clicking on that stream link. This stream information consists of name of the stream, outlet port of the node stream is coming from and inlet port of the node stream is connecting to. Figure13 shows how we are setting this information for the stream that connects source SALK to tank TALK? Figure 14 shows the final view of the model after setting its node and stream information correctly. Now at this point by just clicking Save Model button a new model will be built and an xml file with name Blending One Product\_1.xml storing all the model information will be created in the user root directory. The entire model related information such as model name, total time periods, start time, end time, etc. will be inserted in AMST\_PROC\_SYS table. A new subnetwork id 1 along with application id GB (Gasoline Blending) will be inserted in AMST\_APP\_SUBNET table. NET\_NODE\_MODEL table will be updated with all the

node models this new model consists of. Finally NET\_TOPOLOGY table will be updated will all the topology related information such as nodes and streams connecting to all the nodes.

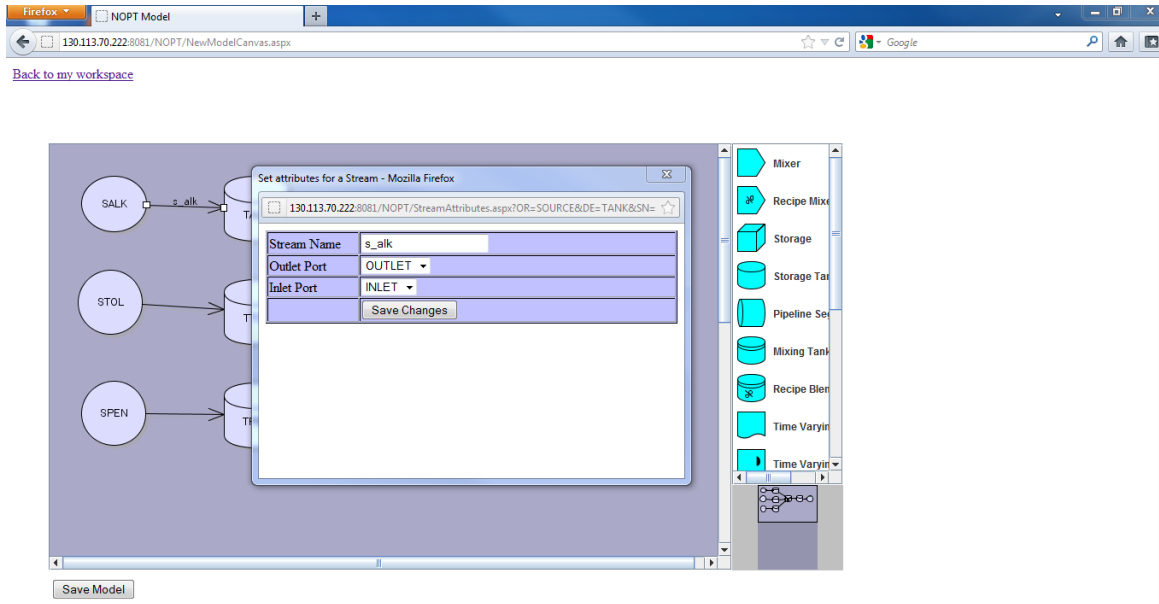


Figure 13: Set Stream Information and Port Selection

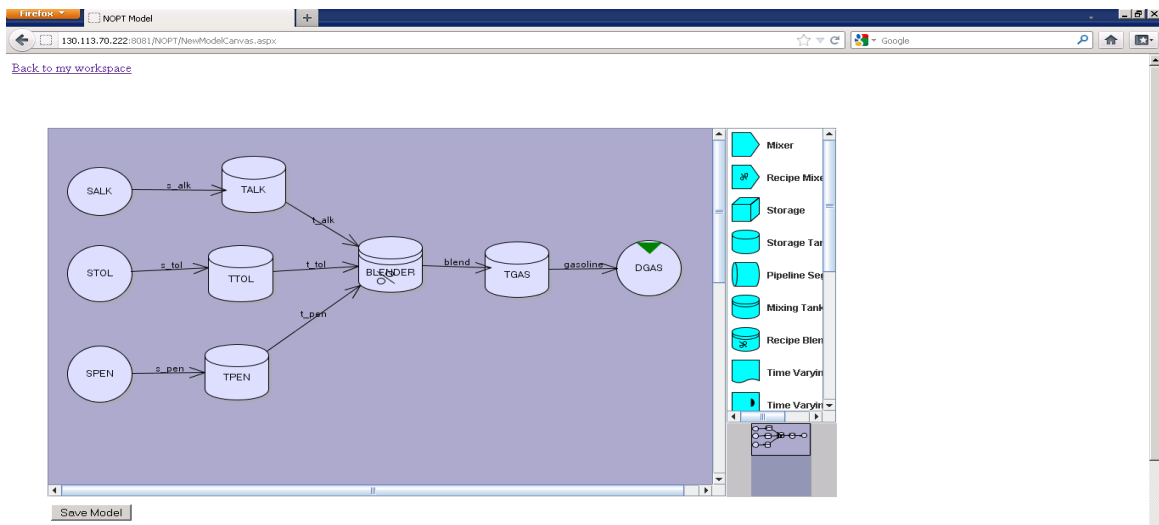
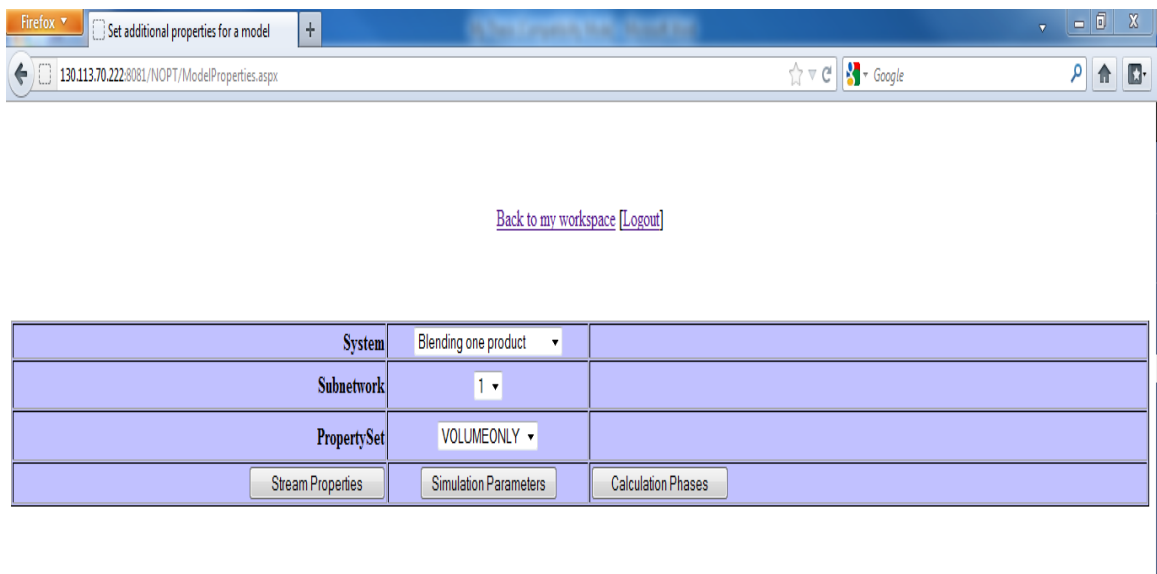


Figure 14: Complete Model

## 4.3.2 Stream Properties

Now our model is built and we need to attach stream classes to our model. So, from workspace page we will click on select stream classes link which will take us to the page where we can select stream properties for our model as shown in Figure 15. By clicking on the Stream Properties button a pop window will appear showing the list of stream properties which user can choose from. Figure 16 shows the list of properties we have selected for our gasoline blending problem. User can always come back to uncheck the selected stream properties. Clear button here will help user uncheck the checked properties all at once and start over again. Once the user clicks the SetProperties button then AMST\_STRM\_PROP table will be updated with all the stream properties that are selected to be attached to our new model.



**Figure 15: Properties related to Model**

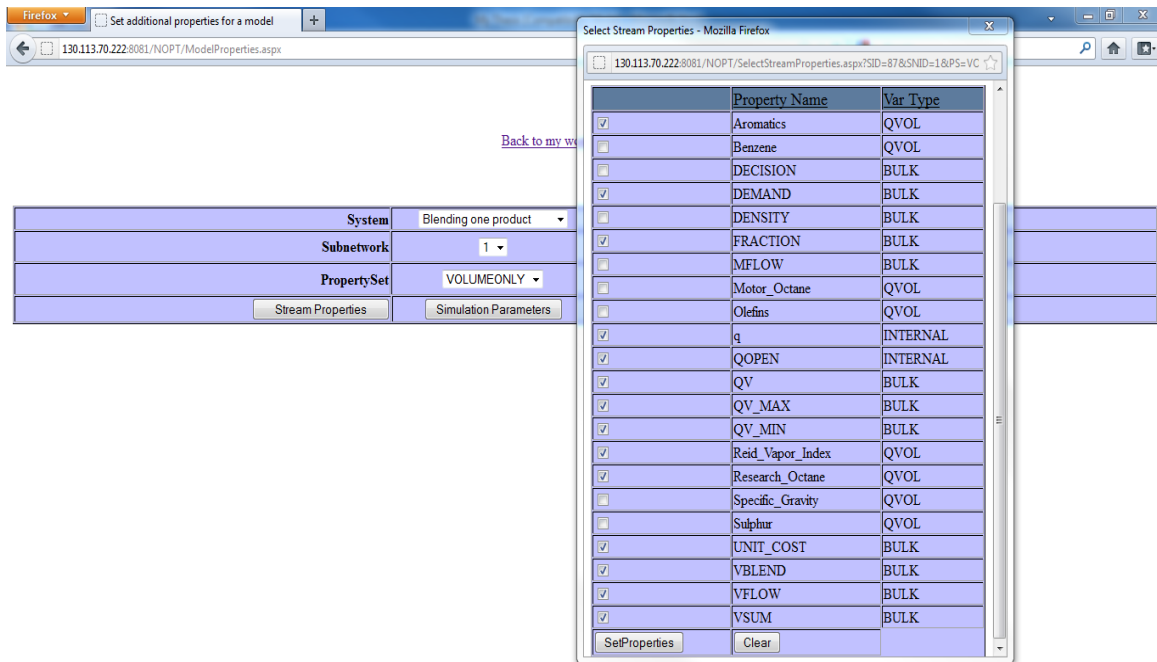


Figure 16: Set Stream Properties for Model

### 4.3.3 Calculation Phases

Solution of the model can be tweaked by changing the order of equations to be solved by the solver. Change in such order is managed by different phases. These phases define the order of equation types to be computed while find the solution to the problem. Every equation type has a particular phase number attached to it. User makes such selection on the Calculation Phase page where one equation type can also be attached to more than one phases separated by comma. Figure 17 shows how are we assigning phase numbers to the equation types for our problem? Controller updates AMST\_CALC\_PHASE table with equation type and phase no for this new model once SetCalculationalPhases button is clicked.

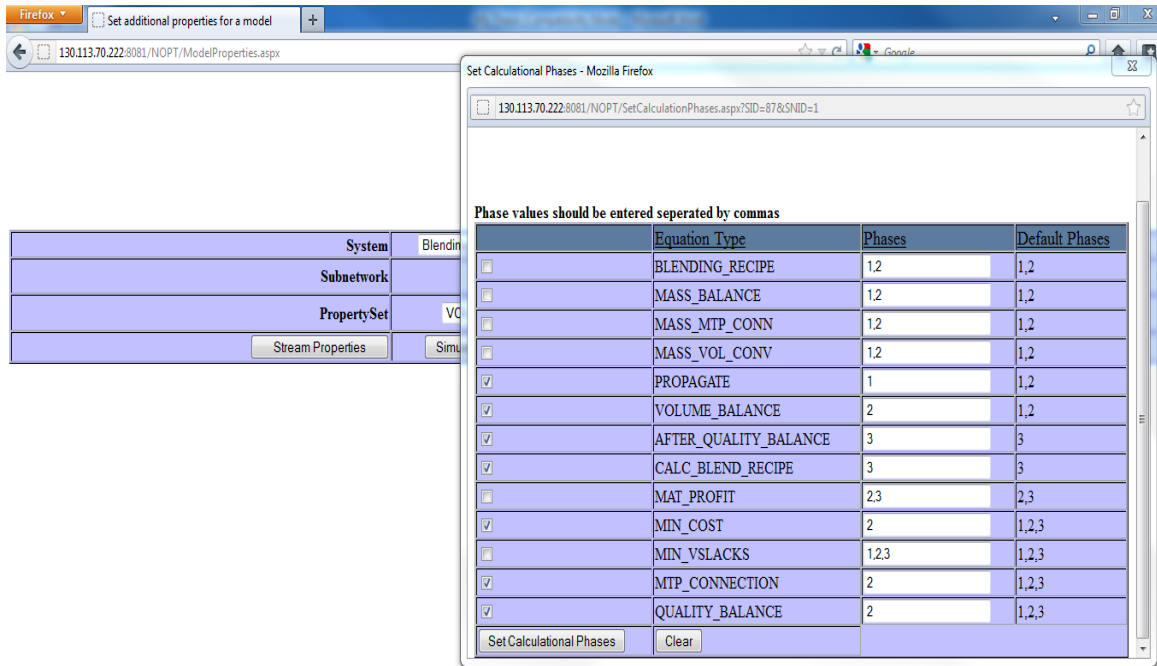


Figure 17: Set Calculation Phases

### 4.3.4 Set Parameter Data

After user defines the calculation phases Main Controller creates NET\_Definition.txt file and in turn invokes NOPT Generator and Assembler to generate required equations and variables for the this model. NET\_Matrix.txt, Net\_Equations.txt and Net\_Vars.txt files are created by NOPT Generator while Assembler creates XBig.txt and Var\_Name.txt files. NOPT Generator updates AMST\_EQUIP\_PARAM table with all the equipment parameters corresponding to all the node models used in this model. It also populates NET\_EQUATIONS, NET\_MATRIX and NET\_VARS tables in database with the information that matches to NET\_Equations.txt, Net\_Matrix.txt and Net\_Vars.txt files. Initially variables in XBig.txt file are set to some default values which user is required to

change according to his problem. Since NOPT can be used to solve single time period as well as multi time period problems therefore it allows user to set values for both type of problem. User is provided two ways of setting the values for all the time periods. Either he can enter the values once and let the NOPT set these values for all the remaining time periods at once or he can enter values for each time period by himself. Figure 18 shows how are we setting values for all the time periods at once however, Figure 19 shows in case if we want to set different values for different periods. Once SetParameters is clicked then Controller updates XBig.txt file with the new values corresponding to different variables for all the time periods.

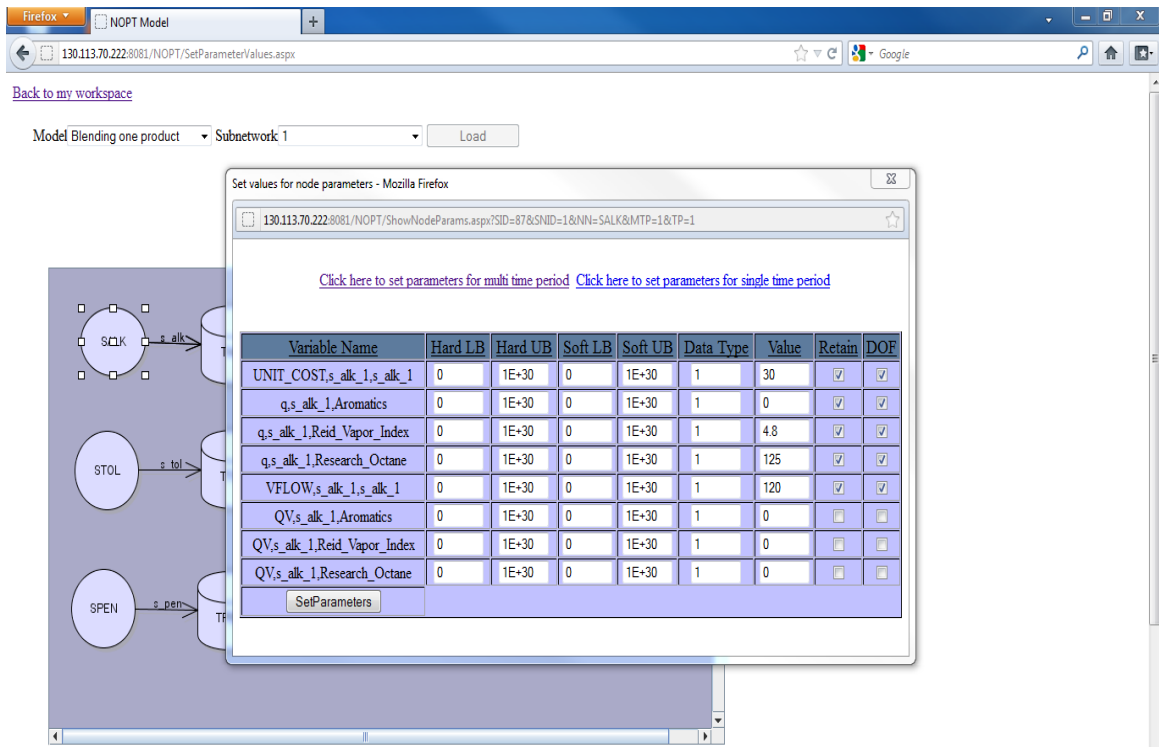


Figure 18: Parameter Data For Multiple Time Periods

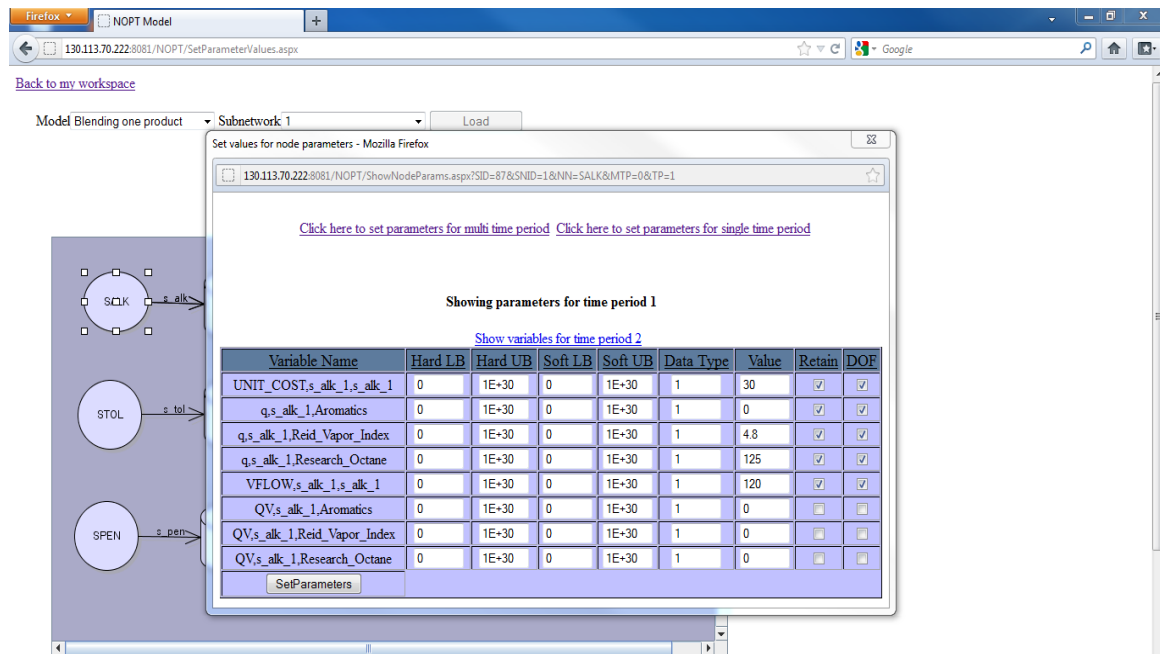


Figure 19: Parameter Data For Single Time Period

### 4.3.5 Computational Sequence & Solver Parameters

NOPT can be used to create multiple subnetworks from an existing model. These subnetworks having different set of properties and parameters can be solved by defining computational sequences. Different type of solvers and parameters can be set to different subnetworks in such computational sequences. Creating a new computational sequence is very easy. User just has to click Add Sequence button which will open a pop window asking different information about this new sequence. User enters time period length this subnetwork to be solved for, select solver at subnetwork level and select solver for each phase along with parameters. This whole process helps user creating one computational

sequence involving one subnetwork. Changes in an existing computational sequence can be made by clicking the Update Sequence button. Figure20 shows how are we creating a computational sequence for our problem that has just one subnetwork? However, Figure21 shows the process of selecting solvers and their parameters for our test problem. With the click of SetCalculationalSequece button Controller insert a new computational sequence in AMST\_CALC\_SEQUENCE table along with the subnetwork id, start time period and end time period value. Controller inserts the solver information against this sequence number in AMST\_SOLVING\_PARAM table. Controller also creates Solving\_Params.txt file which has solver information for this model.

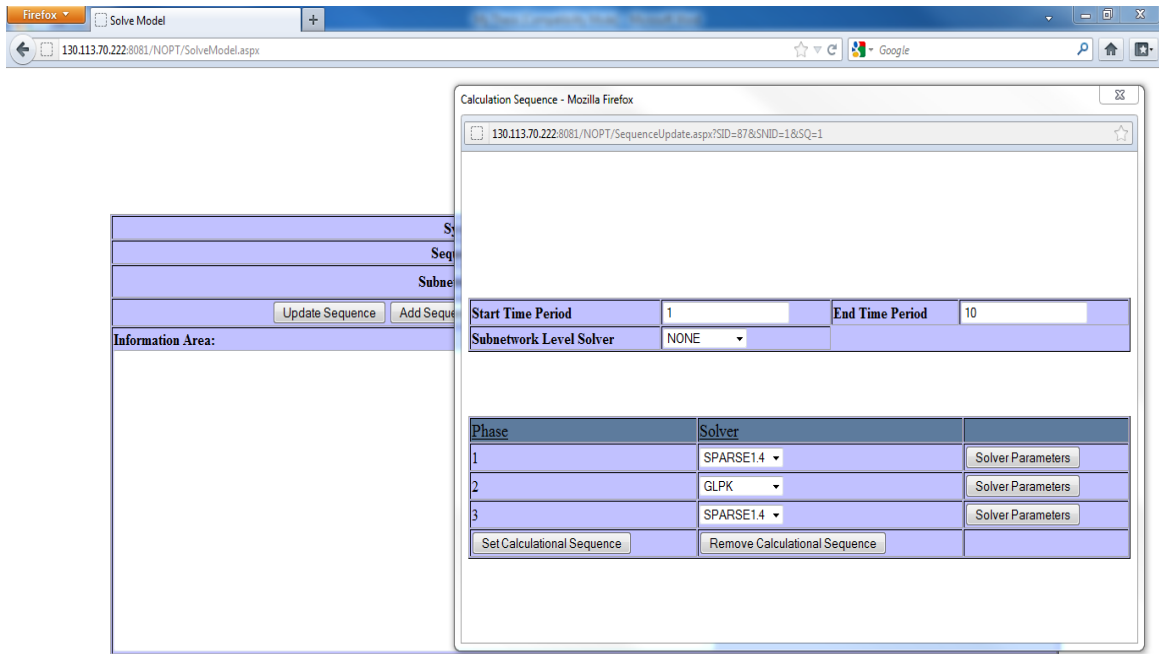


Figure 20: Adding New Computational Sequence



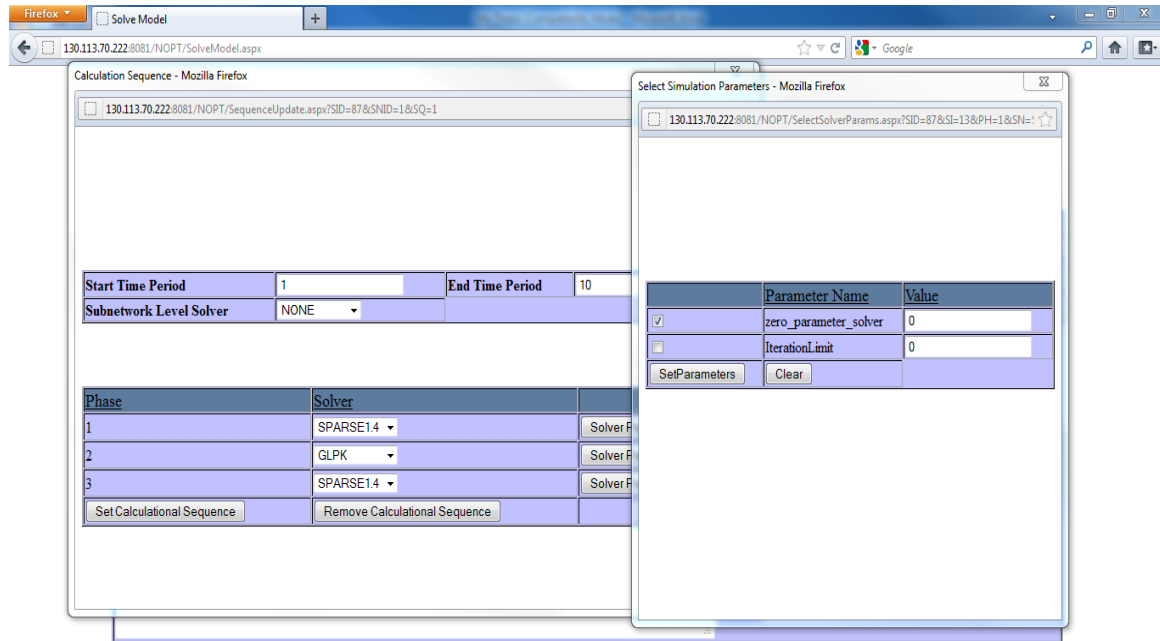


Figure 21: Setting Solver Parameters

### 4.3.6 Numerical Results

After selecting the solver and setting required parameters for each solver now it's time to find the solution to our problem. Before invoking the solver, Controller's job is to make sure that all the required values have been set. This is done with the help of Validate button on the Interface. In order to find the solution user is required to click the Validate button first which runs different validation scripts in the background and display the results in Informative Area. If the message displayed says All Clear then it means all the validation scripts have returned with the status OK otherwise an appropriate error message will be displayed to the user along with the steps need to fix the problems. This is quite handy because interface will only invoke the solver until required information has

been entered resulting non-existence of unnecessary information validation scripts in solver component. Figure22 shows that after clicking the Validate button Controller ran different validation scripts on our model and returns the message saying “All Clear” which means that all the basic information is there for this model which the solver needs in order to find a solution. Solve button has also become enabled. Now, solution to our model can be found by clicking the Solve button which will eventually ask Controller to invoke the solver. Once the solver is invoked different messages will be returned and displayed in the Informative Area section of the page just to inform the user about the updated status of current operation. User will be seeing “Solver is finished executing message” when solver is finished its execution. User will also be getting message in the end detailing whether solver was able to find an optimum solution to the problem or not.

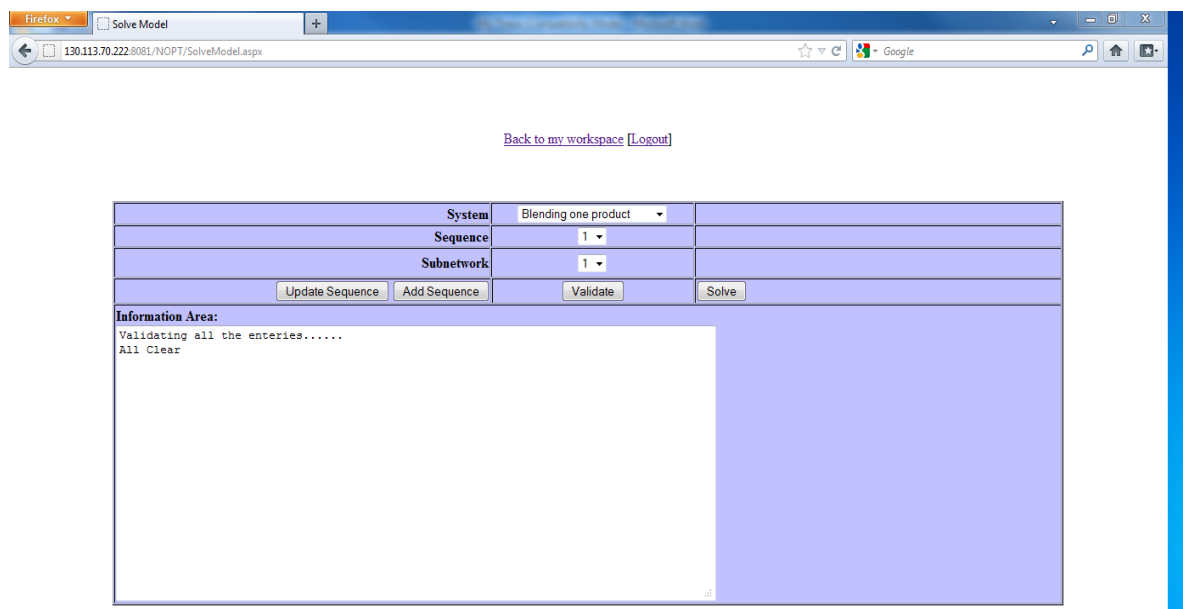


Figure 22: Run Validation Checks

Figure23 shows that an optimal solution to our problem has been found by the solver. Since the solution is found therefore Controller will store all the results in the file called Solution\_1.txt where 1 represents the subnetwork id and inserts the results in database table NET\_VARS\_SOL.

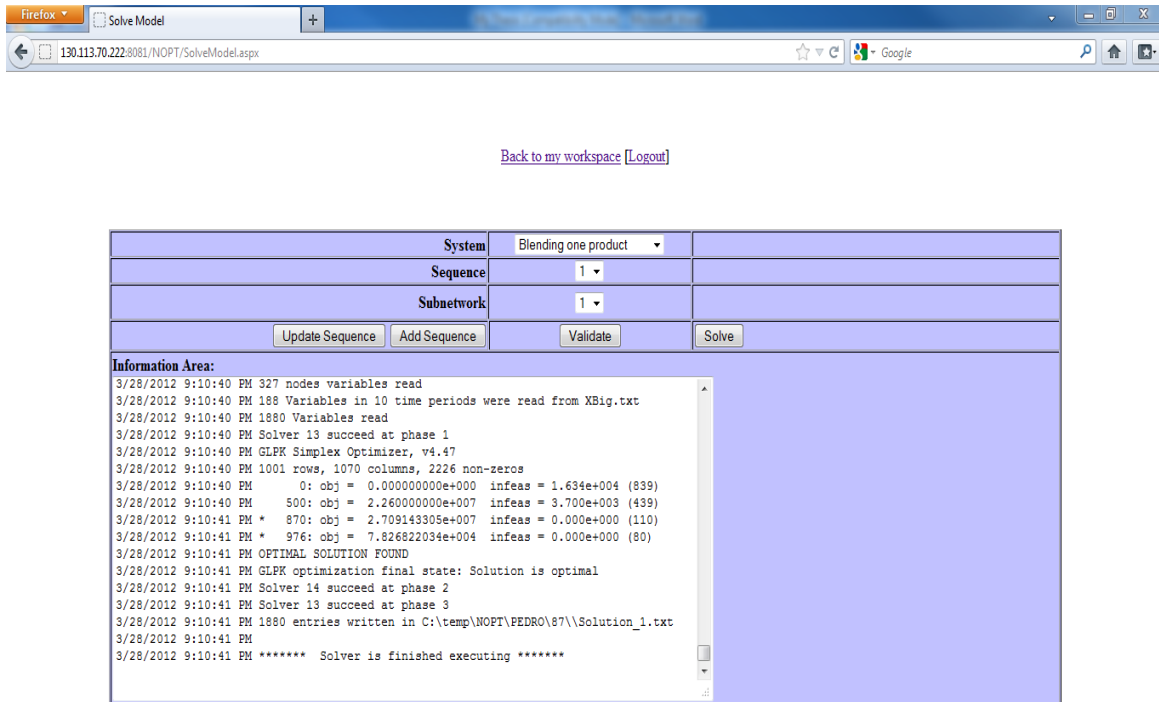


Figure 23: Find A Solution

## Chapter 5

---

### **Conclusion and Future Work**

---

Currently we are testing NOPT to solve large scale gasoline blending and pipeline problems which require building large models consisting of hundreds of nodes. These problems grow rapidly in size as the number of time periods increases, potentially leading to truly large scale problems. NOPT design enables us to model such large systems and solve them very rapidly. Our future work will include capabilities to generate formulations that include two-level decomposition algorithms, as well as integrated planning & scheduling applications. Introduction of this workbench is definitely going to help people to solve diverse problems, systemize their models and share them with other operators. Currently we are also working on allowing operators to share their reports with other operators same way they share their models now.

# Appendix A

---

## Documentation

---

From the beginning the idea was to automate the process of generating NOPT documentation as much as possible. For this purpose open source tools were used and integrated with our own modules where required. This gave us the benefit of maintaining updated documentation all the time. Also, if a change was required either at Design level or implementation level then it didn't take too much time to reflect that change in the appropriate documentation. Since this was made part of the standards which were set for the development of NOPT therefore a lot of research and meetings were conducted in order to find appropriate tools to achieve the above mentioned goal. Finally, some tools were found meeting our needs and requirements which are described below.

### A.1 Doxygen

Doxygen [7] is an open source documentation generator for multiple programming languages such as C#, JAVA, C++, PHP etc. which parses tags defined in the source code and automatically generates html files for documentation. It also has the ability to generate LATEX reference manual. This tool has been developed under Linux and Mac OS X environment but equally compatible for Unix and Windows. Installation and configuration of Doxygen is pretty straightforward mostly by just setting certain

properties in its configuration file that include choosing a programming language, directory location where your source code files reside etc. Once all set then documentation generation is just one click away. Apart from generating class member details, Doxygen also helps generating collaboration diagrams, inheritance diagrams and call graphs of different methods that helps user understanding the flow quite quickly as compare to manually digging into the code which is a time consuming task. Doxygen has vast majority of tags meant for fulfilling different requirements. Tags which have been used mostly for NOPT documentation generation purpose are described below:-

**@file:** It is used to indicate that commented block contains the description for a specific file. The name of that specific file is passed as a parameter.

**@author:** Every source file is making use of this tag in order to show the author information.

**@version:** Since the development of NOPT was an iterative process therefore changes in any source file was tracked through versions set for a particular build.

**@date:** This tag was used to show the date when this source file was created or modified.

**@section:** Every source file made use of this tag in order to provide brief description about the functionality.

**<a href>**: Since html hyperlinks to the external sources can also be set in doxygen therefore this tag was used to provide links to UML diagram images generated through other tools.

## **A.2 Star UML**

Star Uml [9] is an open source tool which helps designing UML diagrams quickly. It also allows user to do forward engineering and reverse engineering on the source code currently for both JAVA and C#. Besides to this it also helps an Architect to organise design level documents and corresponding UML diagrams in hierarchical format. All the diagrams shown above in Architecture View section have been created using this tool and exported in image format so that these diagrams could easily be read by Doxygen generated doc files. Integration of these image files with Doxygen is a manual procedure as Doxygen is set to retrieve these files from a particular directory. So, whenever a change in any of these documents is required then the updated version of the exported image file has to be placed in that directory so that Doxygen can show the updated diagrams on user's next visit to such files.

## **A.3 NOPT Documentation Screenshots**

Screen shots below show different documentation pages for NOPT which have been generated using the above mentioned tools.

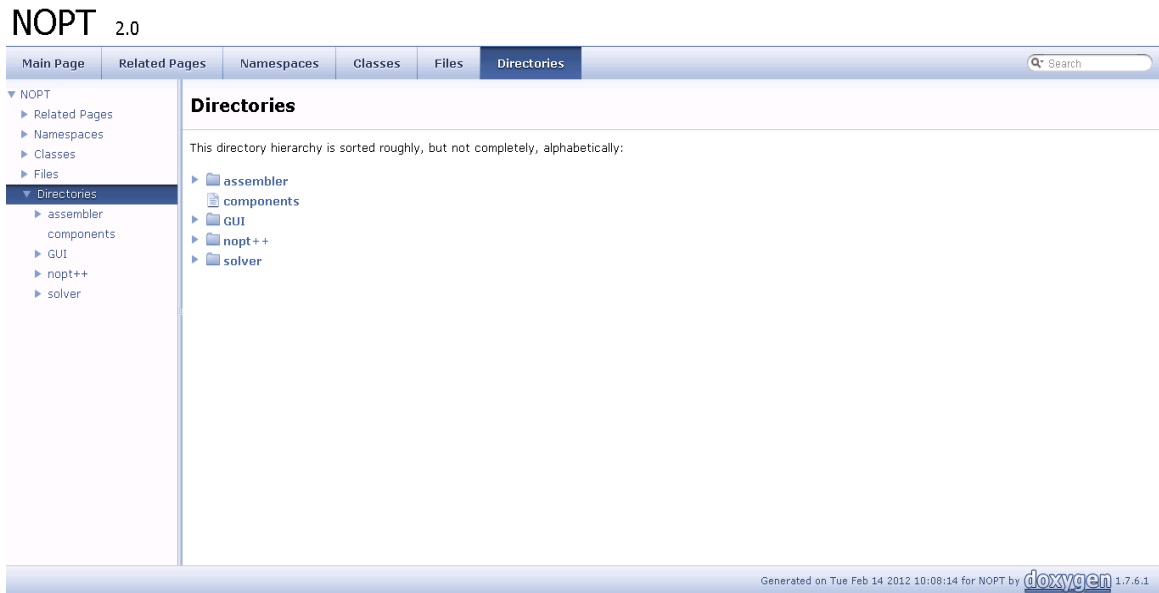


Figure 24: Showing documents directory structure for all the components

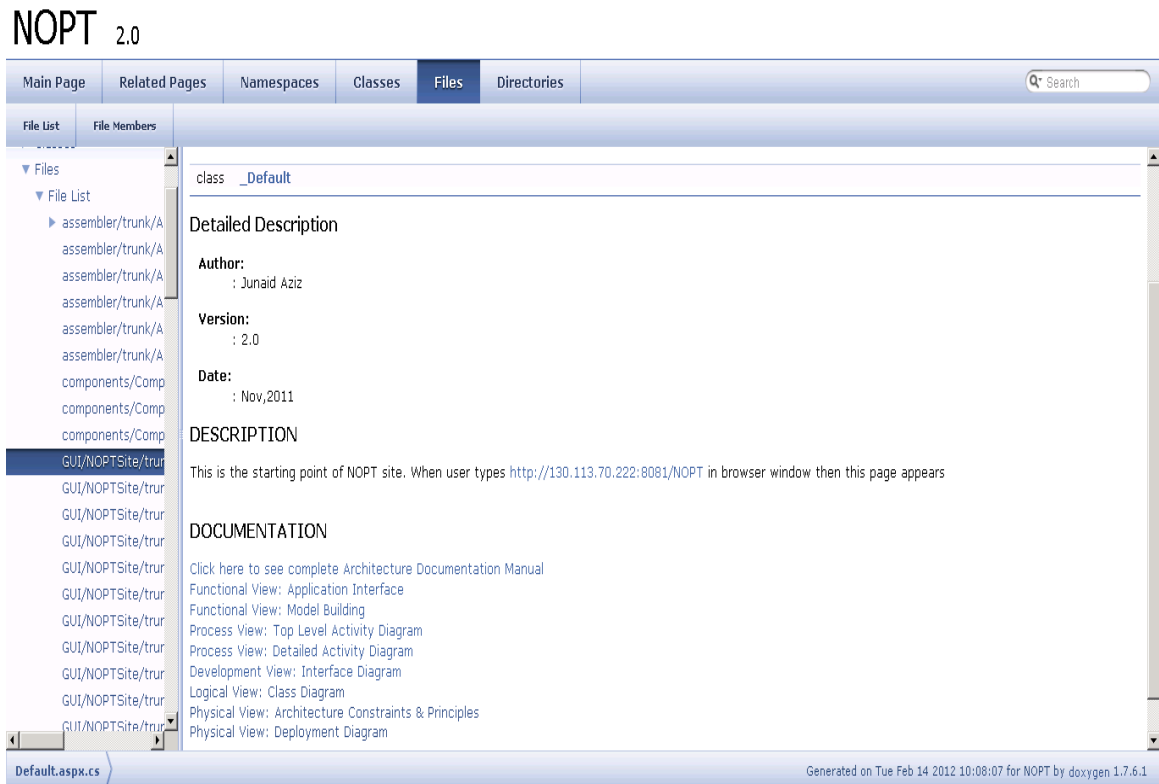


Figure 25: Showing links to UML diagrams



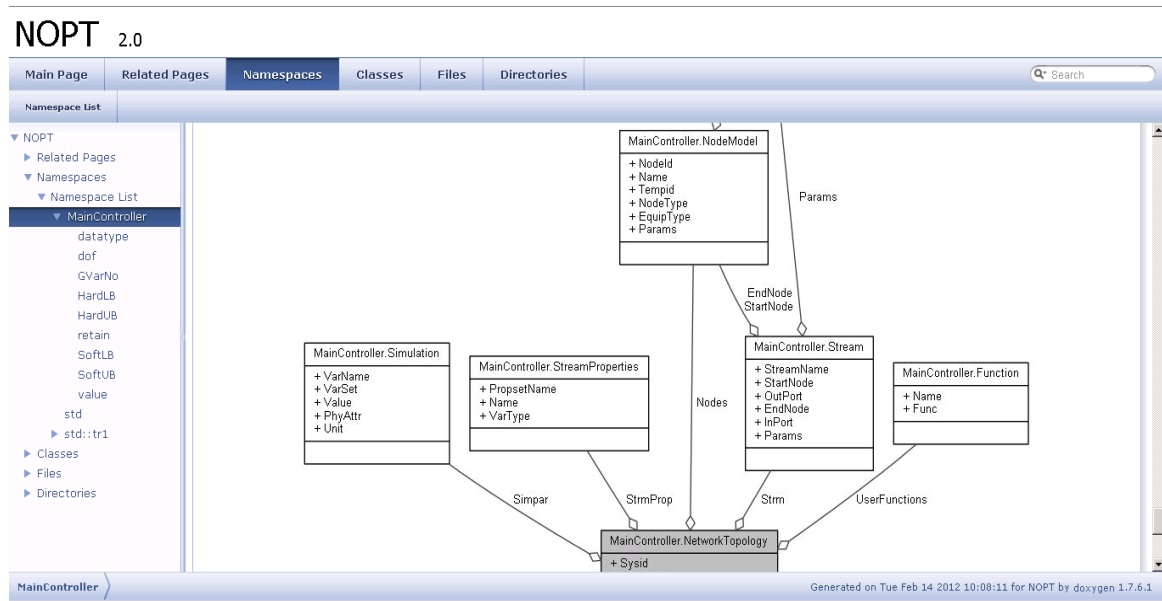


Figure 26: Partial Collaboration Diagram for Main Controller

NOPT 2.0

Main Page Related Pages Namespaces **Classes** Files Directories

Class List Class Index Class Hierarchy Class Members

List of all members.

**Public Member Functions**

```

Controller ()
void SetGenerateEquations (bool ge)
void SetGenerateVariables (bool gv)
void SetAssembleEquations (bool ae)
Variables getVars ()
void UpdateAll ()
int RunExternalApplication (String ApplicationFile, String Arguments)
bool ControllerFirstHalf (int NewSysId, string NewPath)
void ControllerSecondHalf ()
void RunController (int SysId, string NewPath)
    
```

**Static Private Member Functions**

```

static void OutputHandler (object sendingProcess, DataReceivedEventArgs outline)
    
```

**Private Attributes**

```

int Debug = 1
    Flag to enable the level of debugging, Debug=1 (Detail Debug), Debug=0 (Compact Debug)
bool CostNode = false
    Flag to enable whether there is a cost node in the model or not.
bool UserObjectiveFunc = false
    
```

MainController Controller

Generated on Tue Feb 14 2012 10:08:12 for NOPT by doxygen 1.7.6.1

Figure 27: Partial list of class members for Main Controller class

## Appendix B

---

### Use Case Details

---

<b>Use Case ID</b>	UC_1.1
<b>Use Case Name</b>	New User Registration
<b>Description</b>	User creates a new account to use the system
<b>Actors</b>	User
<b>Trigger</b>	User clicks the register link on the web page
<b>Preconditions</b>	User is not registered
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• User clicks the register link on the web page</li> <li>• User provides the information which is required for a registration process in html form</li> <li>• A registration request is made to the SharePoint server</li> <li>• Server validates the information and displays an error page informing the user to provide the correct/missing information if it finds any problem</li> <li>• If server is able to validate the information provided by the user a new account is created on SharePoint server and account information is stored in database.</li> </ul>
<b>Postconditions</b>	User is registered
<b>Dependencies</b>	N/A
<b>Use Case ID</b>	UC_1.2
<b>Use Case Name</b>	User Login

<b>Description</b>	User is Authenticated
<b>Actors</b>	User
<b>Trigger</b>	User clicks the login button
<b>Preconditions</b>	User provides his user name and password
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Authentication request is made to the server</li> <li>• Server validates the information and displays an error page informing the user to provide the correct/missing information if it finds any problem</li> <li>• If user is authenticated successfully home page is shown to the user</li> </ul>
<b>Postconditions</b>	User is authenticated
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.3
<b>Use Case Name</b>	User Logout
<b>Description</b>	User is Logged out from the system
<b>Actors</b>	User
<b>Trigger</b>	User clicks the logout link
<b>Preconditions</b>	User is logged in
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Check if user has made changes in the model and prompt him appropriately</li> <li>• Logout request is made to the server</li> <li>• Server accepts the request and displays an error page informing the user if it finds any problem</li> <li>• If user is logged out successfully, user is taken back to the main page of software</li> </ul>

<b>Postconditions</b>	User is logged out
<b>Dependencies</b>	N/A
<b>Use Case ID</b>	UC_1.4
<b>Use Case Name</b>	Clear Model
<b>Description</b>	Clear model information from the database
<b>Actors</b>	UC_1.3
<b>Trigger</b>	User initiates the logout process
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Connect to the database</li> <li>• Clear the model information from the database</li> <li>• Inform the web application that database has been cleared</li> </ul>
<b>Postconditions</b>	Model information in the database is cleared
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.5
<b>Use Case Name</b>	Edit Model
<b>Description</b>	User wants to edit his model
<b>Actors</b>	User
<b>Trigger</b>	User clicks the Model Name link on List Models page
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User is seeing the models list</li> </ul>

<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Check if model requested belongs to the current user or not</li> <li>• If it doesn't belong to the user display an error page saying "Access to the model is prohibited"</li> <li>• If it belongs to the current user then request for model detail is passed to the server</li> <li>• Model detail is retrieved from server</li> <li>• User is taken to the page where he can edit the model</li> </ul>
<b>Postconditions</b>	User is able to see the page where he can edit the model requested
<b>Dependencies</b>	N/A
<b>Use Case ID</b>	UC_1.6
<b>Use Case Name</b>	Save Model
<b>Description</b>	User wants to save the model
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User clicks the Save model button on Build Model page</li> <li>• User clicks the Save model button on Edit Model page</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User is finished building the model</li> <li>• User is finished editing the model</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Changes in the model are passed to the server</li> <li>• Server persists those changes in the model</li> <li>• Changes in the model are also passed to database</li> </ul>
<b>Postconditions</b>	Changes made in the model by the user are persisted and updated in database
<b>Dependencies</b>	<p>Include: UC_1.6.1, UC_1.6.2</p> <p>Extend: UC_1.5, UC_1.10</p>

<b>Use Case ID</b>	UC_1.6.1
<b>Use Case Name</b>	Persist Model on Server
<b>Description</b>	Changes made by user in the model are persisted
<b>Actors</b>	UC_1.3, UC_1.6
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User initiates the logout process</li> <li>• User made changes in the model</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User made changes in the model which were not persisted earlier</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Changes made in the model are added to the model structure in reusable format</li> <li>• Formatted structure is sent to the server for persistence</li> <li>• Server updates its local repository and acknowledges the application of its outcome</li> </ul>
<b>Postconditions</b>	Changes made in the model have been persisted successfully
<b>Dependencies</b>	N/A
<b>Use Case ID</b>	UC_1.6.2
<b>Use Case Name</b>	Save Model in database
<b>Description</b>	Model information is loaded in the database
<b>Actors</b>	UC_1.6, UC_1.12
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User clicked the save model button to save the model in database</li> <li>• User clicked the Run Model button to run the model</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User made changes in the model which were not saved in the database earlier</li> </ul>

<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Model information is forwarded to the database</li> <li>• Information is updated in the database</li> <li>• Acknowledgment of this action is dispatched to the web application</li> </ul>
<b>Postconditions</b>	Changes made in the model have been saved in database successfully
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.7
<b>Use Case Name</b>	List Models
<b>Description</b>	Models created by the users are displayed
<b>Actors</b>	User
<b>Trigger</b>	User clicks the Existing Models icon
<b>Preconditions</b>	User is logged in
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Network Models along with their sub network models are retrieved from server</li> <li>• Models created by the current user are labelled as “My Model”</li> <li>• Current user is able to edit, remove and run his own models</li> <li>• Models created by others are labelled as “Others”</li> </ul>
<b>Postconditions</b>	Models list is shown successfully
<b>Dependencies</b>	N/A
<b>Use Case ID</b>	UC_1.8

<b>Use Case Name</b>	Show Model
<b>Description</b>	User wants to see the model
<b>Actors</b>	User
<b>Trigger</b>	User clicks the Model he wants to see
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User is seeing the models list</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Request for model detail is passed to the server</li> <li>• Model detail is retrieved from server</li> <li>• If current user is the owner of the model then show an edit link and remove link next to the model otherwise simply show the model</li> </ul>
<b>Postconditions</b>	Requested model is shown successfully
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.9
<b>Use Case Name</b>	Remove Model
<b>Description</b>	User wants to remove the model
<b>Actors</b>	User
<b>Trigger</b>	User clicks the Remove Model link on List Models page
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User is seeing the models list</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Detail of the model user wants to remove is passed to the server</li> <li>• Server receives the requests and sends an error message if it finds any problem with the detail</li> </ul>



	<ul style="list-style-type: none"> <li>• Server sends a reply when model is removed successfully</li> <li>• User is notified that model has been deleted successfully</li> </ul>
<b>Postconditions</b>	Selected model is removed successfully
<b>Dependencies</b>	Include: UC_1.4, UC_1.9.1
<b>Use Case ID</b>	UC_1.9.1
<b>Use Case Name</b>	Change Model location on Server
<b>Description</b>	Keep a copy of the model on the server if model remove requested is received
<b>Actors</b>	UC_1.9
<b>Trigger</b>	User intends to remove a model
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User is seeing the models list</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Server acknowledges a model remove request</li> <li>• Server copies the persisted file containing model related information to a different location</li> <li>• Server removes the original file containing model information</li> <li>• Server informs the application that model is removed successfully</li> </ul>
<b>Postconditions</b>	Selected model is removed successfully
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.10
<b>Use Case Name</b>	Build Model Page
<b>Description</b>	Show Build Model page to the user

<b>Actors</b>	User
<b>Trigger</b>	User clicks the Build Model link
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• List of available business areas are loaded in a list</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Connect to the server</li> <li>• User selects the business area from a list of business areas</li> <li>• User clicks the build model button which shows him the work area where he can build a network model by dragging and dropping the node models and he can also create streams to connect the nodes</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• User is able to see the page where he can build a new model</li> </ul>
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.11
<b>Use Case Name</b>	Show Canvas
<b>Description</b>	Create a Canvas where user can build models
<b>Actors</b>	UC_1.5 , UC_1.10
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User clicks the Build Model button</li> <li>• User clicks the Model Name link on List Models page</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Create the Canvas area</li> <li>• Show the node models, stream classes belonging to the selected business area and work area where user can build a new model or edit an existing one</li> <li>• User is notified when he can start building a model or editing an existing one</li> </ul>

<b>Postconditions</b>	User is able to see the stream classes (lines), node models(shapes) and work area
<b>Dependencies</b>	Include: UC_1.11.1, UC_1.11.2

<b>Use Case ID</b>	UC_1.11.1
<b>Use Case Name</b>	Load library of node models (shapes)
<b>Description</b>	Library of node models are loaded when user goes to Build Model page
<b>Actors</b>	UC_1.11
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User clicks the Build Model button</li> <li>• User clicks the Model name link on List Models page</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has selected the business area</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Connect to the server</li> <li>• Retrieve the node models library file belonging to the business area from the server</li> <li>• Load the node models along with default properties such as template id and display them to the user</li> </ul>
<b>Postconditions</b>	User is able to see the node models
<b>Dependencies</b>	N/A
<b>Use Case ID</b>	UC_1.11.2
<b>Use Case Name</b>	Load library of stream classes (lines)
<b>Description</b>	Library of stream classes are loaded according to specific application area when user goes to Build Model page
<b>Actors</b>	UC_1.11

<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User clicks the Build Model button</li> <li>• User clicks the Model name link on List Models</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has selected the business area</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• Connect to the server</li> <li>• Retrieve the stream classes library file belonging to the business area from the server</li> <li>• Load the stream classes along with default properties attached and display them to the user</li> </ul>
<b>Postconditions</b>	User is able to see the stream classes (lines)
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.12
<b>Use Case Name</b>	Run Model
<b>Description</b>	Run the model
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User clicks the Run Model icon on List Models page</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• User sets the phases and time periods for the model</li> <li>• Model information is loaded into the database</li> <li>• Model information is dispatched to AEG</li> <li>• Acknowledgement of this action is received</li> <li>• AEG generates the equations and forward them to the solver</li> <li>• Solver solves the equations and store the results back in database</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• Model information is forwarded to AEG which starts generating equations and invoke the solver</li> </ul>

<b>Dependencies</b>	Include: UC_1.6.2,UC_1.12.1
<b>Use Case ID</b>	UC_1.12.1
<b>Use Case Name</b>	Set Phases and Time Horizon
<b>Description</b>	Before generating the equations for the model user sets the phases and time horizon
<b>Actors</b>	UC_1.12
<b>Trigger</b>	<ul style="list-style-type: none"> <li>User clicks the Run Model icon on List Models page</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>User is logged in</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>User sets the length of time horizon for the complete model including sub network</li> <li>User specifies the phases and which sub networks to be solved in every phase</li> <li>For every sub network in every phase user defines the number of time periods over the time horizon</li> <li>User sets the equation type for every sub network in every phase</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>Time horizon, number of time periods, phases and equation type for every sub network have been set successfully</li> </ul>
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_1.13
<b>Use Case Name</b>	Model Specification
<b>Description</b>	Change the specification of a model
<b>Actors</b>	User

<b>Trigger</b>	<ul style="list-style-type: none"> <li>User clicks the Model Specification link</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>User is logged in</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>List of all the models created by user is shown to the user</li> <li>User can go up to level 2 and join maximum of three models together</li> <li>After selecting the models to join user clicks the Change Model Specification button</li> <li>User can also use this page to split the models</li> <li>Once user is done all these changes are persisted on SharePoint server.</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>Model information is successfully forwarded to the SharePoint server</li> </ul>
<b>Dependencies</b>	Include: UC_1.13.1

<b>Use Case ID</b>	UC_1.13.1
<b>Use Case Name</b>	Persist Model Specifications on Server
<b>Description</b>	Store the information related to sub net work model on the SharePoint server
<b>Actors</b>	UC_1.13
<b>Trigger</b>	<ul style="list-style-type: none"> <li>User clicks the Change Model Specification button</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>User is logged in</li> <li>User has selected the models to join or to split</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>All the model information is forwarded to the SharePoint server</li> <li>Information is updated in the network model file on SharePoint server having information specifically about its sub network</li> </ul>

	<ul style="list-style-type: none"> <li>• Server sends a reply to the web application about the outcome of this action</li> <li>• Web application informs the user</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• Model file on SharePoint server having information specifically about its sub network is updated</li> </ul>
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_2.1
<b>Use Case Name</b>	Place Node
<b>Description</b>	Drag and Drop a node from node model ( A node which can only have one template attached to it but available in the toolbar as a standard shape) library on Canvas
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User wants to add node to a new/existing model</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has selected the node from the node model library</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• User selects a node from the node model library</li> <li>• User drags the node</li> <li>• User drops the node at the appropriate place on the canvas</li> <li>• Dropped node is shown to the user</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• Model is updated with the node user just dropped</li> </ul>
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_2.2
<b>Use Case Name</b>	Edit Node
<b>Description</b>	Show menu options to user related to the node
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User wants to change node attributes</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has selected the node</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• User selects the node he wants to edit</li> <li>• Menu appears, showing the user different options such as node name, set template etc. related to that particular node</li> <li>• User selects the particular option to make changes</li> <li>• Changes are applied to that node</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• Changes are applied to the node successfully</li> </ul>
<b>Dependencies</b>	Include: UC_2.2.1, UC_2.2.2

<b>Use Case ID</b>	UC_2.2.1
<b>Use Case Name</b>	Set Template
<b>Description</b>	Every node has at least one template and every template has different set of equations.
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User wants to set an appropriate template for a node</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has selected the node he wants to set template for</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• User selects the node to set template for</li> </ul>



	<ul style="list-style-type: none"> <li>User is given the options to select template he wants to set for this particular node and any calculations he wants to perform on the constants belonging to equations attach to this template</li> <li>User clicks Change button to apply the changes</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>Template is set for that particular node</li> </ul>
<b>Dependencies</b>	N/A
<b>Use Case ID</b>	UC_2.2.2
<b>Use Case Name</b>	Node Properties
<b>Description</b>	Set properties of a node
<b>Actors</b>	UC_2.2
<b>Trigger</b>	<ul style="list-style-type: none"> <li>User wants to set properties of a node</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>User is logged in</li> <li>User has selected the node he wants to set properties of</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>User selects the node</li> <li>User sets different properties such as level of demand if it is a demand node, level of supply if it is a supply node, quantity, Octain etc.</li> <li>User clicks set to make the changes</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>Node properties are set successfully</li> </ul>
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_2.2.2.1
<b>Use Case Name</b>	Create Time Sections for Demand/Supply Node
<b>Description</b>	Level of demand and supply are set for certain time periods

<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>User wants to set level of demand and supply of a node</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>User is logged in</li> <li>Node properties dialog is shown to the user</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>User sets the minimum and maximum level of demand if it is a demand node otherwise set minimum and maximum level of supply if it is a supply node</li> <li>User sets the start date, start time, end date and end time for level of demand/supply whichever is appropriate</li> <li>Level of demand or supply is split evenly between user defined time periods if minimum and maximum levels are different</li> <li>If minimum and maximum levels of demand or supply for a certain time period are equal then demand and supply will remain constant for that particular time period.</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>Levels of demand and supply have been split evenly between the specified time periods.</li> </ul>
<b>Dependencies</b>	Extend: UC_2.2.2

<b>Use Case ID</b>	UC_2.3
<b>Use Case Name</b>	Set Algorithm
<b>Description</b>	Sets the algorithm of the model
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>User wants to set an algorithm (LP, NLP, GLP) for the model</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>User is logged in</li> <li>User is finished building a model</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>User is given options to select algorithm for the model (default is LP)</li> <li>User selects the algorithm and its computational</li> </ul>

	parameters <ul style="list-style-type: none"> <li>• User clicks the Select button to set the algorithm for the model</li> <li>• Algorithm for the model is set</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• Algorithm for the model is set successfully</li> </ul>
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_2.4
<b>Use Case Name</b>	Connect Stream
<b>Description</b>	Connect a stream between two nodes
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User wants to connect two nodes with a stream in between</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has selected the stream from stream library</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• User selects a stream from the stream library</li> <li>• User drags the stream</li> <li>• User drops the stream at the appropriate place on the canvas</li> <li>• Nodes are connected with the stream just dropped by the user</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• Nodes are connected with the stream</li> </ul>
<b>Dependencies</b>	Include: UC_2.4.1

<b>Use Case ID</b>	UC_2.4.1
<b>Use Case Name</b>	Connection Node

<b>Description</b>	Split the stream into two streams and joining them by connection node
<b>Actors</b>	UC_2.4
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• Two nodes are connected by a stream</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has just connected two nodes by a stream</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• A stream is created to connect two nodes</li> <li>• This stream is split into two streams which are joined by a connection node</li> <li>• User is not seeing this connection node as it is part of business logic</li> <li>• Stream is changed successfully</li> </ul>
<b>Postconditions</b>	<ul style="list-style-type: none"> <li>• Connection node is created successfully to join the two sub streams</li> </ul>
<b>Dependencies</b>	N/A

<b>Use Case ID</b>	UC_2.5
<b>Use Case Name</b>	Stream Attributes
<b>Description</b>	Allow user to change stream attributes
<b>Actors</b>	User
<b>Trigger</b>	<ul style="list-style-type: none"> <li>• User wants to change stream attributes</li> </ul>
<b>Preconditions</b>	<ul style="list-style-type: none"> <li>• User is logged in</li> <li>• User has selected the stream</li> </ul>
<b>Interaction</b>	<ul style="list-style-type: none"> <li>• After selecting the stream user is able to change stream attributes</li> <li>• User attaches attributes to the stream such as amount, flows, temperature etc.</li> <li>• User is allowed to select system of units such as</li> </ul>

	<p>imperial, SI etc. SI is a default system of units.</p> <ul style="list-style-type: none"><li>• User clicks the Set Attribute button</li><li>• Attributes are set to the stream</li></ul>
<b>Postconditions</b>	<ul style="list-style-type: none"><li>• Attributes for the stream has been changed successfully</li></ul>
<b>Dependencies</b>	N/A

## Appendix C

---

### Node Model Design

---

NOPT workbench consists of library containing icon representation for every node model which is shown to the user when he starts building either a new model or update an existing model. These icons are created and maintained by the Administrator. This section describes the step by step procedure of designing and inserting a new node model into the library. All the node models shown on the canvas in NOPT workbench are saved in an xml file which can be edited by a small utility called “LibraryDesigner” provided by MindFusion. Designing and inserting a new node model is pretty straightforward using this utility. Firstly, an administrator has to draw a shape which he wants to include in node model library to represent a particular node model and then administrator assigns a display name and ID to this newly created shape. Display name of the shape is used as identification in the node model library file and that has to be unique in the library. ID should match with the EQUIP\_TYPE of the node model administrator wants this shape to represent for. Figure28 shows how LibraryDesigner utility can be used to design new shape for CostNode? Next and final step now is to set a unique display name and ID matching EQUIP\_TYPE which in this case is COST\_NODE as shown in Figure29. Now this new node model library file can be saved in NOPT workbench root directory to

become

available.

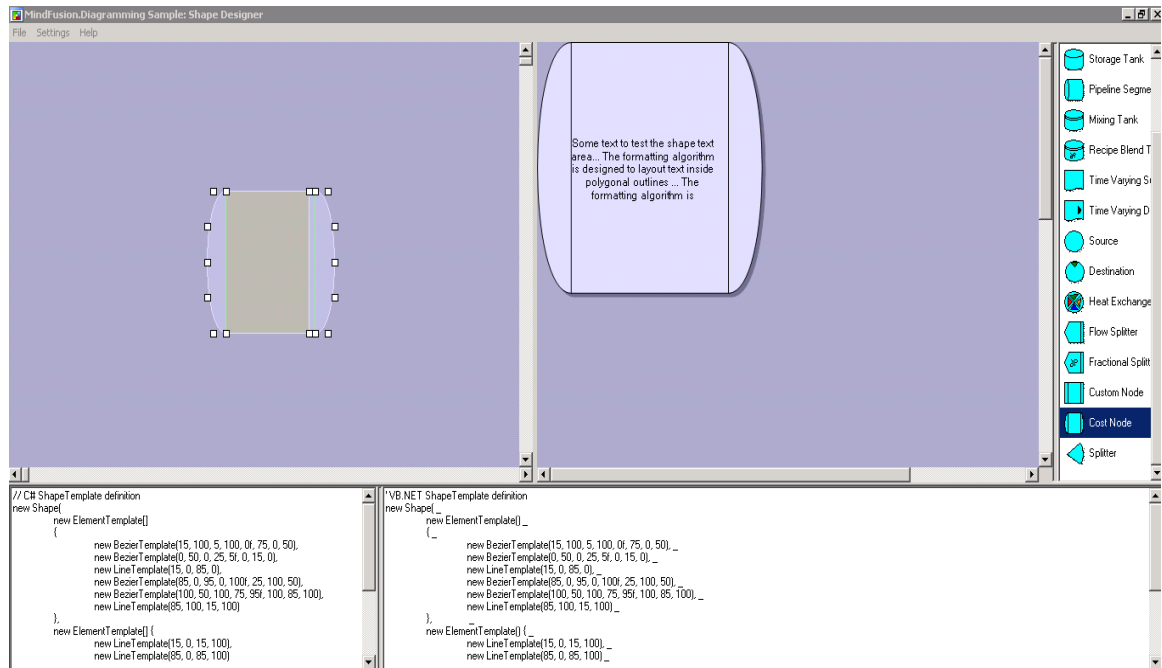


Figure 28: Designing Cost Node

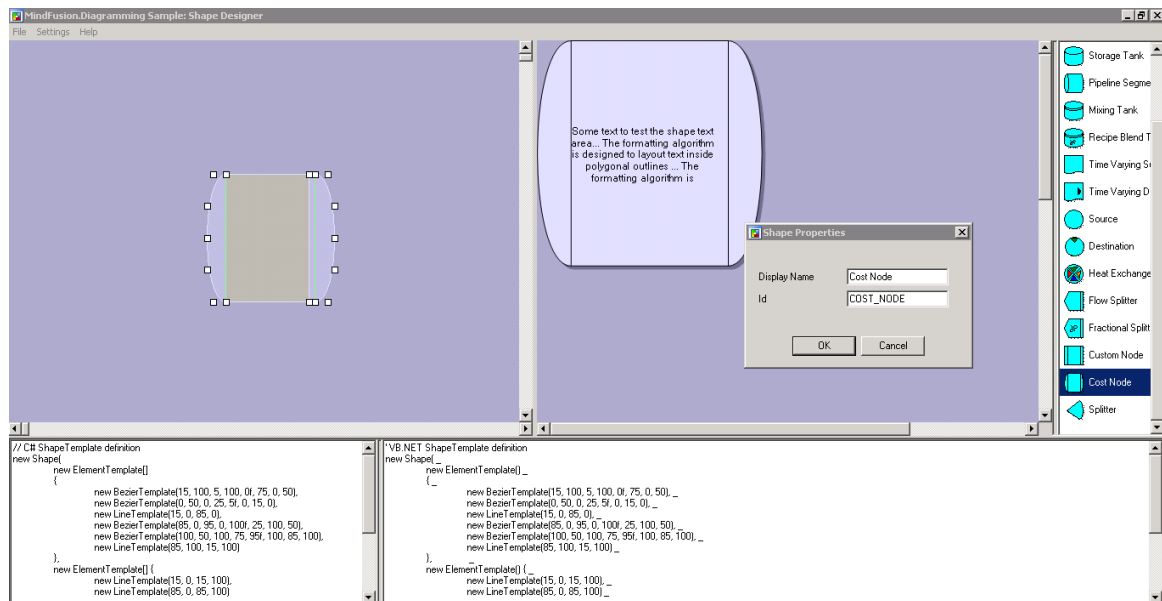


Figure 29: Setting Properties for Cost Node

## Bibliography

- [1] Bakhrankova, K. (2010): “Decision support system for continuous production.”  
Industrial Management & Data Systems, 110:4, pp. 591–610
- [2] Saad, S. M., Lau, K. H., Omer, A., (2009) "Design and analysis of oil production area-A simulation approach" the 23rd European Conference on Modelling and Simulation, Madrid, Spain., June 9-12.
- [3] Herran-Gonzalez A. De La Cruz J. M. De Andres-Toro B., Risco-Martin J.L.  
Modeling and simulation of a gas distribution pipeline network (2009) *Applied Mathematical Modelling*, 33 (3), pp. 1584-1600.
- [4] Heckl I., Kalocsai P., Halasz L. and Kalauz K., (2009), Event driven process simulation of pipeline networks, *Chemical Engineering Transactions*, 18, 737-742  
DOI: 10.3303/CET0918120
- [5] F. Rømo, A. Tomasgard, L. Hellemo, M. Fodstad, B.H. Eidesen, and B. Pedersen,  
"Optimizing the Norwegian Natural Gas Production and Transport", presented at  
*Interfaces*, 2009, pp.46-56.
- [6] Cafaro, D. C.; Cerda, J. Efficient Tool for the Scheduling of Multiproduct Pipelines and Terminal Operations. *Ind. Eng. Chem. Res.* 2008, 47, 9941–9956
- [7] <http://www.stack.nl/~dimitri/doxygen/>
- [8] <http://www.uml.org/>



[9] <http://staruml.sourceforge.net/en/>

[10] <http://www.mindfusion.eu/netdiagram.html>

[11] Allan D Waren. Omega: An improved gasoline blending system for texaco. Interfaces, 1989:85-101, 1989.

[12] A Singh, J F Forbes, P J Vermeer, and S S Woo. Model-based real-time optimization of automotive gasoline blending operations. Journal of Process Control, 10, 2000.