EVALUATION OF POLICIES FOR THE MAINTENANCE OF BRIDGES USING DISCRETE-EVENT SIMULATION

Srinath Devulapalli

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements of the degree of

Master of Science

In

Civil Engineering

Dr. Julio C. Martinez, Chair

Dr. Jesus M. de la Garza

Dr. Gerardo Flintsch

August 16th 2002

Blacksburg, Virginia

Keywords: Bridge Management, Policy Analysis Tool, Discrete Event Simulation

EVALUATION OF POLICIES FOR THE MAINTENANCE OF BRIDGES USING DISCRETE-EVENT SIMULATION

Srinath Devulapalli

(ABSTRACT)

With the recent developments of several bridge managements systems and their wide-spread use, bridge engineers are realizing the importance of systematic and well planned investments and appropriate management. However the results are far from satisfactory. Bridge management systems need more effective policy analysis tools that can take advantage of the vast amounts of available information to be more efficient.

The objective of this research is to develop a policy analysis tool, which is generic in nature and can be applied to any bridge management system provided all the appropriate data is available. In particular, this policy analysis tool is geared to suit policy making, planning and budgeting for the interstate bridges in the state of Virginia.

The policy analysis tool developed in this research is a discrete event simulation model capable of extracting information from text files in the Pontis Data Interchange format and simulate user defined element level policies. The model testing was performed using the interstate bridges of the Salem district in Virginia. All the relevant information was extracted from their PONTIS databases.

Several scenarios with varying network policies were simulated. The results indicate the validity and the accuracy of the model. The policy analysis tool is a useful addition to the existing policy analysis tools and is capable of handling probabilistic distributions of data instead of single value averages. This will enable the tool to capture more information thereby making the simulation model more realistic.

The general framework that was developed here can be applied to any infrastructure problem, and eventually it should be possible to achieve a discrete event simulation based integrated infrastructure management system.

ACKNOWLEDGEMENTS

The support of all my committee members Dr. Julio C. Martinez, Dr. Jesus M de la Garza, Dr. Flintsch and Mr. Allen Williams is greatly acknowledged. I would especially like to thank my guide Dr. Martinez for his constant encouragement and support. Thank you for being there whenever I needed you.

I would also like to thank Mr. Dean Hackett from the Virginia Department of Transportation. His constant feedback was absolutely critical for the completion of this project. Special thanks to Juan Pinero for his help and enthusiasm in the development of the project. I would also like to thank my friend Jitamitra Desai for just being there whenever I needed to talk to someone.

I would also like to thank my family for shaping me into the person I am and ultimately responsible for this work.

I would like to take this opportunity to thank the Virginia Department of Transportation and the National Science Foundation for funding this project and providing me with this unique opportunity.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

Chapter 1

INTRODUCTION

1.1 BACKGROUND

A Nation's productivity and internal competitiveness depends on fast and reliable transportation. According to the "2001 ASCE Report Card for the Nation's Infrastructure", 29% of Virginia's highway bridges are functionally obsolete or structurally deficient. These deficient structures represent significant impediments to the safe, economical use of the highway system and result in safety hazards, high user costs, and huge outlays for preservation and replacement. Balanced against this backlog of bridge needs is a generally inadequate level of funding by public agencies for infrastructure needs.

The collapse of the Silver Bridge in 1967 was the immediate catalyst for what became a comprehensive bridge safety inspection program mandated by the National Bridge Inspection Standards (NBIS). According to this program, created by the Federal Highway Act of 1968, every bridge on a public road must be inspected at least every 2 years and highway agencies across the nation have inspection staffs and programs that collect and update critical bridge inventory and inspection data. The Federal Highway Administration (FHWA) maintains this information in a comprehensive NBI database. Based on their study of the NBI database, Small and Cooper (1998) concluded that during the period of 1983 to 1993 even though the total number of deficient structures and associated percentages have been decreasing, this trend can be attributed not to sound management principles or increased level of funding over the years, but due to the use of new materials with increased durability and fewer maintenance requirements. "As the inventory continues to age, additional demands will be placed on bridge engineers and managers. In particular, the large volume of structures built during the Interstate era will require increased maintenance, major rehabilitation, and in some cases replacement. Given these projected needs and anticipated static budgets, further progress in the

removal of deficiencies is out of the question" (Small and Cooper, 1998).

Bridge owners today should be able to make decisions pertaining to maintenance and improvement that take into account both the financial constraints as well as the overall needs of the highway system. The complexities and costs associated with preserving the nation's bridge infrastructure demand innovative approaches to collection and analysis of data and prediction of current and future bridge preservation actions. These needs, coupled with the availability of modern analytical methods and high-speed computers, are leading to the development of comprehensive bridge management systems.

1.2 COMPONENTS OF A BRIDGE MANAGEMENT SYSTEM

In 1986, a demonstration project was initiated that supported workshops in almost every state and sought to develop sound bridge management practices. Based on this, the American Association for State Highway and Transportation Officials (AASHTO) in conjunction with several other DOT's, was able to establish the primary requirements of a comprehensive Bridge Management System (BMS) and the key features required by any effective Bridge Management System.

Based on the guidelines provided by AASHTO the key components of a Bridge management system have been identified as

1. Data Storage.
2. Deterioration Models.
3. Cost Models.
4. Policy Analysis tools.

Figure 1.1 depicts the different components of a Bridge Management System as described above. These components integrate seamlessly to aid the decision maker in managing the Bridge Network.

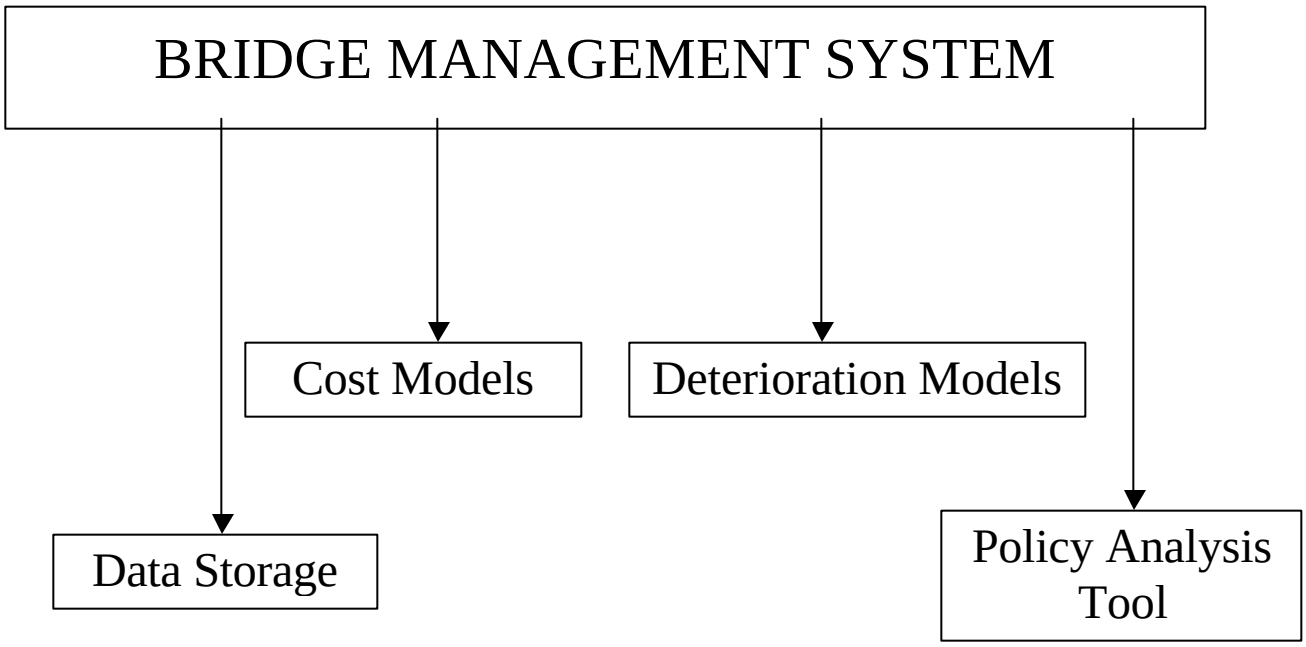

FIGURE 1.1 Components of a Bridge Management System

1.2.1 Data Storage Module:

Data collection and data analysis are essential components of a bridge management system. The database connected to the bridge management system stores data from periodic field inspections. Regardless of the techniques used for data analysis, the success of the policy analysis and bridge management system in general depends upon the quality and sufficiency of the data collected to a great extent. Information stored in the database is used as input into the model. The models are used to predict the future condition and perform "what-if" analyses under different scenarios to determine the impacts on the network. The two types of models are deterioration, and cost models.

1.2.2 Condition Prediction / Deterioration Models

An important purpose of data analysis is the prediction of future bridge conditions under different strategies. For this, several data analysis techniques such as Regression analysis, Markov chains (Schrer and Glagola, 1994), Bayesian estimation and Fuzzy set theory have been successfully applied for the prediction of bridge element condition, user costs and maintenance estimates (Kleywegt and Sinha, 1994). These models effectively

process the database of inspections and arrive at generalized behavior of different components of the bridge that enables them to predict the future condition of a bridge.

1.2.3 Cost Models:

A bridge management system typically estimates two types of costs: improvement and agency. Improvement costs are estimated to determine the cost of a maintenance action to improve the condition of the element. The agency cost could include the expected user cost savings for safety and serviceability improvements. Typically these costs are modeled and updated using Bayesian estimates. One disadvantage of this method is that typically the cost models estimate the cost as a single value, which could be obtained as a weighted average of historic costs and expert opinions. However, lumping all this information into a single value loses the inherent uncertainty in costs. Instead a suitable mechanism should be developed in which distributions are developed for the costs.

1.2.4 Policy Analysis Tools:

The objective of any BMS is not only data storage but also data analysis and the determination of the best maintenance policy. So we need a procedure that compares different policies and determines the best policy to be followed. Several tools have been developed to aid the decision makers in allocating the funds among the competing needs. Some of the common techniques in use are priority setting and optimization. Specifically, linear programming, dynamic programming and integer programming are commonly used optimization techniques. Also the author has developed a discrete-event simulation based policy analysis tool.

1.2.4.a *Priority Setting*

The competing bridges are ranked according to certain criteria so that that the decision maker can focus his attention on the most critical bridges rather than all the bridges. Several ranking methods have been developed to aid in the priority setting. One such method is the sufficiency rating developed according to FHWA's Structure Inventory and Appraisal Guide (1979). Another such method is based on the level of

service. The California DOT has developed a Health Index (HI) in which the bridges are ranked based on its structural condition.

1.2.4.b *Optimization*

The purpose of optimization is to find the optimal set of actions to be implemented. For this a number of techniques can be used.

Minimization of life cycle costs is a bridge level approach to policy analysis. All the alternatives are considered and the promising alternative is implemented for that programming period for every bridge. This essentially reduces to continuing routine maintenance until either rehabilitation or replacement is better. This method does not arrive at the network level global optimum, as it does not consider network level budget constraints into account.

In Linear Programming the value of an objective function is maximized or minimized by changing decision variables, subject to certain linear equality/inequality constraints. One formulation of the Bridge Management problem in linear programming can be – "maximization of the Level of Service of the bridge network by adjusting the maintenance policies subject to budget constraints". If the decision variables are not continuous variables but instead integral values, the problem can be solved as an Integer Linear Programming problem. The main advantage of LP and ILP methods is that the techniques are easy to use and can be applied to problems with hundreds or thousands of variables. But the disadvantage of these methods is that the objective functions and constraints are restricted to linear functions of the decision variables. Non-linear programming techniques can be used for non-linear functions but they are computationally much more demanding and a solution cannot always be determined.

Dynamic Programming is another optimization approach. It is based on the Principle of Optimality, which states that "Optimal policies over time consist of optimal sub alternatives over shorter periods of time". Thus, optimal long-term policies can be viewed as short-term sub policies over successive periods of time. The optimal

alternative can be calculated recursively for each state at each sub time period thereby arriving at the long time optimal strategy.

1.2.4.c *Systems Dynamics*

Systems Dynamics has also been employed as a policy analysis tool (Kim 1996, Kim 1998.). Systems Dynamics views the problems of Bridge Management as a cause-and-effect system. Investment in facility maintenance will upgrade the physical condition of the assets, but physical deterioration will take place simultaneously. If the investment is sizable enough, the physical condition of the facilities will be maintained in good condition. An improper allocation of the funds may result in further deterioration of facilities. Thus in the long run an optimal allocation of funds will maintain the assets in required criteria. Systems Dynamics tries to formulate this equilibrium in terms of differential rate equations and solve them.

1.3 NEEDS FOR A POLICY ANALYSIS TOOL

Any Policy Analysis tool involves decision making related to selecting the most cost-effective bridge improvement strategies at both the project and network levels. The typical variables which go into this analysis include agency costs, user costs, discount rates etc. However it should be realized that uncertainties could exist in these decision variables. These uncertainties are of two types – physical uncertainties and modeling uncertainties. Physical Uncertainty represents the physical variability that is associated with a particular parameter. For example: The cost of repairing a pot hole is never a unique value of say $300 but ranges between $200 and $450 based on several factors like the extent of damage, existing conditions, type of pavement etc. Modeling uncertainty on the other hand is associated with idealizations that are used in assessing a particular effect. For example when trying to asses the condition of a bridge, different inspectors might rate the bridge differently. “Uncertainties can easily lead to making the wrong decisions, especially in selecting the best from pairs of closely ranked competing strategies” (Sobanjo, 1999). The same argument applies to the condition prediction models. The modeling uncertainty can be reduced through the use of standardized

inspection and recording procedures. The physical uncertainty in costs and deterioration rates is inherent because in real life the cost of the maintenance activity or the deterioration is not always the same value but dependent on several factors like weather, quality of work etc. none of which can be easily quantified. Therefore, trying to obtain a single valued average will result in error and these errors could add up and lead to wrong conclusions. An alternative approach is to employ probabilistic distributions fitted to the parameters based on statistical analysis of reliable and available historical data. However most of the existing policy analysis tools cannot handle data based on probabilistic distributions of data.

Discrete Event Simulation is a tool that can effectively handle probabilistic distributions of data. However any realistic simulation model is computationally demanding thereby limiting their use to date due to lack of available technology. With the advances of computational power in the last decade, it has been made possible to develop and test realistic simulation models. These models will help the decision maker to capture all the inherent uncertainty in the data and yet to a reasonable level of reality simulate the real life scenarios (Martinez and Ioannou, 1999).

It is known that it is possible to model any type of system using Discrete Event Simulation (DES), provided that detailed discrete data about the individual elements that make up the model is available (Law & Kelton, 2000). It is therefore possible to make a DES model of a bridge system and incorporate into that the specific strategies of funding for a given scenario. Ever since the passing of the FHWA act of 1968, every state has been collecting detailed condition information about its bridges which can drive a DES model. However this merely reduces to a heuristic model building, coding and a single run of the program to obtain the “answer”. Any changes in the policies or the condition data renders the model useless thereby making this exercise cost ineffective.

The purpose of this research is to investigate how a DES model that is generic in nature can extract data from the available sources, can incorporate how an agency spends the budget and can provide test strategies that are independent of the model and the

network data. Such a model can be populated with any network data and incorporates the decision making process. Such a goal is not specific to bridge management alone and its concepts can be extended to asset management in general by a slight modification of the principles used and the underlying assumptions.

Chapter 2

LITERATURE REVIEW

Prior to the late 1980's, there were no management systems adaptable to the management of bridge programs nor was there any clear definition of key bridge management principles or objectives.

From the 1980's considerable amount of research has been done in developing a comprehensive Bridge Management system, which will aid the user in making crucial decisions pertaining to bridge maintenance. Based on this research several bridge management systems have been developed.

2.1 BRIDGE MANAGEMENT SYSTEMS IN EXISTENCE

PONTIS and BRIDGIT are two nationwide projects with a generic design that can be adapted to accommodate the individual needs of an agency. Based on the survey conducted by the FHWA, 42 states indicated that they intended to implement PONTIS (Khan, 2000). Small and Cooper (1998) recorded that as of May 1998, PONTIS was licensed to 38 states, the District of Columbia, Sacramento and Los Angeles Counties, City of Los Angeles, and the New Jersey Turnpike Authority. Maine and a few other states were looking at implementing BRIDGIT (Khan, 2000).

2.1.1 Pontis

PONTIS ("pontis" is the Latin word meaning "pertaining to bridges") is a bridge management system developed by the Federal Highway Administration (FHWA) in conjunction with six state DOT's and the consultant joint venture of Optima Inc. and Cambridge Systematics. To quote the Pontis User's Manual, "Pontis is a new generation network-level bridge management system which incorporates dynamic, probabilistic models and a detailed bridge database to predict maintenance and improvement needs, recommend optimal policies and schedule projects within budget and policy restraints." Pontis has a database containing the bridge condition data, traffic needs, deterioration

models, accident costs, maintenance, improvement and replacement costs. Using this data Pontis prioritizes the bridges and allocates the limited budget to the bridges to derive maximum benefit for the entire system (Thompson and Shepard, 1994).

2.1.2 Bridgit

BRIDGIT is a bridge management system developed jointly by National Cooperative Highway Research Program (NCHRP) and National Engineering Technology Corporation. BRIDGIT is very similar to Pontis in terms of its modeling and capabilities. The system requires data at an element level and reports the condition of the elements in terms of condition states. Deterioration is modeled as a Markov process. Cost models are addressed in a similar fashion (Lipkus, 1994).

2.1.3 Local Bridge Management Systems

In addition to PONTIS and BRIDGIT, there are several other local bridge management systems that have been adopted by a handful of states. The Alabama DOT (Green and Richardson, 1994), Connecticut DOT (Lauzon and Sime, 1994), Indiana DOT (Woods, 1994), North-Carolina DOT (Johnston and Lee, 1994) and PennDOT (Oravec, 1994) are a few state agencies that have opted to develop their own Bridge Management System as opposed to using an of the shelf software.

2.2 POLICY ANALYSIS TOOLS

2.2.1 Priority Setting

Priority Setting is used in some Bridge Management systems like North Carolina DOT (Johnston and Zia, 1984, Johnston and Lee, 1994) and in a few others (Kleywegt and Sinha, 1994). The advantage of Priority Setting is that it helps the decision makers to narrow their focus on a short range of critical bridges rather than the entire list of bridges. Some variations of the Priority Setting are Concordance analysis and Analytical hierarchy process (Kleywegt and Sinha, 1994). PennDOT's bridge management system has a decision support component that is based on priority setting. It prioritizes bridges according to their maintenance and improvement needs based on a deficiency rating

(Oravec, 1994). The main disadvantage of all such pair-wise comparison methods is that as the number of alternatives increase, the number of pair wise comparisons increases significantly.

Larsen and Holtz (1999) discuss two extensions of Priority ranking – Net Present Value method and Point ranking method. The Net present value method tries to compare the current value of future costs by taking into account factors like inflation, and selects the alternative with the lowest cumulative present value. This method makes it possible to compare strategies for which the costs are spread over a period of time. The point ranking method can be used as an alternative or addition to the net present value method. It calculates the final priority ranking point based on the weighted points assigned to bridge condition, load capacity, and clearance. Then this weighted average is weighted by the importance of the road structure (usually a function of traffic volume, location, importance etc.) to arrive at a common platform for comparing different bridges.

2.2.2 Optimization

Optimization is the most widely used technique in most Bridge Management Systems (Johnston et al, 1994, Hyman and Hughes, 1983). PONTIS uses Linear Programming (Golabi et al, 1992) as its policy analysis tool. It determines the long-term optimal policies for each element definition and then applies them to each bridge and tries to schedule projects subject to the project constraints.

Indiana BMS has applied Integer Linear programming (ILP) to generate optimal policies for bridge systems (Vitale et al, 1996). The formulation tries to maximize the net utility value of the network subject to the budget constraints. The utility actually is a function of (a) structural condition, (b) safety, (c) cost effectiveness, and (d) community. Each objective is quantified using one or more utility curves. A set of 17 maintenance activities has been considered in the analysis. This method does not take into account the durations of the maintenance activities. In reality, all the bridges are not closed for maintenance simultaneously but are spread over the year. This is not modeled in the formulation. The model cannot handle stochastic distributions of maintenance costs or

utility functions. ILP has been suggested as a further improvement to the existing Linear programming module of PONTIS (Golabi et al, 1992). However as observed by them the main disadvantages of ILP as opposed to Linear programming are that the computation time scales much faster as the size increases and the objective functions and constraints are restricted to linear functions of the decision variables. Though non-linear programming techniques can be a solution, they are computationally more demanding.

Dynamic Programming is also being considered as a viable extension to the optimization module of PONTIS (Golabi et al, 1992). Indiana DOT (Woods, 1994) uses dynamic programming in combination with ILP for its optimization module. The planning horizon is split into smaller time zones and the long-term optimal plan is considered to be the sum of the short-term optimal policies of the individual time zones (Dynamic Programming based on Principle of Optimality). At each of these stages the short-term optimal policies are arrived at using ILP. North Carolina's BMS has the OPBRIDGE module responsible for determining the optimum improvement action and time for each bridge and it is based on the same principle (Johnston, 1992). However this module is also based on the NBI rating rather than the CoRe element level classification.

2.2.3 Other Policy Analysis Tools

Genetic Algorithms have been applied to the problem of optimization (Miyamoto et al, 2000). This algorithm is based on the theory of evolution, and using three operators (selection, mutation and crossover) creates a suitable individual solution. The algorithm was applied in conjunction with a bridge management system (Concrete Bridge Rating System) to analyze existing concrete bridges.

Smilowitz and Madanat (2000) have addressed the issues of errors and uncertainty in inspections and non-fixed interval inspections using latent Markov decision process (LDMP). This method originally developed from structural reliability tries to predict the future condition of the element based on all its previous condition states as opposed to the traditional Markov decision process, which simply predicts the future condition based on the current conditions.

Frangopol et al (1999) developed an optimization model based on reliability of the bridges as opposed to the condition state information. This optimization problem consists of minimizing the total expected cost under reliability constraints, which ensure that the bridges have a certain minimum reliability at all times. Since reliability is not a single value but rather a random variable, the model is capable of handling probabilistic distributions of data. Flaig and Lark (1999) proposed the same approach but with a small change. They contest that the risk calculation should be based on the consequences of a bridge failing and not just on the relative importance of a bridge. Cesare et al (1993) have also proposed a reliability based policy analysis tool as an alternative to condition data. Due to the inapplicability of "Full search" and "branch-and-bound" algorithms, they have proposed a solution to the problem using genetic algorithms.

Bieñ (1999) developed an expert system policy analysis tool based on artificial neural networks and analytical functions. This tool has the ability to learn, recognize and choose different goals and try to achieve these goals. The expert functions are based on data collected and on the knowledge represented in the computer system. This expert system has been tested with the Railway Bridge Management system in Poland.

Kleywegt and Sinha (1994) discuss several heuristics developed by researchers. Garcia-Diaz and Liebman (1983) developed an investment staging model for bridge replacement and scheduling of bridges. However these heuristics do not address the entire bridge management issue as a whole but instead address specific issues in Bridge Management.

Kim (1996) developed an integrated infrastructure-planning model for pavements and bridges. This approach determines the causal relationships between the different components of a transportation network and tries to solve the problem based on systems dynamics. The problem with this model is the same in that it can handle only discrete values for the decision variables. Kim (1998) extended this systems dynamics model and validated its applicability to several demographic areas. This planning model was able to predict the future conditions over a 20-year period.

Not much research has been done in the application of discrete event simulation as a policy analysis tool. PONTIS version 4 has a simulation module that simulates the network behavior but neither the PONTIS User's manual nor any other literature sheds light on the inner workings of this simulation module. However the PONTIS simulation module cannot handle distributions for the cost models or the deterioration models. Neither does the PONTIS simulation model take into account the durations of the individual maintenance actions leading to the belief that its working is probably not discrete event simulation based.

Chapter 3

OBJECTIVES

There is a lot of investment in bridges and a lot more should be invested to achieve the desired level of service. It therefore pays off to invest wisely. Several bridge management systems have been developed to aid the policy makers in making better decisions. The key component of bridge management systems is a Policy analysis tool.

Optimization, Markov chains and Systems Dynamics are the three main policy analysis tools which have been employed to date in bridge management systems. Optimization techniques are limited by the complexities of the problem they can solve and the computational time scales with the magnitude of the network. Systems Dynamics is a methodology that uses control theory and feedback structure for a system to analyze the problem. Markov chains assume that everything prior to the current condition does not matter and predict the future condition based on transition probabilities.

The main problem in bridge management is the vast amount of information that should be captured for analysis. Each bridge is unique with its own deterioration rates condition states, maintenance costs and improvement needs. Systems Dynamics is based on the principles of interactions between deterministic, dynamic, non-linear and closed boundary systems (Kim 1996). The systems approach described in Kim (1996) tries to define the different sub-systems existing in a transportation network and defines the relationships between them. It then predicts the behavior of the sub-systems under varying external conditions. The drawback of systems dynamics approach is that it cannot take into account intangible factors like weather in a quantitative manner. Simulation can handle such factors by using distributions for the data instead of deterministic values.

Discrete event simulation based models can be developed for any problem provided that suitable discrete state information is available. Therefore it is possible to

incorporate the available information about a specific bridge sub-system into a Discrete Event simulation model and run simulations to observe the network behavior. However this approach has some shortcomings. First the model will be problem specific and cannot be used for a different sub-system unless it is extensively modified to suit the new sub-system. Further the model will need to be tweaked constantly to keep it up to date. Any changes in the network structure could mean changing the model to incorporate these changes. All this involves significant work rendering this solution cost-ineffective.

This research tries to develop a suitable and generic framework for employing discrete event simulation models to solve the problem of bridge management thereby adding simulation as a policy analysis tool to the already available policy analysis tools. Based on this framework, a DES based model was developed. This model is generic in nature and can extract data from available sources, thereby making sure that it is not problem specific, but can be applied to any bridge network, provided all the compatible data is available.

3.1 SPECIFIC OBJECTIVES

This research will address the following issues

1. To develop a suitable framework for employing DES as a general purpose policy analysis tool in bridge management in particular and asset management in general.

2. To explore the usefulness of this tool in bridge management.

3. To determine the capabilities, limitations and requirements of such a tool.

4. To determine the kind of strategic policies that can be modeled using this tool.

Chapter 4

RESEARCH METHODOLOGY

4.1 UNDERSTANDING BRIDGE MANAGEMENT

The first step in the development of a simulation model is to formulate the problem. This involves a thorough understanding of the bridge management problem and the decision making process of the personnel involved. Through interviews with several experts and literature review, a thorough understanding of the issues in bridge management was obtained. What follows is a description of the main problems involved in bridge management which distinguish it from any other asset management.

4.1.1 Description of the Bridge Management Problem

The primary difficulty with bridge management is the vast number of different bridge types in existence (Cable Stayed, Reinforced Concrete, Timber etc.). Each type of bridge has its own components, deterioration rates, maintenance schedules and costs, which are unique to that type; bridges cannot be viewed as a generic 'Bridge' type. So the bridges need to be classified into different categories based on their common properties. Different approaches have classified the bridges based on their components. One of the most common approaches is the National Bridge Inventory (NBI) classification where each bridge is classified into major components like Deck, Superstructure, Substructure and Other Minor Elements. The status of a bridge is a function of the health of these components. However one major limitation with this method is that it classifies the bridges into broad categories and the properties might vary significantly within a category. The major components of a bridge consist of many elements, materials, possibly different functions for the same element and different quantities of the same element. Each of these elements behaves differently over time as a function of the load and environment that they are subjected to. Thus, by collecting various amounts of information on the components and lumping all that information into one number grossly reduces the value of the information gathered.

This difficulty can be overcome by further classifying the categories into sub-components. The Commonly Recognized (CoRe) element classification of Bridge elements prepared by the FHWA is much more detailed than the NBI classification. This classification is used in PONTIS, BRIDGIT and other Bridge Management systems. What follows is a description of the CoRe element classification.

There are about 160 elements that make up the entire range of bridge components. An element is a basic functional unit of the bridge, such as a girder, bearing pad, column, pier cap, deck or joint. Every bridge will not have all these elements but will most likely be made up of about a dozen of them. The list of all the elements used in PONTIS classification for the state of Virginia is given in Appendix A. Each element has a set of defined condition states. This classification achieves a greater level of detail. There are at least three different states and at most five different states for each element. These condition states are defined using engineering language applicable to that element. For example Table 4.1 shows the condition states for the painted steel deck truss element.

TABLE 4.1 Condition State Definitions for Painted Steel Deck Truss (Element 131)

State	Description of the Condition State
State 1:	There is no evidence of active corrosion and the paint system is sound and functioning as intended to protect the metal surface.
State 2:	There is little or no corrosion. Surface or freckled rust has formed or is forming. The paint system may be chalking, peeling, curling or showing other early evidence of paint system distress but there is no exposure of metal.
State 3:	Surface or freckled rust is prevalent. There may be exposed metal but there is no active corrosion, which is causing loss of section.
State 4:	Corrosion may be present but any section loss due to active corrosion does not yet warrant structural analysis of either the element or the bridge.
State 5:	Corrosion has caused section loss and is sufficient to warrant structural analysis to ascertain the impact on the ultimate strength and/or serviceability of either the element or the bridge.

In addition to the condition of a bridge element, its environment and the effects of traffic and aging govern the behavior of an element. Every condition state has three associated action types, two of which will restore the element to a better condition and a Do Nothing activity, which simply lets the element deteriorate at its natural rate.

For example, consider the Painted Steel Deck Truss (element 131) in condition state 3. There are three possible types of maintenance actions as given in table 4.2.

TABLE 4.2 Maintenance Actions for Painted Steel Deck Truss

Maintenance Action Type	Description of the Maintenance Activity
0	Do Nothing
1	Remove Surface Rust
2	Clean and Spot Paint

Another difference in this rating from the NBI rating is that it rates bridge elements in quantitative units rather than an average rating for the entire element. Since total quantities of an element are rated, it is reasonable that an element can have multiple condition states. For example: beam-ends. If there are 4 girders, each 100 feet long (a total of 400 LF). However, usually only the beam ends have advanced deterioration due to joint leakage. The entire 400 LF would not be coded in condition state 4 but only a portion of the total amount, say 5 LF per beam end under the open joint. Thus only 5% of the beam-ends will be rated in condition state 5. The remaining 95% is placed in condition state 3. Therefore different portions of an element can be in different condition states. The Element Data Collection Manual (1996) gives a detailed description of the condition states and the way in which they are assigned.

Bridge Management as mentioned earlier is a complex problem. Some of the issues arising in the development of a simulation model for bridge maintenance are:

1. Bridges cannot be viewed as a single entity but need to be divided into their sub-elements. The CoRe classification as mentioned earlier consists of about 160 elements, each of which can exist in 3-5 states with each state having its own deterioration rates, maintenance activities and costs. All this data needs to be

suitably represented and managed in the simulation model.

2. The different permutations of policies that can be performed on each individual element give rise to a physically unmanageable number of policies at the network level. A suitable mechanism needs to be developed so that the user can effectively manage the numerous possible network policies.

4.1.2 Policy Analysis

At any given point of time each bridge has its own needs and competes for resources. Though each bridge is unique, a common platform needs to be developed in order to compare the competing bridges and ensure maximum benefit for the resources consumed.

4.1.2.a *Health Index:*

The need to have a single number that could be used to judge the performance of maintenance and rehabilitation efforts was identified by California Department of Transportation. They have developed an improved performance measure called the health index (HI), which ranks a bridge on a scale of 0 to 100 (Roberts and Shepard, 1999). The HI uses element inspection data based on the FHWA CoRe element description to determine the remaining asset value of a bridge. It weighs the quantity of each element in each condition state against the failure cost of the element. A summation of the value of all elements on a bridge is then compared with the summation of the reduced value of any deficient elements; the ratio of the two numbers is the HI for that bridge.

Thus $HI = (SCEV/STEV)*100$ (1)

Where TEV is the total element value and CEV is the current element value.

TEV=Total element quantity*failure cost of the element (FC)

$CEV=\Sigma(\text{Quantity Condition State}_i*WF_i)*FC$

The method of calculation of HI along with an example is given in Appendix B.

4.1.2.b *Improvement Cost Ratio*

The HI provides an effective mechanism to compare different bridges and rank them accordingly. However merely selecting the bridges with the lowest HI for maintenance is not an effective strategy since this merely translates to reconstruction and rehabilitation as opposed to a combination of preventive maintenance and rehabilitation and might not result in effective spending of the available funds.

In order to ensure effective spending a mechanism called the Improvement Cost ratio has been developed by the author. This concept is based on benefit cost ratio. The Improvement Cost ratio is defined as

$$Improvement\ Cost\ Ratio = \frac{Future\ HealthIndex - Current\ HealthIndex}{Cost\ of\ Maint.Action} \qquad (2)$$

where the Future Health Index is the new health index the bridge will achieve if the proposed maintenance action is performed, Current Health Index is the HI of the bridge before the maintenance action is performed and Cost of Maint. Action is the total cost of the proposed maintenance action.

Based on the element level policies the overall Improvement Cost ratio (IC) for the bridges is developed every year. The bridges are then ranked in descending order of their IC ratio. Based on the available funds the bridges are selected from the top until the model runs out of funds thereby ensuring maximum improvement for the money spent. If the model encounters a situation where the available funds are insufficient for the bridge under consideration in the list, the model proceeds down the list to see if it can perform maintenance activity on a bridge with lower IC ratio.

4.1.3 Classification of Bridges

For the purpose of reporting, the average HI is not a suitable mechanism, as it does not give the entire picture of the network. Consider an example with 5 bridges. The average HI can be the same if all the bridges have a HI of 80 or if the HI's of the bridges are say 95, 95, 95, 95 and 20. However the two cases are not the same. In the second case

the last bridge is well below the level of service and probably close to failing, which is not reflected in the average value. What is needed is a reporting mechanism that can show the distribution of bridges and give a clearer picture of the state of the network. For this purpose the bridges have been classified into five different categories – Excellent, Good, Fair, Poor, and VPoor. The specific qualifying criterion for each of these categories is given in Table 4.3.

TABLE 4.3 – Classification of Bridges

Category	**Qualifying Criterion**
Excellent	HI>95
Good	HI<=95 and HI>82
Fair	HI<=82 and HI>70
Poor	HI<=70 and HI>40
VPoor	HI<=40

Every year the bridges can now be classified into these five categories giving a clearer picture of the state of the network. Using this classification in the above example yields 5 bridges in Good condition in the first case and 4 in excellent and 1 in VPoor condition state in the second case.

There might be a target distribution of Bridges among these categories (e.g. 25% in Excellent, 65% in Good and 10% in Fair). However the actual distributions might be very different from this. The simulation model can be run with different budgets and policies in order to compare the actual conditions versus the target conditions. Since the region of deficit can be observed clearly, it can be focused in the next run. For example if it is observed that the actual distribution is 30% Excellent, 35% Good and 35% Fair, then most of the budget should be allocated to the Bridges in Fair condition to restore them to Good and probably a portion to Excellent to maintain them in that state.

4.2 CONCEPTUAL MODEL

Based on this understanding of the working of bridge management and the

available data a conceptual model was developed to represent the bridge management process.

4.2.1 Software Development Tools

Microsoft Windows operating system was used as the development platform for the software development part of the research. The software was developed such that it can be implemented on any Windows platform (95/98/NT/2000).

Next is the selection of suitable discrete event simulation software. Stroboscope simulation language was used for Simulation modeling. Its powerful modeling capabilities made the representation of complex models very easy. Further Stroboscope provides a front end, which enables the overall structure to be viewed visually similar to a flowchart thereby reducing the possibility of any logical errors. The conceptual model is developed in Microsoft VISIO using Stroboscope templates.

Some extensions had to be made to Stroboscope for it to interface with PONTIS data. These extensions were developed in C++ using Microsoft Visual Studio – version 6.

4.2.2 Stroboscope

Stroboscope is a discrete event simulation language with a Microsoft VISIO interface to develop models. What follows is a brief description of the key concepts used in Stroboscope. This discussion should help the reader follow the implementation of the model in Stroboscope.

Almost all models in Stroboscope are represented using networks. A network is a high level representation of a simulation model and consists of nodes connected by links through which resources of different types flow. Resources are the basic things used to represent real life entities like *Trucks, Loaders, Bridges, Pre-Stressed beams* etc. Resources however need not be real-life objects and can be abstract 'things' also.

There are different kinds of modeling elements in Stroboscope. The first element is the queue element represented graphically with a circle. The queue entity is a container

for holding a resource or resources that are idle until certain criteria are met. Each queue is associated with a particular type of resource. The next entity is the activity which as opposed to queues draws resources as soon as they are available and does something. They represent the work or tasks to be performed using the necessary resources. An activity can represent a task that can take place zero, one or several times during simulation. Each of these instances can be independent of one another and can occur simultaneously or can occur one after another. Each instance can have its own duration and can hold specific resources that were required for that activity to start.

The most common activities used in Stroboscope are the Normal and Combi activities. Combi activities represent tasks that start when certain conditions are met. At appropriate moments in the simulation, Combi's are scanned to determine if the necessary conditions exist for them to start. Combi's can only draw inactive resources i.e. from queues. They are represented as rectangles with their top-left corner chopped off. Normal activities represent tasks that start immediately after other tasks end. A Normal acquires resources from the activity that has just finished. They are represented as rectangular boxes in the model.

These three elements are the basic building blocks of a Stroboscope simulation model are can be used to represent the basic models. A more complex model can have more complex building blocks like assemblers, disassemblers, consolidators etc. For a more detailed discussion of the elements of Stroboscope and understanding Stroboscope models, refer Martinez (1996).

4.2.3. Working of Simulation Model

Based on the understanding of the bridge management process and the available data the overall working of the simulation model is developed to emulate the bridge management process. Figure 4.1 shows the flow chart working of the simulation model. This model is described below. The bridges are created and initialized and then placed in the bridge network where they reside for a major portion of the simulation. During each year the qualifying bridges are drawn through the bridge network based on the selection

criteria (Improvement Cost ratio and available funds). The qualifying bridge is sent to the *RepairBridges* activity where the preferred maintenance activities are performed on each element and the repaired bridge is sent back to the Bridge Network where it resides until the next year.

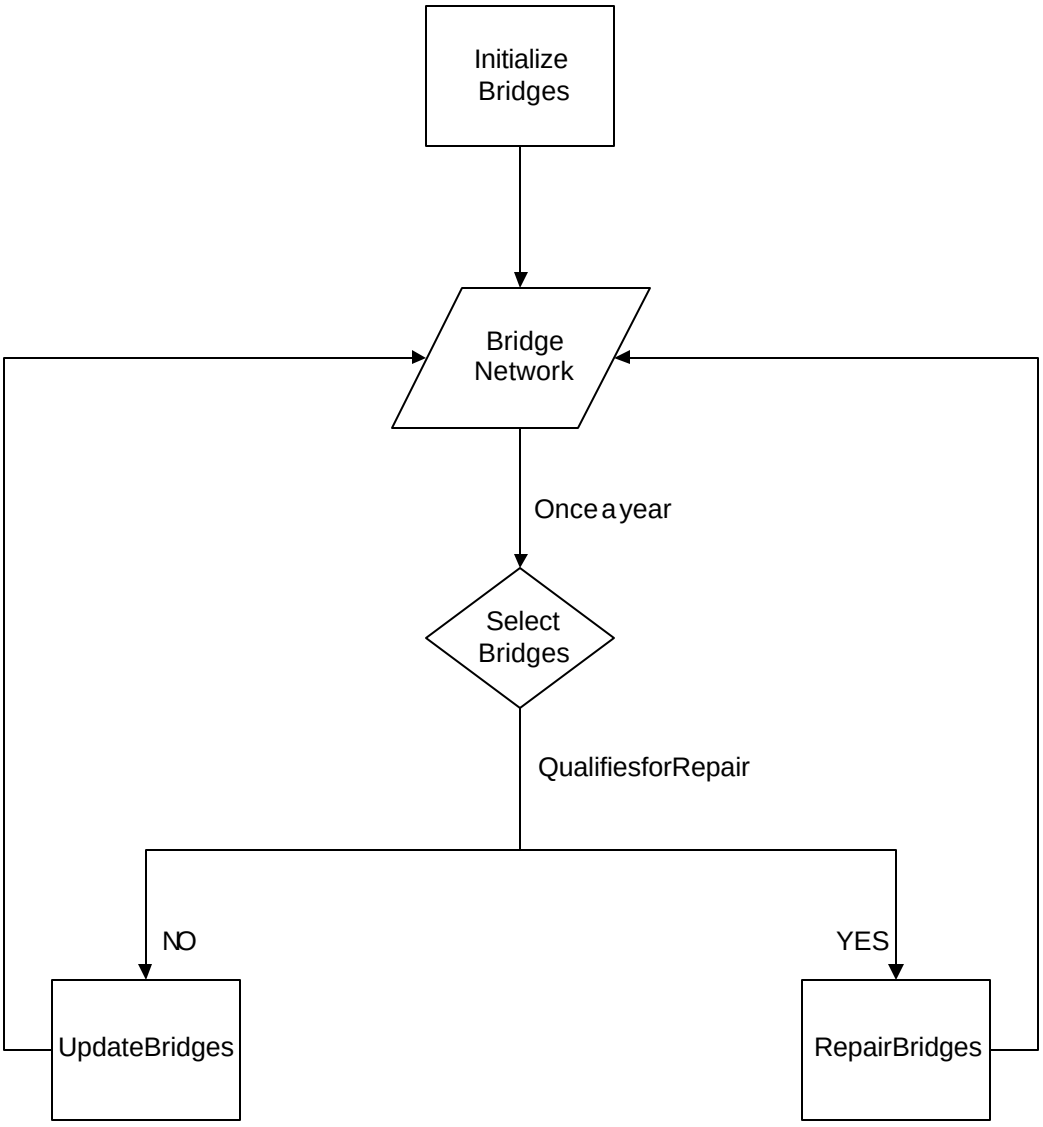

FIGURE 4.1 Flow Chart of the Bridge Maintenance Process

At the end of the year, all the bridges that did not qualify for repair are sent to the *UpdateBridges* activity, which updates the conditions of the bridge elements and sends the bridge back to the bridge network. Note that in addition to the Improvement Cost ratio and the available funds, there are two additional criteria which determine if a bridge qualifies for repair or not. The first is the delay, which ensures that if a bridge is repaired then it is not repaired for the next three years. This condition is imposed to ensure that the same bridge does not qualify for repair over successive years, thereby depriving other bridges of the needed funds. This is actually not far from reality. The second criterion introduced is that at any instant of time only a certain number of bridges can be repaired simultaneously. This condition is imposed to capture the resource constraint in the network. Working on all the bridges simultaneously brings the transportation infrastructure to a halt. Any small area considered in the simulation will have a small number of competent contractors who can undertake the repair work thus limiting the number of simultaneous repairs. Further the decision makers might not wish to close more than a specified number of bridges for repair to avoid inconvenience to the general public. The effect of all these real life constraints is that at any point of time only a limited number of bridges should be closed for repair, which is what is being modeled. It should be realized that these are strategies in the model and their parameters can be changed to try out different strategies. For example to try out the hypothetical situation where unlimited number of contractors are available, the number of bridges which can be worked on can be set to a very high number. Similarly the impact of changing the size of the infrastructure maintenance staff can be indirectly measured by varying this number.

The front end of the simulation model is shown in Figure 4.2. What follows is a brief description of the working and the underlying logic of the model in Stroboscope.

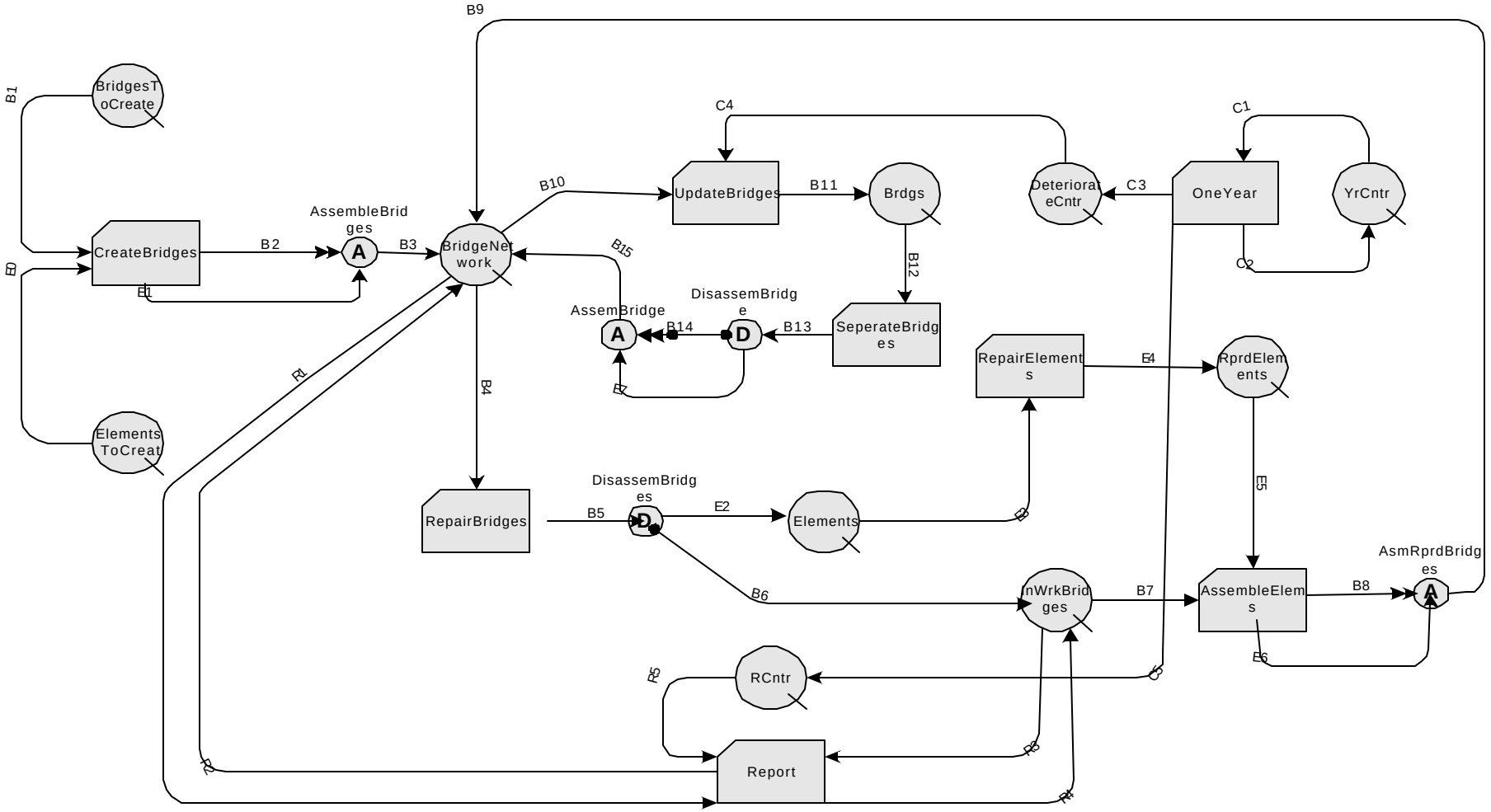

FIGURE 4.2: Visualization of the Stroboscope Model

The *BridgesToCreate* node is where the bridges in the network are created. Here the bridges are a simple place holder for the elements and do not contain any information other than Bridge ID. The *ElementsToCreat* is where the individual elements of the bridge are created. The elements contain the inspection data and deterioration conditions. *CreateBridges* node matches the bridges to the corresponding elements and creates whole bridges. At this point in the simulation the bridge properties like number of elements, overall health index (HI) of the bridge are determined. The bridges and their corresponding elements are combined together in the *AssembleBridges* node as a unit. At this stage the Bridge component contains the individual element components and the two are inseparable. Once assembled the individual elements cannot be accessed directly. Only the Bridge properties are available at this stage. This feature is used to group together all the elements of a bridge together and work on them as a bridge unit instead of the individual elements. It also helps to compare bridges instead of individual elements of a bridge. At this stage the bridges pass on to the *BridgeNetwork* node. This concludes the initialization phase of the model and at the end of this stage bridge network with all the relevant information has been created.

RepairBridges node draws the bridges from the network for repair and maintenance one by one. The total number of bridges that can be drawn for repair depends on the available budget, which can be changed every year. The link B4 connecting *BridgeNetwork* and *RepairBridges* is the link where decisions pertaining to which bridge to work on are taken. This is a crucial part of the simulation model since the network policies are what decide the behavior of the model. Once a bridge is selected it passes on to the *DisassemBridges* node, which separates the elements from the bridges so that they can be worked on. The *RepairElements* is where the elements are repaired and their condition, cost incurred and other parameters are updated. Also the available funds are deducted by the appropriate amount. Once all the elements of the bridge are repaired, the bridge can be reassembled. *AssembleElems* matches the elements to the corresponding bridges and they are put back together in *AsmRprdBridges* and sent back to the *BridgeNetwork*.

If a bridge is not repaired then it deteriorates and its status needs to be updated. This updating of the bridges takes place at the end of every year. The loop *UpdateBridges*, *Brdgs*, *SeperateBridges*, *DisassemBridge*, and *AssemBridge* performs this action. The names of the nodes indicate the roles they play. *UpdateBridges* selects the bridges whose condition needs to be updated. *Brdgs* and *SeperateBridges* separate the bridges and send them one by one to *DisassemBridge*. The link *E7* connecting *DisassemBridge* and *AsssemBridge* updates the conditions of the elements. *AsssemBridge* updates the bridge parameters (HI, RepairCost etc.) based on the new element conditions. *DisassemBridge* and *AssemBridge* update the deterioration of the node. The *YrCntr* and *OneYear* loop assures that the bridges are updated once every year. Finally *Report* draws the bridges every year and reports their status to the user.

As the proposed model was being developed, experts in the field were consulted to test the accuracy of the model. This iterative process was continued until an accurate representation of the bridge management process was obtained.

4.3 STROBOSCOPE IMPLEMENTATION OF THE MODEL

Based on the above behavior of the model the corresponding Stroboscope code is created and the appropriate Stroboscope statements are written. The complete Stroboscope program is given in APPENDIX D. What follows is a description of the parameters that can be varied to change the behavior of the model.

4.3.1 Simulation Parameters

The parameter YrsToSimulate defined in line 7 is the number of years the model is to be simulated. The array FUNDSMTRX (line 9) contains the funds assigned every year. The next four parameters in lines 12 through 15 give the lower bounds of the corresponding ranges. Only four parameters are required as the upper limit of Excellent is assumed to be 100 and the lower bound of VPoor is assumed to be 0. The parameter Delay in line 17 gives the minimum time interval between successive maintenance works on any bridge. Finally the parameter MaxBridgeCount on line 17a gives the maximum number of bridges that can be closed for repair simultaneously.

4.3.2 Additional Components of the Policy Analysis Tool

The Stroboscope model described above can simulate the network and project the network health over a period of time. However the model cannot work in isolation. A suitable mechanism needs to be developed to incorporate the existing network conditions as a starting point for the model. Since the goal of this research is to employ Discrete Event Simulation as a policy analysis tool and not to develop a Bridge Management System, it was decided to develop the policy analysis tool to work in conjunction with a Bridge Management System. PONTIS bridge management system was selected due to its availability and ease of use. The input mechanism that should be developed will import all the appropriate data from PONTIS. Further an appropriate querying tool should be devised so that the user can try out different network policies. All these components should work in tandem for an effective policy analysis tool. Figure 4.4 indicates the different components required by the policy analysis tool.

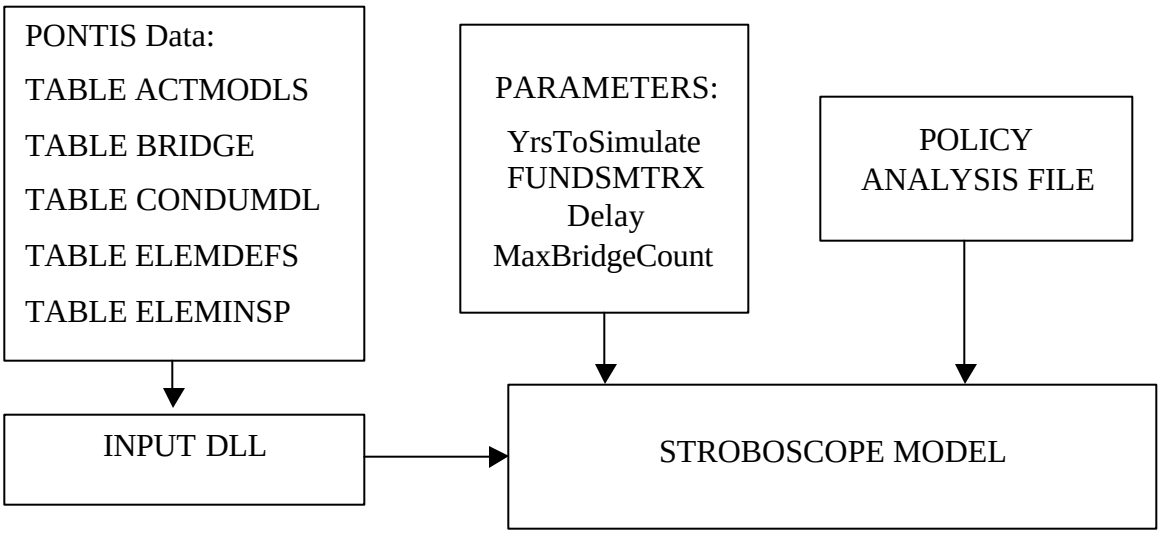

FIGURE 4.3 Components of the Policy Analysis Tool

4.3.3 Working of the Stroboscope Code

Stroboscope cannot extract data from text files. Therefore an add-on DLL was developed which aids Stroboscope to extract the data from the appropriate PONTIS files. The list of all the tables which are extracted from PONTIS are given in figure 4.3. A detailed description of all the tables imported from PONTIS follows in section 4.4. The

command LOADADDON in line 6 loads the appropriate DLL. Once the DLL is loaded the information read from the PDI files is extracted by calling the appropriate functions provided by the DLL. First the data files need to be opened using the OpenInputFile[ ] function. This function takes the name of the file as the parameter and returns a Stroboscope address to the file. Unless the file is in the same directory as the Stroboscope model, the full path of the file should be specified as the parameter. Then the appropriate functions are called in lines 34,36,40,42 and 45 using the Stroboscope file address as the parameter as given below.

```
34 CALL ReadElementData[data];
35 ASSIGN data OpenInputFile["Bridges.txt"];
36 CALL GetBridgeInfo[data];
37 ASSIGN data OpenInputFile["ElemInsp.txt"];
38 CALL GetElementInfo[data];
39 ASSIGN data OpenInputFile["int_elemdefs.PDI"];
40 CALL GetStateCount[data];
41 ASSIGN data OpenInputFile["int_condumdl.PDI"];
42 CALL GetFailureCost[data];
43 /Read the element level policies to be implemented
44 ASSIGN data OpenInputFile["PolicyAnalysis.txt"];
45 CALL GetPolicyInfo[data];
```

The names of the functions indicate the specific role played by them. Lines 19 to 29 define the various data containers (arrays) needed to effectively manage the data and are given below.

```
18. /DECLARATION FOR TRANSITION PROBABILITIES
19. ARRAY TransProbData 800;
20. /DECLARATION FOR MR&R ACTIONS AND TRANSITION
PROBABILITIES FOR ACTION TYPE 1
21. ARRAY MaintAct1Data 800;
22. /DECLARATION FOR MR&R ACTIONS AND TRANSITION
PROBABILITIES FOR ACTION TYPE 2
23. ARRAY MaintAct2Data 800;
24. /The preferred element level activity matrix
25. ARRAY PrefMaintAct 800 5;
26. /Failure cost information for every element
27. ARRAY FailrCostData 800;
28. /Array giving the maximum number of allowable states
for each element
29. ARRAY StateCnt 800;
```

The specific information contained in each array is given in the comments

associated with the array definitions. In particular the TransProbData, MaintAct1Data, and MaintAct2Data arrays are 3-dimensional arrays, even though the definitions indicate that they are 1-dimensional arrays of size 800. Each of the 800 values is in turn a 2-dimensional matrix. These matrices are defined in line 454, 469, and 483 of the Input DLL and are filled with the appropriate values read from the PONTIS PDI files. Each element of the TransProbData is a 2-D array of size 5X5 where each element indicates the transition probability from a condition state (the row) to a transition state (the column). The arrays MaintAct1Data and MaintAct2Data however are of size 6X5. The 6^{th} element in each row corresponds to the unit cost for that maintenance activity. The row gives the initial condition state of the element and the column gives the target condition state.

These tables predict the future condition of the element after a year given its current condition. For example consider figure 4.4. It shows the transition probability table for element 131. It can be observed from the figure that a painted steel deck truss (element 131) in condition state 3 has a probability of 0.7 of staying in the same condition state in a span of one year, a probability of 0.2 of deteriorating to condition state 4 and 0.1 of reaching condition state 5. In other words, if 100 square feet of the deck truss is in condition state 3, after a year approximately 70 square feet will remain in state 3, 20 square feet will reach state 4 and 10 square feet will be close to failure (state 5).

CONDITION STATE FROM	CONDITION STATE TO 1	2	3	4	5
1	0.1	0.9	0.0	0.0	0.0
2	0.0	0.8	0.2	0.0	0.0
3	0.0	0.0	0.7	0.2	0.1
4	0.0	0.0	0.0	0.7	0.3
5	0.0	0.0	0.0	0.0	1.0

FIGURE 4.4 Transition Probability Data for Element 131

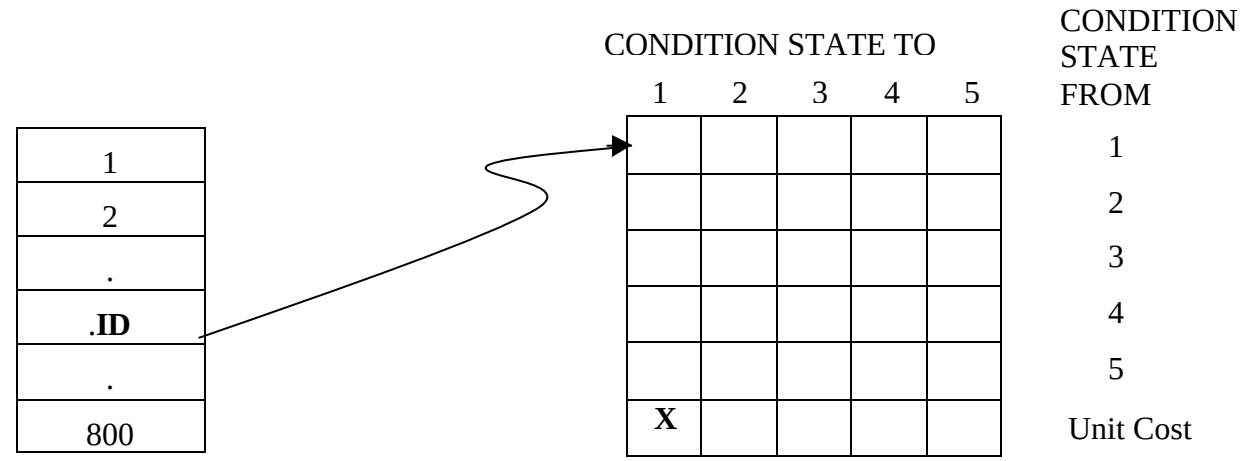

FIGURE 4.5 Organization Of Data Into 3-D Arrays

The organization of the data is shown in figure 4.5. Since only 2-D array elements can be accessed using the regular Stroboscope syntax, the elements in these 3-D arrays should be accessed using a function GetArrayElement[ ]. Thus in the above figure if there are 800 2-D arrays for each table, GetArrayElement will point to the appropriate 5X5 table for TransProbData and the appropriate 6X5 array for MaintAct1Data and MaintAct2Data. For example consider GetArrayElement [MaintAct2Data [ID],0,5] from line 149 of the Stroboscope model. The MaintAct2Data specifies the array from which the data needs to be extracted. The ID indicates the element number whose data needs to be extracted. ID varies between 0 and 800. The next two parameters indicate the specific value that needs to be extracted. For example the 0 and 5 here indicate that the number corresponds to the unit cost for MaintAct2Data in condition state 0 as indicated by the 'X' in the figure. Similarly GetArrayElement[TransProbData[ID],2,3] gives the transition probability of element ID from condition state 2 to condition state 3 for no maintenance (Since TransProbData array contains the no maintenance transition probabilities).

The resource types that are used in the simulation are defined in lines 48 to 69. The bridge and the element are the basic resource types of the model. In addition the counter resource type is a generic resource type defined to help in counting and managing the model. The properties associated with each resource type are also defined here. The

CEV and TEV of the element correspond to the Current Element Value and the Total Element Value of the element.

```
48. COMPTYPE   Bridge; /BR
49. SAVEPROPS  Bridge HI TotElems RepairCost BrID
QueueFrm LastRepairedTm FHI;
50. GENTYPE         Counter; /CO
51. COMPTYPE   Element; /EL
52. SAVEPROPS  Element State1 State2 State3 State4
State5 Qty ID EHI Source UnitFlrCst statecnt S1 S2 S3
S4 S5 Cost1 Cost2 Cost3 Cost4 Cost5 FState1 FState2
FState3 FState4 FState5;
53. /Variable storing the current element value
54. VARPROP    Element CEV 'statecnt==5?
55.
(State1+State2*0.75+State3*0.5+State4*0.25)*Qty*UnitFl
rCst:
56.
statecnt==4?(State1+State2*0.67+State3*0.33)*Qty*UnitF
lrCst:
57. (State1+State2*0.50)*Qty*UnitFlrCst';
58. /Variable storing the future element value
59. VARPROP Element FEV 'statecnt==5?
60. (FState1+FState2*0.75+FState3*0.5+FState4*0.25)
*Qty*UnitFlrCst:
61. statecnt==4?(FState1+FState2*0.67+FState3*0.33)
*Qty*UnitFlrCst:
62. (FState1+FState2*0.50)*Qty*UnitFlrCst';
63. /Variable storing the total repair cost of the
element
64. VARPROP Element RepairCst '
65.
(State1*Cost1+State2*Cost2+State3*Cost3+State4*Cost4+
State5*Cost5)*Qty';
66. /Variable storing the total element value
67. VARPROP Element TEV 'Qty*UnitFlrCst';
68. /Variable storing the IC ratio for the bridge
69. VARPROP Bridge IC 'RepairCost!=0?( (FHI-
HI)*1000/RepairCost ):-100';
```

The Current Element value is defined by the California Department of Transportation as $CEV=\Sigma(\text{Quantity Condition State}_i*WF_i)*FC$. Appendix B describes the calculation of the CEV in detail. This calculation of CEV corresponds to the lines 54 to 57 in the model. Similarly the formula for calculating the FEV is given in lines 59 to 62 and is given below.

```
54. VARPROP    Element CEV 'statecnt==5?
55.(State1+State2*0.75+State3*0.5+State4*0.25)*
Qty*UnitFlrCst:
56.statecnt==4?(State1+State2*0.67+State3*0.33)*Qty*Un
itFlrCst:
57. (State1+State2*0.50)*Qty*UnitFlrCst';
59. VARPROP Element FEV 'statecnt==5?
60. (FState1+FState2*0.75+FState3*0.5+FState4*0.25) *
Qty*UnitFlrCst:
61.statecnt==4?(FState1+FState2*0.67+FState3*0.33)*
Qty*UnitFlrCst:
62. (FState1+FState2*0.50)*Qty*UnitFlrCst';
```

The FEV corresponds to the Future Element Value of the bridge. The FEV is the element value of the bridge in the future provided the recommended maintenance actions are performed. The total repair cost given in lines 64 and 65 is the summation of the repair costs in all condition states i.e. state1*cost1+state2*cost2+…+state5*cost5. The IC calculates the Improvement Cost of the bridge and is given by the improvement in the HI for the money spent. This formula is incorporated in line 69. Note that if the repair cost of the bridge is 0, then the IC is assigned a value of –100 to ensure that the bridge does not qualify for repair.

Lines 72 to 132 describe the different nodes used in the Stroboscope model. These nodes correspond to the combis, queues, and links given in figure 5.4. The bridges and elements are created at the start of the simulation in lines 818 and 820. The current element information from the PONTIS database is associated with the elements in lines 772 to 780 and is given below.

```
772. ONENTRY ElementsToCreat ASSIGN ID
El__INFO[ResNum-1,0];
773. ONENTRY ElementsToCreat ASSIGN Qty
El__INFO[ResNum-1,2]+El__INFO[ResNum-
1,4]+El__INFO[ResNum-1,6]+El__INFO[ResNum-
1,8]+El__INFO[ResNum-1,10];
774. ONENTRY ElementsToCreat ASSIGN State1
El__INFO[ResNum-1,1]/100;
775. ONENTRY ElementsToCreat ASSIGN State2
El__INFO[ResNum-1,3]/100;
776. ONENTRY ElementsToCreat ASSIGN State3
El__INFO[ResNum-1,5]/100;
777. ONENTRY ElementsToCreat ASSIGN State4
```

```
El__INFO[ResNum-1,7]/100;
778. ONENTRY ElementsToCreat ASSIGN State5
El__INFO[ResNum-1,9]/100;
779. ONENTRY ElementsToCreat ASSIGN Source
El__INFO[ResNum-1,11];
780. ONENTRY ElementsToCreat ASSIGN statecnt
StateCnt[ID];
```

The Input DLL fills the EL__INFO array with the appropriate element info. Similarly the DLL fills Br__INFO array with the bridge information from the PONTIS data files. This bridge information is associated with the bridges upon creation in line 327. Thus the DLL reads the number of bridges and stores it in Br_INFO and the number of elements and stores it in El_INFO so that the stroboscope model can create the appropriate number of bridges and elements as there are in the bridge network.

The statements in lines 140 to 145 are responsible for matching the bridges and their corresponding elements upon creation.

```
140. /* Startup of CreateBridges
141. PRIORITY       CreateBridges '500';
142. DRAWUNTIL      B1 'CreateBridges.Bridge.Count';
143. DRAWUNTIL      E0
'ElementsToCreat.ElemsofBridge.Count==0';
144. DRAWWHERE      E0
'Source==CreateBridges.Bridge.BrID';
```

They correspond to the links E0, B1 and the combi CreateBridges in the network. The bridges are selected one at a time and sent to CreateBridges. The filter ElemsofBridge created in line 136 matches the elements corresponding to that bridge. Line 143 ensures that all the elements of a bridge are drawn to match them to their corresponding bridge. The condition in line 144 ensures that only the elements belonging to the bridge in question are selected. The bridges and their corresponding elements are combined together in the AssembleBridge node. At this point the properties of the bridge are updated (lines 316 – 324) and the bridges are sent to the BridgeNetwork.

When drawing the elements through the link E0 their properties are set. The code corresponding to this operation is in lines 149 – 310. The unit cost information of the elements is set in lines 149 – 153 based on the preferred maintenance activities defined in

the policy analysis table. For example consider line 149 given below.

```
149. ONDRAW E0 ASSIGN Cost1 ' (PrefMaintAct[ID,0]==2)?
GetArrayElement[MaintAct2Data[ID],0,5]:
((PrefMaintAct[ID,0]==1)?
GetArrayElement[MaintAct1Data[ID],0,5]:0)';
```

The maintenance cost for element in state1 depends on the recommended maintenance action for state1. If maintenance action1 is recommended, then Cost1 = Unit cost of maintenance action1 (Given by GetArrayElement[MaintAct1Data[ID],0,5]), otherwise Cost1 = Unit cost of maintenance action2 (Given by GetArrayElement[MaintAct2Data[ID],0,5]). Similarly the other costs are defined in the appropriate manner.

The transition probability information is then copied to a temporary array ActProbData (155-304). For example observe lines 155 to 160

```
155. ONDRAW E0 ASSIGN ActProbData 0 0 '
156. PrefMaintAct[ID,0]==2?
157. (GetArrayElement[MaintAct2Data[ID],0,0]):
158. PrefMaintAct[ID,0]==1?
159. (GetArrayElement[MaintAct1Data[ID],0,0]):
160. (GetArrayElement[TransProbData[ID],0,0])';
```

The model checks whether maintenance action 1 or maintenance action 2 is recommended for state1. Based on this information, it extracts the appropriate transition probability (GetArrayElement[MaintAct2Data[ID],0,0] or GetArrayElement [MaintAct1Data[ID]0,0]). Using this future condition states are set in lines 306 - 310. The future state is merely the transition probability multiplied by the amount of quantity. For example. consider line 306 shown below.

```
306. ONDRAW E0 ASSIGN FState1
'State1*ActProbData[0,0]+
State2*ActProbData[1,0]+ State3*ActProbData[2,0]+
State4*ActProbData[3,0]+ State5*ActProbData[4,0]';
```

The total quantity in Future state1 is given by the current quantity in a state* probability that it will go to state1 when the recommended maintenance actions are performed i.e. ?$State_i$*Probability that that state goes to state1.

The selection of bridges should include the minimum delay period before a bridge

can qualify for repair. This is determined in the NonDelayedHI parameter defined in line 328, which ensures that a bridge qualifies for repair only if SimTime-LastRepairedTm >= Delay (The acceptable value is to be assigned by the user.) The NonDelayedHI parameter also takes into account if the funds are sufficient to undertake repair on a bridge through the condition RepairCost <= Funds. Thus if the bridge should not qualify for repair based on either the delay or lack of funds, the NonDelayedHI is assigned a very low number to prevent the selection of this bridge and if the bridge qualifies then the IC is assigned the NonDelayedHI (Note that all these are defined as VarProps which ensures that their values are computed instantaneously thereby ensuring that the current conditions are taken into account). Lines 758 to 766 contain the ordering mechanism for the bridges.

```
758. /* Startup of RepairBridges
759. FILTER BestExc Bridge 'NonDelayedHI ==
BridgeNetwork.NonDelayedHI.MaxVal & NonDelayedHI!=-
100';
760. SEMAPHORE RepairBridges ' (BridgeNetwork.CurCount
+ InWrkBridges.CurCount == Br__Count) ';
761. PRIORITY       RepairBridges 20;
762. ENOUGH B4 'BridgeNetwork.BestExc.Count &
BridgeNetwork.BestExc.RepairCost.AveVal<=Funds';
763. DRAWWHERE B4
'NonDelayedHI==BridgeNetwork.NonDelayedHI.MaxVal &
NonDelayedHI!=-100 & RepairCost<=Funds';
764. DRAWUNTIL B4 RepairBridges.Bridge.Count;
765. DRAWORDER B4 -NonDelayedHI;
766. ONDRAW         B4 ASSIGN Funds Funds-RepairCost;
```

These lines rank the bridges in increasing order of NonDelayedHI (effectively the IC) and select the bridges with the highest value until the model runs out of funds. The filter in line 759 determines the bridge with the highest value. This bridge is selected in line 763. Line 764 ensures that only one bridge is selected at a time. The bridges are then repaired and sent back to the bridge network. Line 765 deducts the cost of the repair work of the selected bridge from the total available funds and proceeds to the next bridge.

The node DisassemBridges then segregates the bridge from the elements in order to update the condition of the elements. The RepairElements repairs the elements for the duration of the maintenance activity and then sends the updated elements to the RprdElements queue. While flowing through the link E4, the condition states of the

elements are updated (Lines 346 - 513). Similar to the link E0, the current states and the future states of the element are updated. For the purpose of the updating the element and performing the repair work, the preferred maintenance activities are determined from the PrefMaintAct array.

Once all the elements corresponding to a bridge are repaired, the bridge is ready to be combined with its elements and sent back to the network. Lines 521 to 531 check if all the elements corresponding to the bridge have been repaired and then initiate the next step. They are also responsible for the matching of the elements to the corresponding bridges similar to the lines 140 - 145 for links E0 and B1. The bridge and elements are then assembled in the AsmRprdBridges where the properties of the bridge are updated (Lines 537 – 546) and the bridge is sent back to the bridge network. It should be noted that the LastRepairedTime parameter of the bridge is also reset here to the current simulation time in order to ensure that the bridge does not qualify for the specified number of years (Line 538).

```
538. ONASSEMBLY AsmRprdBridges ASSIGN LastRepairedTm
SimTime;
539. ONASSEMBLY AsmRprdBridges ASSIGN RepairCost '
540. AsmRprdBridges.Element.RepairCst.SumVal';
541. ONASSEMBLY AsmRprdBridges ASSIGN HI '
542. AsmRprdBridges.Element.TEV.SumVal!=0?
(AsmRprdBridges.Element.CEV.SumVal/
543. AsmRprdBridges.Element.TEV.SumVal*100):0';
544. ONASSEMBLY AsmRprdBridges ASSIGN FHI '
545. AsmRprdBridges.Element.TEV.SumVal!=0?
(AsmRprdBridges.Element.FEV.SumVal/
546. AsmRprdBridges.Element.TEV.SumVal*100):0';
```

If the bridge does not qualify for repair, then it is selected by the UpdateBridges node at the end of the year. This selection is coded in lines 561 to 563.

```
561. ENOUGH B10          1;
562. DRAWWHERE      B10 'SimTime-LastRepairedTm>=1';
563. DRAWUNTIL      B10 0;
```

The link B10 only selects the bridges that were not repaired in the current year and need to be updated. Once the bridges are selected, they need to be sent one by one to the disassembler, which is ensured by the queue Brdgs and combi SeperateBridges. The

DisassemBridge node separates the bridges from the elements so that they can be updated. The link E7 updates the condition of the elements (Lines 576 – 743). The calculations are similar to those performed in link E7, except they are simpler since only the transition probabilities from the TransProbData need to be used. Once the element conditions are updated, the bridge properties are updated in the assembler AssemBridge (line 746 – 754) after which the updated bridges are sent back to the bridge network.

For the purpose of reporting the bridge network conditions, the Report activity draws the bridges every year. In order to ensure that the bridges do not enter the repair or deterioration cycles before reporting, the Report activity is given a higher priority so that it takes precedence over the other activities (Line 795). The bridges could be in either the bridge network or the InWrkBridges queue and so the report draws the bridges from both the links. In order to ensure that the bridges are returned to the appropriate queue on termination of Report, each bridge has a parameter Queue which is set to the appropriate queue on entering that queue (lines 327, 338). Upon termination of the Report activity the bridges are sent back to the appropriate queue using this parameter (lines 809, 810).

```
809. RELEASEWHERE   R2 'QueueFrm==BridgeNetwork';
810. RELEASEWHERE   R4 'QueueFrm==InWrkBridges';
```

The conditions of the bridges are collected in the BinCollector HICollector which sorts the HI values into appropriate bins from 0-100 (lines 784, 788, 790). This information is then sent to the output to display the network health (lines 797 – 804).

```
797. ONSTART Report PRINT StdOutput "%4.0f
%15.2f%15.0f%12.0f%12.0f%12.0f%12.0f\t"SimTime
798. HICollector.AveVal
799. HICollector.nSamples-
HitsAtOrBelowBin[HICollector,Excellent]
800. HitsAtOrBelowBin[HICollector,Excellent]-
HitsAtOrBelowBin[HICollector,Good]
801. HitsAtOrBelowBin[HICollector,Good]-
HitsAtOrBelowBin[HICollector,Fair]
802. HitsAtOrBelowBin[HICollector,Fair]-
HitsAtOrBelowBin[HICollector,Poor]
803. HitsAtOrBelowBin[HICollector,Poor];
```

Upon termination of the report activity the funds for the new year are allocated to start the repair and updating cycle one again (line 812).

4.4 SEGREGATION OF DATA AND MODEL

It can be observed that the model does not depend on the number of bridges in the network or the condition of the bridges, or the transition probabilities or any other network characteristics. These parameters are specific to each bridge network and if they are incorporated into the model, this renders the model problem-specific, which is contrary to the goal of this project. Therefore these parameters should be separated from the model. To facilitate this, the available data from the Salem District of Virginia was studied to develop a suitable mechanism to input the data into the model. Another advantage of this date segregation is that any changes made to the model will not affect the data and vice versa.

After careful study of the Salem district databases, it was decided to input the data into the model using text based data files. The format of these files is set up to coincide with the PONTIS Data Interface (PDI) files. The PDI files can be generated from PONTIS by selecting the appropriate tables using the import mechanism. These files drive the simulation model without any changes thus simplifying the process of data input to the model significantly. Since all the inspections are entered into the PONTIS databases regularly, it ensures that the simulation model works with the latest information.

4.5 DATA INPUT FROM PONTIS: (C++ DLL)

The PDI information should be retrieved by the Stroboscope model. However Stroboscope was not equipped with the provision to extract data from an input file. So a DLL was developed in C++, which facilitated reading data from a text file. Then a set of functions were written which automated the process of reading the data from the text files and interpreting it accordingly and thereby driving the simulation model. The following tables have been imported from PONTIS in the Pontis Data Interchange (PDI) format

1. TABLE ACTMODLS
2. TABLE BRIDGE
3. TABLE CONDUMDL

4. TABLE ELEMDEFS
5. TABLE ELEMINSP

The complete list of all the tables imported from PONTIS along with the fields and their purpose in the model is given in APPENDIX C. The data is extracted by calling appropriate functions provided by the DLL. Once the information is extracted, the model proceeds with the simulation.

The complete C++ source code for the Input DLL is given in APPENDIX E. The comments in the code describe the working of each section of the code. In addition the development of DLL's for Stroboscope is documented in Martinez (1996). The specific functions supported by the DLL along with their descriptions are given in APPENDIX G.

4.5.1 Assumptions made in the Model Development

1. All maintenance actions have duration of 15 days. Though data corresponding to the duration of maintenance actions is not available, it is assumed that with the growing importance of bridge management state agencies will soon start collecting this information and it will be available. Once the information is available it can be incorporated into the model. The model has the provision to incorporate this information when available.

2. All the elements are assumed to be in a moderate environment (envID=2). Most state agencies have not yet classified their bridges into the different environment categories and so it was felt that modeling the separate environments would be a futile exercise. Therefore the default environment for all bridges is assumed to be moderate. If this assumption is felt inadequate, the environment type can be modified to suit the specific bridge system under consideration. However the model does not have the capability to handle more than one environment simultaneously.

3. Only bridges of one category are modeled. A detailed analysis of bridges with varying importance (like interstate vs. primary or secondary) is beyond the scope of this research and so all bridges are assumed to be of one category – interstate,

primary or secondary. In the specific databases used for testing purposes only interstate bridges were used.

4.5.2 Policy Analysis File

The policy information was also decided to be segregated from the model. A strategy to be simulated can be converted to a rule-based text file system which can then be read by the model. Thus the network model depicting the bridge management process is static. By varying the policy analysis file, different strategies can be simulated. By varying the input data files, different network can be simulated. This segregation and isolation of components assures that each section has specific responsibilities and any changes made to one component will not affect the others thus promoting flexibility and reusability.

The command GetPolicyInfo in line 45 of the Stroboscope model extracts the element level policy information from the policy file. A complete Policy Analysis file is given in Appendix G. It is a simple text file and can be edited using any text editor like Microsoft Notepad etc. The general format of the Policy info file is given here.

```
/ "Unp Conc Deck/AC Ovl"
13 0 0 0 2 1
/ "P Conc Deck/AC Ovly"
14 0 0 0 2 1
/ "P Conc Deck/Thin Ovl"
18 0 0 0 1 1
.  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .
```

Every element whose policy is to be defined has two lines in the policy file. The first line gives the PONTIS description of the element. Note that the line should be preceded by a '/'. The '/' indicates that the line is a comments line and should be ignored. The next line contains the actual policy information. The first number indicates the element to which this information belongs. The next five numbers indicate the preferred maintenance activity in each condition state. The '0' indicates a do-nothing activity, '1' indicates a maintenance activity type 1 corresponding to that condition state and similarly '2' corresponds to maintenance activity 2. Thus the unpainted concrete deck with AC overlay in condition states 1, 2 and 3 do not get any maintenance activity performed,

whereas maintenance activity type 2 is performed in condition state 4 and maintenance activity type 1 is performed in condition state5. These element level policies need to be defined for all the elements. By varying the preferred activities, different policies can be implemented. For example the maintenance policy given above is more inclined towards reconstruction and rehabilitation. A maintenance policy like 13 1 1 1 2 1 is geared towards preventive maintenance.

Chapter 5

MODEL TESTING

The first step in the testing of the model was to ensure that the model represented reality. This was ensured by an iterative process of interviews with experts and modifications until the final accurate model was developed.

The next step was to ensure that the translation of the understanding/proposed model to Stroboscope was accurate. For this purpose, the interstate bridges of the Salem district of Virginia were chosen to represent actual test data. This data was obtained from the PONTIS databases of VDOT. A total of 102 interstate bridges were simulated. Three different scenarios were tried out for testing purposes.

5.1 SCENARIO 1

This scenario involves the network behavior over time if no maintenance activities are performed. This is modeled by assigning no maintenance budget for the 10 years of simulation. The results of this simulation run are given in table 5.1. It can be observed in the table that the average health of the network decreases over time which is what is expected in this case. When these results were shown to the experts their response to the behavior of the network was "The prediction of the network where the average HI falls from 85 to 75 is accurate. Further since most of the interstate bridges are new, the low drop in the HI (about 10) over a span of 10 years is also acceptable." (Hackett, 2002)

5.2 SCENARIO 2

Having addressed the issue of populating the model with actual data and trying out test simulations, the next question was "Is it possible to represent the currently used maintenance strategies in the Policy Analysis framework that has been developed?" To answer this question a second scenario was devised. This scenario involves the simulation of the network with the currently used VDOT maintenance policies. To capture the required maintenance policies, these maintenance policies had to be translated

into the rules and incorporated into the Policy Analysis file. To achieve this goal, several interviews with VDOT bridge engineers were conducted to determine the actual maintenance policies used in the field. For each element, the different maintenance policies possible in all possible condition states were enumerated and the bridge engineers were asked to pick the most commonly used/preferred maintenance actions. A complete list of all the maintenance actions examined in the interviews is given in Appendix F. The highlighted actions are the preferred maintenance actions for a condition state. This information was in turn translated into the policy analysis text file and used as an input for the simulation. This simulation output is shown in table 5.1. The response of the personnel involved to this output was "Comparing with scenario 1 the prediction of the model that the network health improves is a valid prediction. Further a budget of 1 million dollars is sufficient to almost maintain the network in its present state which is an acceptable prediction." (Hackett, 2002)

5.3 SCENARIO 3

The next scenario explores the possibility of the ease of modifying the existing strategies to perform "what-if" analyses. The currently used VDOT maintenance policies were used as a starting point and several maintenance actions were modified to determine their impact on the network.

Decks and Girders are an important part of every bridge and are a major contribution to the health index of the bridges. Therefore they were chosen for analysis in this scenario. It was observed that the VDOT maintenance strategies typically had a Do-Nothing policy for the initial condition states and performed reconstruction/replacement in the later stages especially for the elements under consideration. It was decided to change this policy to prefer preventive maintenance over maintenance on an as-needed basis. For this three different cases were devised. The first case involved changing the element level policies for the deck elements to ensure preventive maintenance. The second case involved changing the element level policies for the girder elements and the final case was a combination of the first two cases. The specific Policy Analysis files were generated in all three cases. A detailed list of the policies that were altered is given

in tables 5.3 and 5.4. For the purpose of comparison a target distribution of the bridges among the various categories was decided based on expert opinion as – Excellent – 5%, Good – 75% and Fair – 20%. The simulations were compared to this target distribution to determine the effectiveness of the policies used.

Tables 5.5 through 5.8 illustrate the results in a tabular format for easy comparison. A comparison of the four tables indicates that the element level policies can have a significant impact on the overall network even though the budgets are the same. Table 5.6 and 5.7 show a improvement in the overall HI of the network as opposed to Table 5.5 over 10 years, indicating that preventive maintenance is more beneficial than reconstruction in the long run. The specific Stroboscope commands required to test these scenarios are summarized at the end of the corresponding tables. For the commands to work, the specified policy analysis file should be available.

The simulation results are summarized in figures 5.1 and 5.2. It can be observed in figure 5.1 that when no maintenance is performed the network deteriorates steadily from 85.56 to 75.35. When the currently used VDOT policies were simulated, the network improves from 85.56 to 83.57. However the network does not match the target distribution that is aimed for as observed in figure 5.2. The modified policies that were simulated achieve a better improvement in the network indicating that preventive maintenance is more beneficial compared to reconstruction.

On interviewing the bridge engineers about these predictions their response was "Considering the fact that the CoRe definitions for decks cover only 10% deterioration in the first three condition states and girders cover much more it is logical that changes in the girder policies will have a significantly higher impact on the network as opposed to decks. It is also likely to achieve improvements of these magnitudes in the network by varying the maintenance policies. However VDOT is currently in the process of redefining the CoRe element definitions for the condition states, the transition probabilities and cost models to suit their needs. Therefore the results should not be attached much significance until the input data to the model has been verified and validated" (Hackett, 2002).

These scenarios demonstrate the validity of the simulation model and indicate that the model represents the bridge management process to a reasonable degree of accuracy. Any simulation model is as accurate as the data provided to it. Therefore even though the simulation model has been verified, the data extracted from the PONTIS databases must be validated before any detailed analysis of the network policies can be performed. A detailed examination and validation of the input data followed by a thorough policy analysis is beyond the scope of this study.

TABLE 5.1 No Maintenance Budget

	Excellent	Good	Fair	Poor	VPoor
Target Distribution	5	77	20	0	0

Year	HI	Excellent	Good	Fair	Poor	VPoor	$'s Budget	$'s Left
0	85.56	17	52	27	6	0	0	0
1	84.54	15	51	30	6	0	0	0
2	83.52	9	52	32	9	0	0	0
3	82.5	7	52	31	12	0	0	0
4	81.47	6	47	35	14	0	0	0
5	80.45	4	46	38	14	0	0	0
6	79.42	1	42	44	15	0	0	0
7	78.4	0	36	48	18	0	0	0
8	77.38	0	34	44	24	0	0	0
9	76.36	0	33	43	26	0	0	0
10	75.35	0	28	46	28	0	0	0

Set Budget in Line 9 to 0 using the following line
ARRAY FUNDSMTRX 11; / {1000000 1000000 … .. ….. };

TABLE 5.2 VDOT Element Level Policies with Budget of 1 Million

Year	HI	Excellent	Good	Fair	Poor	VPoor	$'s Budget	$'s Left
0	85.56	17	52	27	6	0	0	0
1	86.32	19	54	26	3	0	1,000,000	50
2	86.43	15	57	26	4	0	1,000,000	186
3	86.21	16	56	27	3	0	1,000,000	605
4	85.92	15	58	26	3	0	1,000,000	1,997
5	85.66	11	61	28	2	0	1,000,000	6,020
6	84.95	6	66	28	2	0	1,000,000	68,487
7	84.65	0	72	28	2	0	1,000,000	6,972
8	84.21	0	70	30	2	0	1,000,000	7,373
9	83.87	0	69	31	2	0	1,000,000	9,130
10	83.57	0	69	31	2	0	1,000,000	4,407
						Ave yearly Expenditure		989,477

Set Budget in Line 9 to 1 million and use VDOT Policy file in line 44 using the following lines
ARRAY FUNDSMTRX 11 {1000000 1000000 … .. ….. };
ASSIGN dat OpenInputFile["PolicyAnalysis.txt"];

TABLE 5.3 Description of Modified Deck Policies

Bare Concrete Deck (12)				
Condition State	**Current PONTIS Policies**		**Modified Policies**	
	Act No.	**Maintenance Act Description**	**Act No.**	**Maintenance Act Description**
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	0	Do Nothing
3	0	Do Nothing	2	Repair Spalls and Delaminations & add a protective system on entire deck
4	2	Repair Spalls and Delaminations & add a protective system on entire deck	2	Repair Spalls and Delaminations & add a protective system on entire deck
5	1	Repair Spalls and Delaminations & add a protective system on entire deck	1	Repair Spalls and Delaminations & add a protective system on entire deck

Unprotected Concrete Deck w/ ACOverlay (13)				
Condition State	**Current PONTIS Policies**		**Modified Policies**	
	Act No.	**Maintenance Act Description**	**Act No.**	**Maintenance Act Description**
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	0	Do Nothing
3	0	Do Nothing	1	Repair substrate and replace overlay
4	2	Repair substrate and replace overlay	2	Repair substrate and replace overlay
5	1	Repair substrate and replace overlay	1	Repair substrate and replace overlay

Protected Concrete Deck w/AC Overlay (14)				
Condition State	**Current PONTIS Policies**		**Modified Policies**	
	Act No.	**Maintenance Act Description**	**Act No.**	**Maintenance Act Description**
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	0	Do Nothing
3	0	Do Nothing	1	Repair substrate and replace overlay
4	2	Repair substrate and replace overlay	2	Repair substrate and replace overlay
5	1	Repair substrate and replace overlay	1	Repair substrate and replace overlay

Protected Concrete Deck w/AC Overlay (22)				
Condition State	**Current PONTIS Policies**		**Modified Policies**	
	Act No.	**Maintenance Act Description**	**Act No.**	**Maintenance Act Description**
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	1	Repair spalls and delaminations
3	0	Do Nothing	1	Repair spalls and delaminations
4	1	Repair spalls and delaminations	1	Repair spalls and delaminations
5	1	Replace overlay	1	Replace overlay

Concrete Deck w/ Coated bars (26)				
Condition State	**Current PONTIS Policies**		**Modified Policies**	
	Act No.	**Maintenance Act Description**	**Act No.**	**Maintenance Act Description**
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	1	Repair spalls and delaminations
3	0	Do Nothing	1	Repair spalls and delaminations
4	2	Repair spalls and delaminations & add a protective system on entire deck	2	Repair spalls and delaminations & add a protective system on entire deck
5	1	Repair spalls and delaminations & add a protective system on entire deck	1	Repair spalls and delaminations & add a protective system on entire deck

TABLE 5.4 Description of Modified Girder Policies

P/S Concrete Web/Box Girder (104)				
Condition State	Current PONTIS Policies		Modified Policies	
	Act No.	Maintenance Act Description	Act No.	Maintenance Act Description
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	1	Seal cracks and minor patching
3	1	Clean steel and patch (and/or seal)	1	Clean steel and patch (and/or seal)
4	2	Replace unit	2	Replace unit

Unpainted Steel Open Girder/Beam (106)				
Condition State	Current PONTIS Policies		Modified Policies	
	Act No.	Maintenance Act Description	Act No.	Maintenance Act Description
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	1	Clean and paint
3	1	Clean and paint	1	Clean and paint
4	1	Rehab unit	1	Rehab unit

Painted Steel Open Girder/Beam (107)				
Condition State	Current PONTIS Policies		Modified Policies	
	Act No.	Maintenance Act Description	Act No.	Maintenance Act Description
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	1	Surface clean
3	0	Do Nothing	1	Spot blast, clean and paint
4	2	Replace paint system	2	Replace paint system
5	1	Rehab unit	1	Rehab unit

P/S Open Girder/Beam (109)				
Condition State	Current PONTIS Policies		Modified Policies	
	Act No.	Maintenance Act Description	Act No.	Maintenance Act Description
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	1	Seal cracks and minor patching
3	1	Clean stell and patch (and/or seal)	1	Clean stell and patch (and/or seal)
4	2	Replace unit	2	Replace unit

Reinforced Open Girder/Beam (110)				
Condition State	Current PONTIS Policies		Modified Policies	
	Act No.	Maintenance Act Description	Act No.	Maintenance Act Description
1	0	Do Nothing	0	Do Nothing
2	0	Do Nothing	1	Seal cracks and minor patching
3	1	Clean rebar and patch (and/or seal)	1	Clean rebar and patch (and/or seal)
4	1	Rehab unit	1	Rehab unit

TABLE 5.5 VDOT Element Level Policies with Budget of 1 Million

	Excellent	Good	Fair	Poor	VPoor
Target Distribution	5	77	20	0	0

Year	HI	Excellent	Good	Fair	Poor	VPoor	Budget	$'s left
0	85.56	17	52	27	6	0	$0	$0
1	86.32	19	54	26	3	0	$1,000,000	$50
2	86.43	15	57	26	4	0	$1,000,000	$186
3	86.21	16	56	27	3	0	$1,000,000	$605
4	85.92	15	58	26	3	0	$1,000,000	$1,997
5	85.66	11	61	28	2	0	$1,000,000	$6,020
6	84.95	6	66	28	2	0	$1,000,000	$68,487
7	84.65	0	72	28	2	0	$1,000,000	$6,972
8	84.21	0	70	30	2	0	$1,000,000	$7,373
9	83.87	0	69	31	2	0	$1,000,000	$9,130
10	83.57	0	69	31	2	0	$1,000,000	$4,407
						Avg. Yearly Budget		**$989,477**

Set Budget in Line 9 to 1 million and use VDOT Policy file in line 44 using the following lines
ARRAY FUNDSMTRX 11 {1000000 1000000 … .. ….. };
ASSIGN dat OpenInputFile["PolicyAnalysis.txt"];

TABLE 5.6 Modified Deck Policies

Year	HI	Excellent	Good	Fair	Poor	VPoor	Budget	$'s left
0	85.56	17	52	27	6	0	$0	$0
1	86.13	20	51	25	6	0	$1,000,000	$12
2	86.2	15	57	25	5	0	$1,000,000	$182
3	86.01	16	58	22	6	0	$1,000,000	$108
4	85.6	14	60	21	7	0	$1,000,000	$1,391
5	85.45	10	63	23	6	0	$1,000,000	$739
6	85.21	6	69	24	3	0	$1,000,000	$2,163
7	84.81	2	70	28	2	0	$1,000,000	$4,905
8	84.62	0	70	29	3	0	$1,000,000	$12,284
9	84.11	0	71	25	6	0	$1,000,000	$8,146
10	83.77	0	69	30	3	0	$1,000,000	$4,943
						Avg. Yearly Budget		**$996,513**

Set Budget in Line 9 to 1 million and use VDOT Policy file in line 44 using the following lines
ARRAY FUNDSMTRX 11 {1000000 1000000 … .. ….. };
ASSIGN dat OpenInputFile["PolicyAnalysis-modifiedDeck.txt"];

TABLE 5.7 Modified Girder Policies

Year	HI	Excellent	Good	Fair	Poor	VPoor	Budget	$'s left
0	85.56	17	52	27	6	0	$0	$0
1	86.6	22	52	24	4	0	$1,000,000	$27
2	87.16	21	54	23	4	0	$1,000,000	$313
3	87.54	24	54	20	4	0	$1,000,000	$2,757
4	87.78	25	55	19	3	0	$1,000,000	$1,662
5	88.02	22	63	14	3	0	$1,000,000	$1,239
6	88.08	21	67	11	3	0	$1,000,000	$2,123
7	88.19	22	68	10	2	0	$1,000,000	$1,075
8	88.18	10	81	9	2	0	$1,000,000	$607
9	88.1	8	83	10	1	0	$1,000,000	$3,761
10	88.21	7	87	7	1	0	$1,000,000	$3,080
							Avg. Yearly Budget	**$998,336**

Set Budget in Line 9 to 1 million and use VDOT Policy file in line 44 using the following lines
ARRAY FUNDSMTRX 11 {1000000 1000000 … .. ….. };
ASSIGN dat OpenInputFile["PolicyAnalysis-modifiedgirders.txt"];

TABLE 5.8 Modified Deck and Girder Policies

Year	HI	Excellent	Good	Fair	Poor	VPoor	Budget	$'s left
0	85.56	17	52	27	6	0	$0	$0
1	86.42	23	48	27	4	0	$1,000,000	$122
2	86.79	23	49	26	4	0	$1,000,000	$859
3	87.06	24	51	23	4	0	$1,000,000	$88
4	87.29	23	56	18	5	0	$1,000,000	$1,068
5	87.49	25	56	16	5	0	$1,000,000	$7,189
6	87.43	23	60	18	4	0	$1,000,000	$3,570
7	87.45	17	65	16	4	0	$1,000,000	$6,746
8	87.36	9	76	19	4	0	$1,000,000	$8,160
9	87.32	9	78	11	4	0	$1,000,000	$5,513
10	87.26	7	78	14	3	0	$1,000,000	$1,367
							Avg. Yearly Budget	**$996,532**

Set Budget in Line 9 to 1 million and use VDOT Policy file in line 44 using the following lines
ARRAY FUNDSMTRX 11 {1000000 1000000 … .. ….. };
ASSIGN dat OpenInputFile["PolicyAnalysis-modifiedgirdersanddeck.txt"];

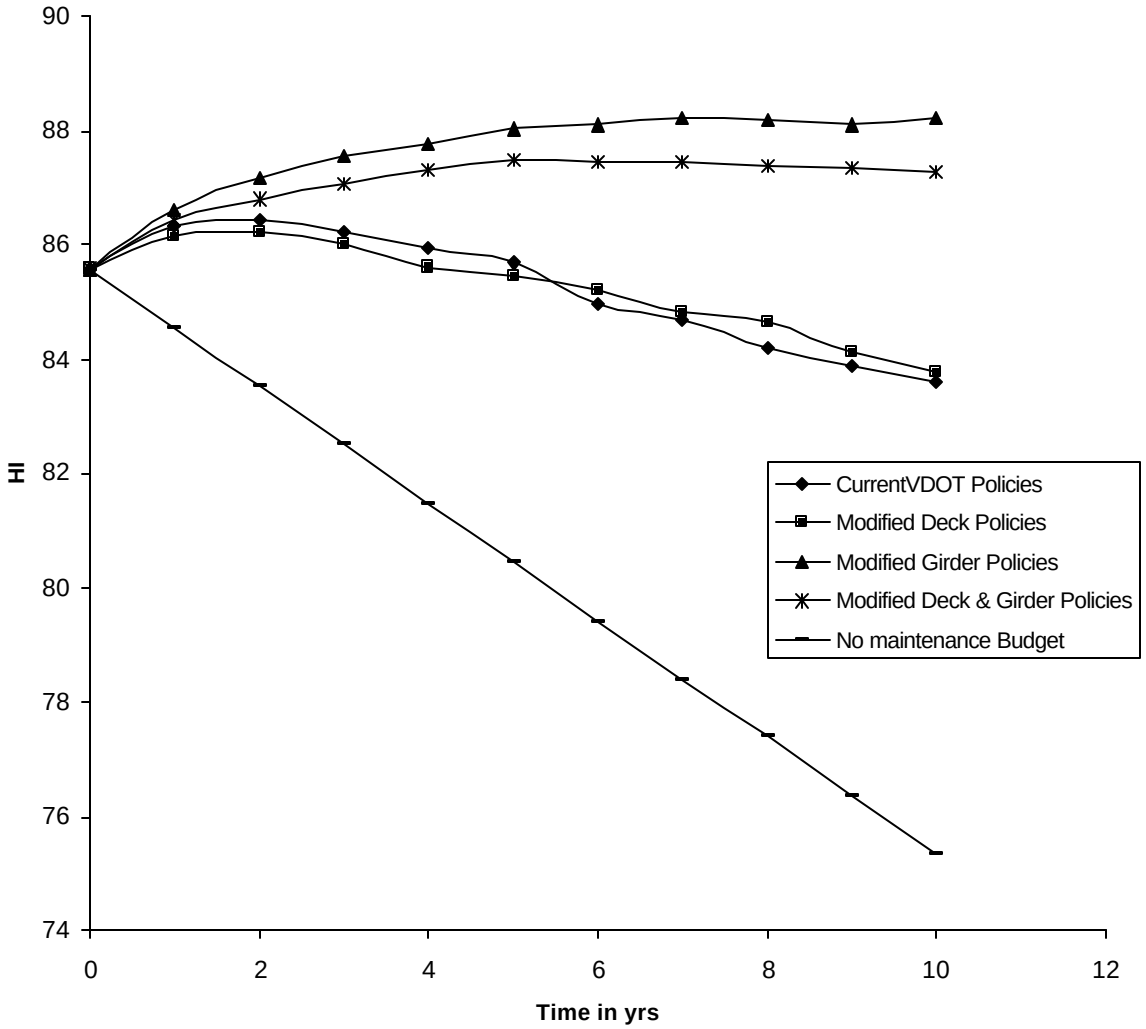

FIGURE 5.1 - Comparison of Element Level Policies

* These results are based on experimental data obtained from the PONTIS databases of Salem District of Virginia. However this data has not been validated and no qualitative inferences should be drawn from the figures regarding the nature of the policies or the improvements acchieved. This figure merely validates the accuracy and working of the Policy Analysi s Tool.

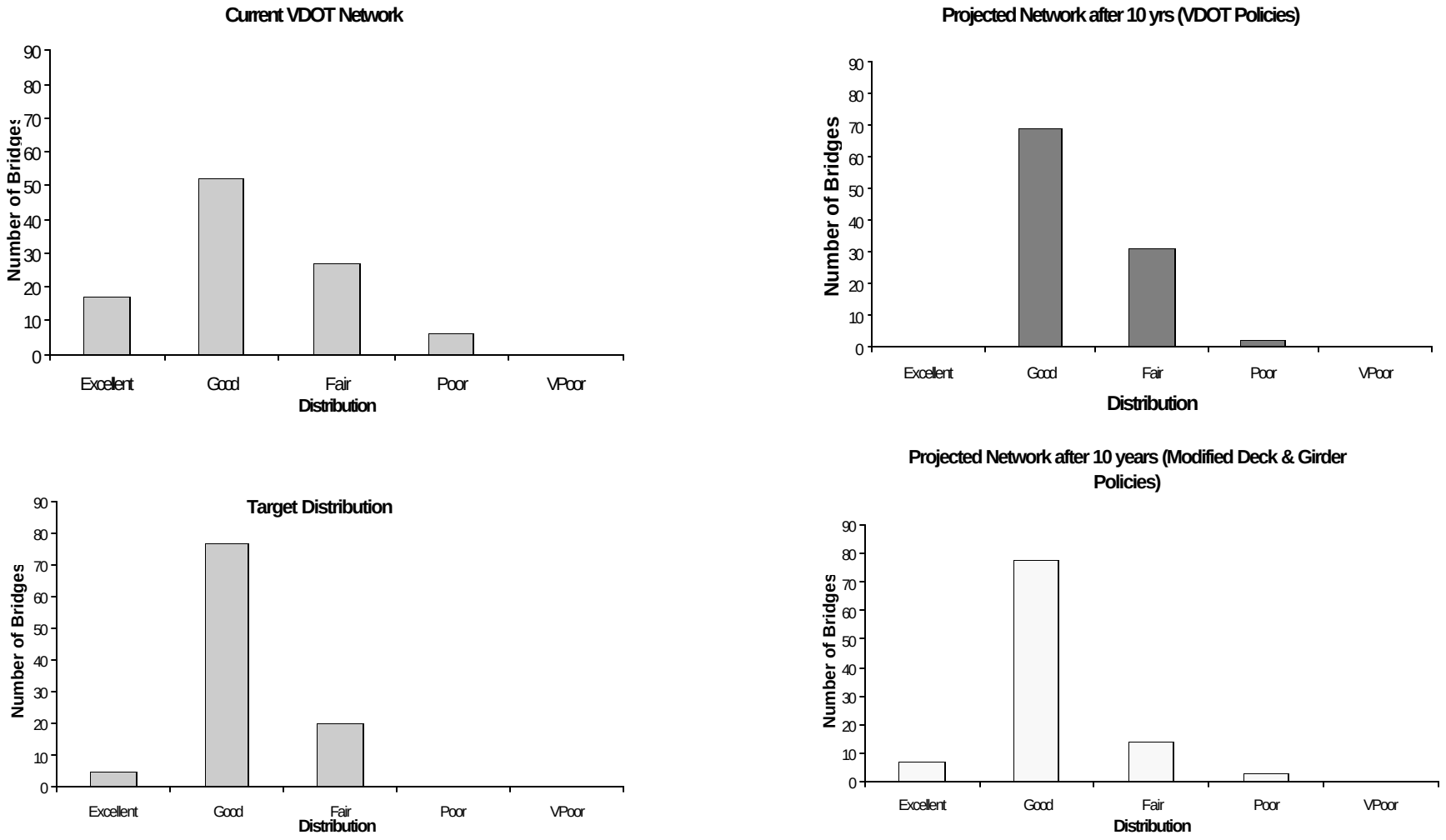

FIGURE 5.2 Comparison of Actual vs. Target Distributions

Chapter 6

CONCLUSION and FURTHER STUDIES

6.1 RESEARCH SUMMARY AND CONCLUSION

Efficient management of the scarce resources is the most important challenge facing bridge management personnel today. Several researchers have developed effective policy analysis tools to help the decision makers manage the funds effectively and maintain bridges.

The objective of this research was to develop a suitable framework for the application of DES as a policy analysis tool and to ensure that this framework is generic and can be applied to a variety of networks and test out a variety of strategies.

The first step was developing a thorough understanding of the bridge management process and the decision making process involved. This was obtained through literature review and interviews with the experts involved. This was followed by the planning and development of the three modules - the data module, the simulation module and the policy query module, which integrate seamlessly to drive the tool. At every stage of the development expert opinion was sought to ensure that the tool was accurate.

The simulation module was developed initially based on the understanding of the bridge management. A careful analysis of literature and the databases on Salem district of Virginia helped in the development of the data module using text based files. The format of these files was chosen to mimic that of the PONTIS PDI files. This ensured that the transfer of data from the PONTIS databases into the simulation model is very easy. However since the simulation module is not linked directly to PONTIS, a slight change in the input file mechanism will ensure that the tool works with BRIDGIT or any other bridge management system which satisfies the data requirements of the model.

The final and the most important module was the policy analysis which was also decided to be text based in a format easy to understand and change. Based on this the

policy analysis tool was developed using discrete event simulation. The different kinds of policies that can be simulated are specified at the element level. The model queries the preferred maintenance activities for each element in every condition state. By altering these activities, different policies can be programmed into the model.

The simulation model can process PONTIS data and simulate the network. Therefore the existing methods of inspection are sufficient and the model does not have any additional data requirements. A few test simulations were run to demonstrate the working and validity of the model. It was observed that modifications of the policies could yield significant improvement in the network conditions. All the simulations were performed on a Pentium MMX processor with 96Mb RAM and Windows 2000 operating system. The simulations processed the 102 interstate bridges and took less than a minute demonstrating the effectiveness of the tool. Several simulations can be run and analyzed almost instantaneously thereby bringing discrete event simulation into the realm of practical use.

6.2 FURTHER STUDIES

The model could be extended in several directions. The first is the data calibration. The model currently imports deterioration and cost models from the PONTIS database. However PONTIS models are discrete value based and therefore not effective. To capture the inherent randomness in the data, distribution models have to be developed for the deterioration and costs. In addition the durations for the maintenance activities have to be developed. However the detailed development of stochastic distributions for the data is beyond the scope of this study and is recommended as the next step. This will in turn calibrate and validate the model for the SALEM district of Virginia thereby transforming it from an effective yet academic tool to a complete and practical policy analysis tool.

The improvability index is based on the Health Index. Health Index fails to take into account factors like Annual Daily Traffic (ADT), indirect costs like Traffic control etc. In order to incorporate these factors, Virginia Department of Transportation has developed the Maintainability Index (MI). A complete explanation of the MI and the

factors involved is given in Appendix I. However the MI is based on the old NBI rating instead of the CoRe element definition. The MI can be expanded to be based on the CoRe element definition as an extension to the model. Then this new MI can be incorporated into the model as the selection criterion instead of the Improvability Index.

The selection criterion needs to be improved. The selection of bridges currently is based on the Improvability Index alone. The bridges that yield the maximum returns for the resources consumed are selected. However typically the goal of a transportation agency is to provide a minimum level of service in conjunction with efficient utilization of resources. Therefore the selection criterion needs to incorporate a trigger mechanism that selects a bridge for maintenance if the bridge condition (HI or MI) falls below a specified value.

Another useful extension will be the development of a graphical user interface that will make it easier for the end user to work with the model. Also an expert system can be developed which will query the user for the different kinds of strategies and maintenance policies.

REFERENCES

1. Bieñ J., "Expert Functions in Bridge Management Systems", *Transportation Research Circular 498*, 1999.
2. Cambridge Systematics Inc., "PONTIS Release 4.0 User's Manual", AASHTO Inc., Washington, August 2001.
3. Cesare M., Santamarina J.C., Turkstra C. J., Vanmarcke E., "Risk-Based Bridge Management", *Journal of Transportation Engineering, Vol. 119, No.5*, 1993.
4. Flaig K.D., Lark R.J., "Integration of Reliability-Based Assessment Techniques into an Advance BMS", *Transportation Research Circular 498*, 1999.
5. Frangopal D.M., Enright M.P., Estes A.C., "Integration of Maintenance, Repair, and Replacement Decisions in Bridge Management Based Reliability, Optimization, and Life-Cycle Cost", *Transportation Research Circular 498*, 1999.
6. Garcia-Diaz A., Liebman J.S., "Optimal Strategies for Bridge Replacement", *Journal of Transportation Engineering*, 1983.
7. Golabi K., Thompson P.D., Hyman W.A., "Pontis Technical Manual, a Network Optimization System for Bridge Improvements and Maintenance", *Report to FHWA*, Cambridge Systematics / Optima, 1992.
8. Gopal S, and Majidzadeh K, "Application of Markov Decision Process to Level-of-Service-Based Maintenance Systems", *Transportation Research Record* 1304, TRB, 1991.
9. Green S.G. and Richardson J.A., "Development of a Bridge Management System in Alabama", *Transportation Research Circular 423*, April 1994.
10. Hackett D. "Personal Interview in Salem with Srinath Devulapalli dated 04-21-2002", 2002.
11. Hawk H., "BRIDGIT Deterioration Models" *Transportation Research Record 1490*, TRB, 1995.

12. Hyman W.A., and Hughes D.J., "Computer Model of Life-Cycle Cost Analysis of Statewide Bridge Repair and Replacement Needs", *Transportation Research Record 899,* 1983.

13. Johnston D.W., Chen C., and Abed-Al-Rahim I., "Developing User Costs for Bridge Management Systems" *Transportation Research Circular 423,* 1994.

14. Johnston D. W. and Lee J.D. "Analysis of Bridge Management Data in North Carolina" *Transportation Research Circular 423,* April 1994.

15. Johnston D.W. and Zia, "A Level of Service System for Bridge Evaluation", *Transportation Research Record no. 899,* 1984

16. Khan M.S., "Bridge Management Systems: Past, Present, and Future", *Concrete International,* 2000.

17. Kim W., "A Systems Approach to Transportation Infrastructure Management: Development of a Highway Management System for the Virginia DOT", A Ph.D. Dissertation; Dept. of Civil Engineering; Virginia Polytechnic and State University, 1996.

18. Kim K., "A Transportation Planning Model for State Highway Management: A Decision Support System Methodology to Achieve Sustainable Development" A Ph.D. Dissertation; Dept. of Civil Engineering; Virginia Polytechnic and State University, 1998.

19. Kleywegt A. J., and Sinha K.C., "Tools for Bridge Management Data Analysis" *Transportation Research Circular 423,* April 1994.

20. Larsen E.S., and Holtz J., "Inspection, Monitoring, and Priority-Ranking of Bridges", *Transportation Research Circular 498,* 1999.

21. Lauzon R.G., Sime J.M., "Connecticut's Bridge Management Information System" *Transportation Research Circular 423,* April 1994.

22. Law A.M., and Kelton W.D., "Simulation Modeling and Analysis", McGraw-Hill 2000.

23. Lipkus S.E., "BRIDGIT Bridge Management System Software" *Transportation*

Research Circular 423, April 1994.

24. Martinez J.C., "STROBOSCOPE – State and Resource Based Simulation of Construction Processes", A Ph.D. Dissertation; Dept. of Civil Engineering, University of Michigan, 1996.

25. Martinez J.C., and Ioannou P.G., "General Purpose Systems for Effective Construction Simulation", *Journal of Construction Engineering and Management,* 125(4), ASCE, 265-276, 1999.

26. Miyamoto A., Kawamura K., Nakamura H., "Bridge Management System and Maintenance Optimization for Existing Bridges", *Computer-Aided Civil and Infrastructure Engineering,15,* 2000.

27. Mohamed H. A., Halim O.A., and Razaqpur A.G., "Use of Neural Networks in Bridge Management Systems" *Transportation Research Record 1490,* TRB, 1995.

28. Oravec J.D., "PennDOT's Bridge Management Decision Support Process" *Transportation Research Circular 423,* April 1994.

29. Roberts J.E., Shepard R., "Bridge Management for the 21st Century", *Transportation Research Record 1696,* 1999.

30. Schrer W. T., Glagola D. M., "Markovian Models for Bridge Maintenance Management", *Journal of Transportation Engineering Vol. 120, no. 1,* 1994.

31. Shirole A. M., "Bridge Management to the Year 2000 and Beyond" *Transportation Research Circular 423,* April 1994.

32. Shirole A. M., Winkler W.J., and Fitzpatrick M.W., "Bridge Management Decision Support" *Transportation Research Circular 423,* April 1994.

33. Small and Cooper, "Condition of the Nation's Highway Bridges, A Look at the Past, Present and Future", *TR News, no 194,* 1998.

34. Smilowitz K., Madanat S., "Optimal Inspection and Maintenance Policies for Infrastructure Networks", *Computer-Aided Civil and Infrastructure Engineering, 15,* 2000.

35. Sobanjo J.O., "Cost Estimating Under Uncertainty: Issues in Bridge Management" *8th International Bridge Management Conference*, April 1999.

36. Thompson P.D., and Shepard R.W., "PONTIS" *Transportation Research Circular 423,* April 1994.

37. Turner D. S., and Richardson J. A., "Bridge Management System and Data Needs and Data Collection" *Transportation Research Circular 423,* April 1994.

38. Vitale J. D., Sinha K. C., Woods R. E., "Analysis of Optimal Bridge Programming Policies" *Transportation Research Record 1561,* 1996.

39. Woods R.E., "INDIANA's Approach to a Bridge Management System", *Transportation Research Circular 423,* April 1994.

40. "1999 Status of the Nation's Highways, Bridges, and Transit: Conditions & Performance", Report to Congress Executive Summary, Federal Highway Administration, U.S Department of Transportation, 1999.

41. "Element Data Collection Manual" Virginia Transportation Research Council, January 1996.

42. "Recording and Coding Guide for the Structure Inventory and Appraisal of the Nation's Bridges" Report No. FHWA-PD-96-001, U.S. Department of Transportation, Federal Highway Administration, December 1995.

APPENDIX A

Element Level Data

A major element is a component of a bridge (such as abutment, girders, piles etc.), which can be further subdivided by material type (such as prestressed concrete, timber, weathering steel etc.). The list of elements is given below. The X indicates the existing possible elements of the given material type.

BRIDGE ELEMENTS

	Material Type							
	Steel		Concrete		Timber	Protected	Unprotected	Other
	Unptd	Ptd	P/S	Reinf				
SUPERSTRUCTURE								
Closed Web/Box girder	X	X	X	X				
Open Girder/Stringer	X	X	X	X	X			
Stringer	X	X	X	X	X			
Thru Truss(Bottom Chord)	X	X						
Thru Truss(Excluding bottom chord)	X	X						
Deck Truss	X	X						
Timber Truss					X			
Arch	X	X	X	X				X
Cable	X	X						
Floor Beam	X	X	X	X	X			
Pin & Hanger Assembly	X	X						

P/S - Prestressed

Unptd - Unpainted

	Material Type							
	Steel		Concrete		Timber	Protected	Unprotected	Other
SUBSTRUCTURE	Unptd	Ptd	P/S	Reinf				
Column or pile extension	X	X	X	X	X			
Pier Wall				X				X
Abutment				X	X			X
Wingwall				X	X			X
Slope						X	X	
Submerged Pile Cap/Footing				X				
Submerged Pile Cap/Footing	X		X	X	X			
Cap	X	X	X	X	X			
Culvert	X			X	X			X

Decks/Slabs	Material Type	
	Decks	Slabs
Concrete(Bare)	X	X
Concrete covered with fill		X
Concrete unprotected w/AC overlay	X	X
Concrete protected with AC overlay	X	X
Concrete w/Thin overlay	X	X
Concrete w/Rigid overlay	X	X
Concrete w/Coated bars	X	X
Concrete w/cathodic protection	X	X
Open Grid - Steel	X	
Concrete Filled Grid - Steel	X	
Corrugated/Orthotropc/Etc	X	
Timber (Bare)	X	X
Timber w/AC Overlay	X	X

OTHERS	Material Type					
	Metal		Concrete		Timber	Other
	Uncoated	Coated	P/S	Reinf		
Strip Seal Expansion Joint						X
Pourable Joint Seal						X
Compression Joint Seal						X
Assembly Joint Seal						X
Open Expansion Joint						X
Elastometric Bearing						X
Movable Bearing						X
Enclosed/Concealed Bearing						X
Fixed Bearing						X
Pot Bearing						X
Disk Bearing						X
Approach Slab			X	X		
Bridge Railing	X	X		X	X	X
Sidewalk	X			X	X	

SMART FLAGS	
Steel Fatigue	X
Pack Rust	X
Deck Cracking	X
Soffit of Deck	X
Underside of Overhang	X
Soffit w/ SIP Forms	X
Settlement	X
Scour	X
Traffic Impact	X
Section Loss	X
Utilities	X
Drains	X
Lighting	X
Roadway Over Culvert	X

APPENDIX B

Calculation of HI

HI=(ΣCEV/ΣTEV)*100

Where TEV is the total element value and CEV is the current element value.

TEV=Total element quantity*failure cost of the element (FC)

CEV=Σ(Quantity Condition $State_i$*WF_i)*FC

The condition state weighting factor WF is given by

WF=[1-(Condition State # -1)(1/(State Count –1))]

Given below is an example of the calculations of HI from Roberts and Shepard (1999)

Condition State Weighting Factor, WF

Number of Condition States	Weight Factor				
	State1	State2	State3	State4	State5
3 Condition States	1	0.5	0		
4 Condition States	1	0.67	0.33	0	
5 Condition States	1	0.75	0.5	0.25	0

Sample Element Distribution for a Bridge

Element Description	Units	Total Quantity	State1	State2	State3	State4	State5	Unit Failure Cost(FC)
Conc. Deck	Sq. m	300			300			$600
Steel Girger	m	100	61	34	5			$3,500
RC Abutment	m	24	24					$7,700
RCColumn	each	4	4					$9,000
Joint Seal	m	24			24			$556

Determination of the bridge total element value (TEV)

Element Description	Calculation	Resulting Element Value
Conc. Deck	300*$600	180,000
Steel Girder	100*$3,500	350,000
RC Abutment	24*$7,700	184,800
RC Column	4*$9,000	36,000
Joint Seal	24*$556	13,344
	Total	**764,144**

Determination of the bridge current element value (CEV)

Element Description	Calculation	CEV	Element Health
Conc. Deck	300*0.5*600	$90,000	50
Steel Girder	((61*1.0)+(34*0.75)+(5*0.5))*3500	$311,500	89
RC Abutment	24*1.0*7700	$184,800	100
RC Column	4*1.0*9000	$36,000	100
Joint Seal	24*0*556	$0	0
	Total	**$622,300**	

From the above calculations we see the overall HI of the bridge is given by

HI=(ΣCEV/ΣTEV)*100 = ($622,300/$764,144)*100 = 81.4

APPENDIX C

Tables Imported From Pontis

TABLE actmodls	
Data Member	**Purpose**
elemkey	Identifies the element about which data is given
skey	Identifies the state about which data is given
akey	Identifies the activity about which data is given
envkey	Identifies the environment about which data is given
prob1	Identifies the probability of returning to state 1
prob2	Identifies the probability of returning to state 2
prob3	Identifies the probability of returning to state 3
prob4	Identifies the probability of returning to state 4
prob5	Identifies the probability of returning to state 5
unitco	The cost of the maintenance activity per unit

TABLE condumdl	
Data Member	**Purpose**
elemkey	Identifies the element about which data is given
envkey	Identifies the environment in which the element is
failagcyco	Gives the failure cost of the element

TABLE eleminsp	
Data Member	**Purpose**
brkey	Identifies the bridge about which data is given
elemkey	Identifies the element about which data is given
skey	Identifies the state about which data is given
envkey	Identifies the environment existing
quantity	Gives the total quantity of the element
pctstate1	Gives the percentage of element in state 1
qtystate1	Gives the quantity of element in state 1
pctstate2	Gives the percentage of element in state 2
qtystate2	Gives the quantity of element in state 2
pctstate3	Gives the percentage of element in state 3
qtystate3	Gives the quantity of element in state 3
pctstate4	Gives the percentage of element in state 4
qtystate4	Gives the quantity of element in state4
pctstate5	Gives the percentage of element in state 5
qtystate5	Gives the percentage of element in state 5

TABLE bridge	
Data Member	**Purpose**
brkey	Identifies the bridge about which data is given
bridge id	Gives the identifier commonly used for that bridge

TABLE ELEMDEFS	
Data Member	**Purpose**
elemkey	Identifies the element about which data is given
statecnt	Identifies the maximum possible states for the element

APPENDIX D

Listing Of Stroboscope Code

```
1. /*****************************************************************************************
2. /* Stroboscope source file generated from Visio drawing
   d:\srin\proj\inputfiledll\inputfilewithfunctions\debug\bridgemaintenance.vsd

3. /*****************************************************************************************

4. /*****************************************************************************************

5. /* General section for problem parameters

6. LOADADDON "InputFile.dll";

7. SAVEVALUE YrsToSimulate* 10;

8. /Funding values YR:    1       2       3       4       5       6       7       8       9
   10

9. ARRAY FUNDSMTRX 11 { 1000000 1000000 1000000 1000000 1000000 1000000 1000000 1000000 1000000
   1000000 0};

10.      SAVEVALUE Funds 0; /Stores the available funds at any instant of time. Is set /from
   FUNDSMTRX every year

11.      /The lower bounds of the ranges
12.      SAVEVALUE Excellent 95;
13.      SAVEVALUE Good 82;
14.      SAVEVALUE Fair 70;
15.      SAVEVALUE Poor 40;

16.      /The minimum delay between successive repairs on a bridge
17.      SAVEVALUE Delay 3;

17a.     SAVEVALUE MaxBridgeCount 4;
```

```
18.      /DECLARATION FOR TRANSITION PROBABILITIES

19.      ARRAY TransProbData 800; /The transition probability data for the deterioration of
   elements

20.      /DECLARATION FOR MR&R ACTIONS AND TRANSITION PROBABILITIES FOR ACTION TYPE 1

21.      ARRAY MaintAct1Data 800;

22.      /DECLARATION FOR MR&R ACTIONS AND TRANSITION PROBABILITIES FOR ACTION TYPE 2

23.      ARRAY MaintAct2Data 800;

24.      /The preferred element level activity matrix
25.      ARRAY PrefMaintAct 800 5;

26.      /Failure cost information for every element
27.      ARRAY FailrCostData 800;

28.      /Array giving the maximum number of allowable states for each element
29.      ARRAY StateCnt 800;

30.      /Variable for handling files
31.      SAVEVALUE data 0;

32.      /Read the data from the PONTIS PDI files
33.      ASSIGN data OpenInputFile["int_actmodls.PDI"];

34.      CALL ReadElementData[data];

35.      ASSIGN data OpenInputFile["Bridges.txt"];
36.      CALL GetBridgeInfo[data];

37.      ASSIGN data OpenInputFile["ElemInsp.txt"];
38.      CALL GetElementInfo[data];

39.      ASSIGN data OpenInputFile["int_elemdefs.PDI"];
```

```
40.      CALL GetStateCount[data];

41.      ASSIGN data OpenInputFile["int_condumdl.PDI"];
42.      CALL GetFailureCost[data];

43.      /Read the element level policies to be implemented
44.      ASSIGN data OpenInputFile["PolicyAnalysis.txt"];
45.      CALL GetPolicyInfo[data];

46.      /*************************************************************************************

47.      /* Definition of resource types

48.      COMPTYPE    Bridge; /BR

49.      SAVEPROPS   Bridge HI TotElems RepairCost BrID QueueFrm LastRepairedTm FHI;

50.      GENTYPE           Counter; /CO

51.      COMPTYPE    Element; /EL

52.      SAVEPROPS   Element State1 State2 State3 State4 State5 Qty ID EHI Source UnitFlrCst
   statecnt S1 S2 S3 S4 S5 Cost1 Cost2 Cost3 Cost4 Cost5 FState1 FState2 FState3 FState4 FState5;

53.      /Variable storing the current element value
54.      VARPROP     Element CEV 'statecnt==5?
55.      (State1+State2*0.75+State3*0.5+State4*0.25)*Qty*UnitFlrCst:
56.      statecnt==4?(State1+State2*0.67+State3*0.33)*Qty*UnitFlrCst:
57.      (State1+State2*0.50)*Qty*UnitFlrCst';

58.      /Variable storing the future element value
59.      VARPROP Element FEV 'statecnt==5?
60.      (FState1+FState2*0.75+FState3*0.5+FState4*0.25)*Qty*UnitFlrCst:
61.      statecnt==4?(FState1+FState2*0.67+FState3*0.33)*Qty*UnitFlrCst:
62.      (FState1+FState2*0.50)*Qty*UnitFlrCst';

63.      /Variable storing the total repair cost of the element
```

```
64.      VARPROP Element RepairCst '
65.      (State1*Cost1+State2*Cost2+State3*Cost3+State4*Cost4+State5*Cost5)*Qty';
66.      /Variable storing the total element value
67.      VARPROP Element TEV 'Qty*UnitFlrCst';
68.      /Variable storing the IC ratio for the bridge
69.      VARPROP Bridge IC 'RepairCost!=0?( (FHI-HI)*1000/RepairCost ):-100';
70.      /*****************************************************************************
71.      /* Definition of network nodes
72.      COMBI             CreateBridges;
73.      ASSEMBLER         AssembleBridges Bridge;
74.      QUEUE             BridgeNetwork Bridge;
75.      QUEUE             BridgesToCreate Bridge;
76.      DISASSEMBLER      DisassemBridges Bridge;
77.      QUEUE             Elements Element;
78.      QUEUE             InWrkBridges Bridge;
79.      COMBI             RepairElements;
80.      QUEUE             RprdElements Element;
81.      COMBI             AssembleElems;
82.      ASSEMBLER         AsmRprdBridges Bridge;
83.      COMBI             OneYear;
```

```
84.      QUEUE             YrCntr Counter;

85.      QUEUE             DeteriorateCntr Counter;

86.      COMBI             UpdateBridges;

87.      QUEUE             Brdgs Bridge;

88.      COMBI             SeperateBridges;

89.      DISASSEMBLER      DisassemBridge Bridge;

90.      ASSEMBLER         AssemBridge Bridge;

91.      COMBI             RepairBridges;
92.      QUEUE             ElementsToCreat Element;

93.      COMBI             Report;

94.      QUEUE             RCntr Counter;
95.      QUEUE             BridgeCount Counter;

96.      /*****************************************************************************

97.      /* Definition of network Links

98.      LINK              B3 AssembleBridges BridgeNetwork;

99.      DISASMBASELINK    B6 DisassemBridges InWrkBridges;

100.     LINK              B7 InWrkBridges AssembleElems;

101.     ASMBASELINK       B8 AssembleElems AsmRprdBridges;

102.     LINK              B9 AsmRprdBridges BridgeNetwork;

103.     LINK              B15 AssemBridge BridgeNetwork;
```

```
104.     LINK             B5 RepairBridges DisassemBridges;
105.     LINK             E2 DisassemBridges Elements;

106.     LINK             E5 RprdElements AssembleElems;

107.     LINK             E6 AssembleElems AsmRprdBridges Element;

108.     ASMBASELINK      B2 CreateBridges AssembleBridges;

109.     LINK             E1 CreateBridges AssembleBridges Element;

110.     LINK             B1 BridgesToCreate CreateBridges;

111.     LINK             E3 Elements RepairElements;

112.     LINK             E4 RepairElements RprdElements;

113.     LINK             C1 YrCntr OneYear;

114.     LINK             C2 OneYear YrCntr;

115.     LINK             C3 OneYear DeteriorateCntr;

116.     LINK             C4 DeteriorateCntr UpdateBridges;

117.     LINK             B10 BridgeNetwork UpdateBridges;
118.     LINK             B11 UpdateBridges Brdgs;

119.     LINK             B12 Brdgs SeperateBridges;

120.     LINK             B13 SeperateBridges DisassemBridge;

121.     DUALBASELINK     B14 DisassemBridge AssemBridge;

122.     LINK             E7 DisassemBridge AssemBridge Element;

123.     LINK             B4 BridgeNetwork RepairBridges;
```

```
124.     LINK             E0 ElementsToCreat CreateBridges;

125.     LINK             R1 BridgeNetwork Report;
126.     LINK             R2 Report BridgeNetwork;

127.     LINK             R3 InWrkBridges Report;

128.     LINK             R4 Report InWrkBridges;

129.     LINK             R5 RCntr Report;

130.     LINK             C5 OneYear RCntr;

131.     LINK             IC1 BridgeCount RepairBridges;
132.     LINK             IC2 AssembleElems BridgeCount;

133.     /*****************************************************************************

134.     /* Statements to assist in the definition of attributes of CreateBridges and its related
   links

135.     /Ensure that the elements are matched to the corresponding bridges
136.     FILTER ElemsofBridge Element 'Source==CreateBridges.Bridge.BrID';

137.     /Array to assist in the updating of states and future states of the elements
138.     ARRAY ActProbData 5 5;

139.     *****************************************************************************/

140.     /* Startup of CreateBridges

141.     PRIORITY         CreateBridges '500';

142.     DRAWUNTIL        B1 'CreateBridges.Bridge.Count';

143.     DRAWUNTIL        E0 'ElementsToCreat.ElemsofBridge.Count==0';
144.     DRAWWHERE        E0 'Source==CreateBridges.Bridge.BrID';
```

```
145.    /*****************************************************************************************

146.    /* Termination of CreateBridges


147.    ONDRAW E0 ASSIGN UnitFlrCst FailrCostData[ID];


148.    /ASSIGN the appropriate cost information
149.    ONDRAW E0 ASSIGN Cost1 '

            (PrefMaintAct[ID,0]==2)?
            GetArrayElement[MaintAct2Data[ID],0,5]:
            ((PrefMaintAct[ID,0]==1)?
            GetArrayElement[MaintAct1Data[ID],0,5]:0)';

150.    ONDRAW E0 ASSIGN Cost2 '

            (PrefMaintAct[ID,1]==2)?
            GetArrayElement[MaintAct2Data[ID],1,5]:
            ((PrefMaintAct[ID,1]==1)?
            GetArrayElement[MaintAct1Data[ID],1,5]:0)';

151.    ONDRAW E0 ASSIGN Cost3 '

            (PrefMaintAct[ID,2]==2)?
            GetArrayElement[MaintAct2Data[ID],2,5]:
            ((PrefMaintAct[ID,2]==1)?
            GetArrayElement[MaintAct1Data[ID],2,5]:0)';

152.    ONDRAW E0 ASSIGN Cost4 '

            (PrefMaintAct[ID,3]==2)?
            GetArrayElement[MaintAct2Data[ID],3,5]:
            ((PrefMaintAct[ID,3]==1)?
            GetArrayElement[MaintAct1Data[ID],3,5]:0)';
```

```
153.     ONDRAW E0 ASSIGN Cost5 '

             (PrefMaintAct[ID,4]==2)?
             GetArrayElement[MaintAct2Data[ID],4,5]:
             ((PrefMaintAct[ID,4]==1)?
             GetArrayElement[MaintAct1Data[ID],4,5]:0)';

154.     /Assign the transition probailty data
155.     ONDRAW E0 ASSIGN ActProbData 0 0 '

156.     PrefMaintAct[ID,0]==2?
157.     (GetArrayElement[MaintAct2Data[ID],0,0]):

158.     PrefMaintAct[ID,0]==1?

159.     (GetArrayElement[MaintAct1Data[ID],0,0]):

160.     (GetArrayElement[TransProbData[ID],0,0])';

161.     ONDRAW E0 ASSIGN ActProbData 0 1 '

162.     PrefMaintAct[ID,0]==2?

163.     (GetArrayElement[MaintAct2Data[ID],0,1]):

164.     PrefMaintAct[ID,0]==1?

165.     (GetArrayElement[MaintAct1Data[ID],0,1]):

166.     (GetArrayElement[TransProbData[ID],0,1])';

167.     ONDRAW E0 ASSIGN ActProbData 0 2 '

168.     PrefMaintAct[ID,0]==2?

169.     (GetArrayElement[MaintAct2Data[ID],0,2]):
```

```
170.     PrefMaintAct[ID,0]==1?
171.     (GetArrayElement[MaintAct1Data[ID],0,2]):
172.     (GetArrayElement[TransProbData[ID],0,2])';
173.     ONDRAW E0 ASSIGN ActProbData 0 3 '
174.     PrefMaintAct[ID,0]==2?
175.     (GetArrayElement[MaintAct2Data[ID],0,3]):
176.     PrefMaintAct[ID,0]==1?
177.     (GetArrayElement[MaintAct1Data[ID],0,3]):
178.     (GetArrayElement[TransProbData[ID],0,3])';
179.     ONDRAW E0 ASSIGN ActProbData 0 4 '
180.     PrefMaintAct[ID,0]==2?
181.     (GetArrayElement[MaintAct2Data[ID],0,4]):
182.     PrefMaintAct[ID,0]==1?
183.     (GetArrayElement[MaintAct1Data[ID],0,4]):
184.     (GetArrayElement[TransProbData[ID],0,4])';
185.     ONDRAW E0 ASSIGN ActProbData 1 0 '
186.     PrefMaintAct[ID,1]==2?
187.     (GetArrayElement[MaintAct2Data[ID],1,0]):
188.     PrefMaintAct[ID,1]==1?
```

```
189.     (GetArrayElement[MaintAct1Data[ID],1,0]):
190.     (GetArrayElement[TransProbData[ID],1,0])';
191.     ONDRAW E0 ASSIGN ActProbData 1 1 '
192.     PrefMaintAct[ID,1]==2?
193.     (GetArrayElement[MaintAct2Data[ID],1,1]):
194.     PrefMaintAct[ID,1]==1?
195.     (GetArrayElement[MaintAct1Data[ID],1,1]):
196.     (GetArrayElement[TransProbData[ID],1,1])';
197.     ONDRAW E0 ASSIGN ActProbData 1 2 '
198.     PrefMaintAct[ID,1]==2?
199.     (GetArrayElement[MaintAct2Data[ID],1,2]):
200.     PrefMaintAct[ID,1]==1?
201.     (GetArrayElement[MaintAct1Data[ID],1,2]):
202.     (GetArrayElement[TransProbData[ID],1,2])';
203.     ONDRAW  E0 ASSIGN ActProbData 1 3 '
204.     PrefMaintAct[ID,1]==2?
205.     (GetArrayElement[MaintAct2Data[ID],1,3]):
206.     PrefMaintAct[ID,1]==1?
207.     (GetArrayElement[MaintAct1Data[ID],1,3]):
```

```
208.     (GetArrayElement[TransProbData[ID],1,3])';
209.     ONDRAW E0 ASSIGN ActProbData 1 4 '
210.     PrefMaintAct[ID,1]==2?
211.     (GetArrayElement[MaintAct2Data[ID],1,4]):
212.     PrefMaintAct[ID,1]==1?
213.     (GetArrayElement[MaintAct1Data[ID],1,4]):
214.     (GetArrayElement[TransProbData[ID],1,4])';
215.     ONDRAW E0 ASSIGN ActProbData 2 0 '
216.     PrefMaintAct[ID,2]==2?
217.     (GetArrayElement[MaintAct2Data[ID],2,0]):
218.     PrefMaintAct[ID,2]==1?
219.     (GetArrayElement[MaintAct1Data[ID],2,0]):
220.     (GetArrayElement[TransProbData[ID],2,0])';
221.     ONDRAW E0 ASSIGN ActProbData 2 1 '
222.     PrefMaintAct[ID,2]==2?
223.     (GetArrayElement[MaintAct2Data[ID],2,1]):
224.     PrefMaintAct[ID,2]==1?
225.     (GetArrayElement[MaintAct1Data[ID],2,1]):
226.     (GetArrayElement[TransProbData[ID],2,1])';
```

```
227.     ONDRAW E0 ASSIGN ActProbData 2 2 '
228.     PrefMaintAct[ID,2]==2?
229.     (GetArrayElement[MaintAct2Data[ID],2,2]):
230.     PrefMaintAct[ID,2]==1?
231.     (GetArrayElement[MaintAct1Data[ID],2,2]):
232.     (GetArrayElement[TransProbData[ID],2,2])';
233.     ONDRAW E0 ASSIGN ActProbData 2 3 '
234.     PrefMaintAct[ID,2]==2?
235.     (GetArrayElement[MaintAct2Data[ID],2,3]):
236.     PrefMaintAct[ID,2]==1?
237.     (GetArrayElement[MaintAct1Data[ID],2,3]):
238.     (GetArrayElement[TransProbData[ID],2,3])';
239.     ONDRAW E0 ASSIGN ActProbData 2 4 '
240.     PrefMaintAct[ID,2]==2?
241.     (GetArrayElement[MaintAct2Data[ID],2,4]):
242.     PrefMaintAct[ID,2]==1?
243.     (GetArrayElement[MaintAct1Data[ID],2,4]):
244.     (GetArrayElement[TransProbData[ID],2,4])';
245.     ONDRAW E0 ASSIGN ActProbData 3 0 '
```

```
246.     PrefMaintAct[ID,3]==2?
247.     (GetArrayElement[MaintAct2Data[ID],3,0]):
248.     PrefMaintAct[ID,3]==1?
249.     (GetArrayElement[MaintAct1Data[ID],3,0]):
250.     (GetArrayElement[TransProbData[ID],3,0])';
251.     ONDRAW E0 ASSIGN ActProbData 3 1 '
252.     PrefMaintAct[ID,3]==2?
253.     (GetArrayElement[MaintAct2Data[ID],3,1]):
254.     PrefMaintAct[ID,3]==1?
255.     (GetArrayElement[MaintAct1Data[ID],3,1]):
256.     (GetArrayElement[TransProbData[ID],3,1])';
257.     ONDRAW E0 ASSIGN ActProbData 3 2 '
258.     PrefMaintAct[ID,3]==2?
259.     (GetArrayElement[MaintAct2Data[ID],3,2]):
260.     PrefMaintAct[ID,3]==1?
261.     (GetArrayElement[MaintAct1Data[ID],3,2]):
262.     (GetArrayElement[TransProbData[ID],3,2])';
263.     ONDRAW E0 ASSIGN ActProbData 3 3 '
264.     PrefMaintAct[ID,3]==2?
```

```
265.     (GetArrayElement[MaintAct2Data[ID],3,3]):
266.     PrefMaintAct[ID,3]==1?
267.     (GetArrayElement[MaintAct1Data[ID],3,3]):
268.     (GetArrayElement[TransProbData[ID],3,3])';
269.     ONDRAW E0 ASSIGN ActProbData 3 4 '
270.     PrefMaintAct[ID,3]==2?
271.     (GetArrayElement[MaintAct2Data[ID],3,4]):
272.     PrefMaintAct[ID,3]==1?
273.     (GetArrayElement[MaintAct1Data[ID],3,4]):
274.     (GetArrayElement[TransProbData[ID],3,4])';
275.     ONDRAW E0 ASSIGN ActProbData 4 0 '
276.     PrefMaintAct[ID,3]==2?
277.     (GetArrayElement[MaintAct2Data[ID],4,0]):
278.     PrefMaintAct[ID,4]==1?
279.     (GetArrayElement[MaintAct1Data[ID],4,0]):
280.     (GetArrayElement[TransProbData[ID],4,0])';
281.     ONDRAW E0 ASSIGN ActProbData 4 1 '
282.     PrefMaintAct[ID,3]==2?
283.     (GetArrayElement[MaintAct2Data[ID],4,1]):
```

```
284.     PrefMaintAct[ID,4]==1?
285.     (GetArrayElement[MaintAct1Data[ID],4,1]):
286.     (GetArrayElement[TransProbData[ID],4,1])';
287.     ONDRAW E0 ASSIGN ActProbData 4 2 '
288.     PrefMaintAct[ID,3]==2?
289.     (GetArrayElement[MaintAct2Data[ID],4,2]):
290.     PrefMaintAct[ID,4]==1?
291.     (GetArrayElement[MaintAct1Data[ID],4,2]):
292.     (GetArrayElement[TransProbData[ID],4,2])';
293.     ONDRAW E0 ASSIGN ActProbData 4 3 '
294.     PrefMaintAct[ID,3]==2?
295.     (GetArrayElement[MaintAct2Data[ID],4,3]):
296.     PrefMaintAct[ID,4]==1?
297.     (GetArrayElement[MaintAct1Data[ID],4,3]):
298.     (GetArrayElement[TransProbData[ID],4,3])';
299.     ONDRAW E0 ASSIGN ActProbData 4 4 '
300.     PrefMaintAct[ID,3]==2?
301.     (GetArrayElement[MaintAct2Data[ID],4,4]):
302.     PrefMaintAct[ID,4]==1?
```

```
303.     (GetArrayElement[MaintAct1Data[ID],4,4]):

304.     (GetArrayElement[TransProbData[ID],4,4])';

305.     /Update the future states based on preferred element level maintenance activities
306.     ONDRAW E0 ASSIGN FState1 '

                  State1*ActProbData[0,0]+
                  State2*ActProbData[1,0]+
                  State3*ActProbData[2,0]+
                  State4*ActProbData[3,0]+
                  State5*ActProbData[4,0]';

307.     ONDRAW E0 ASSIGN FState2 '

                  State1*ActProbData[0,1]+
                  State2*ActProbData[1,1]+
                  State3*ActProbData[2,1]+
                  State4*ActProbData[3,1]+
                  State5*ActProbData[4,1]';

308.     ONDRAW E0 ASSIGN FState3 '

                  State1*ActProbData[0,2]+
                  State2*ActProbData[1,2]+
                  State3*ActProbData[2,2]+
                  State4*ActProbData[3,2]+
                  State5*ActProbData[4,2]';

309.     ONDRAW E0 ASSIGN FState4 '

                  State1*ActProbData[0,3]+
                  State2*ActProbData[1,3]+
                  State3*ActProbData[2,3]+
                  State4*ActProbData[3,3]+
                  State5*ActProbData[4,3]';
```

```
310.     ONDRAW E0 ASSIGN FState5 '

                 State1*ActProbData[0,4]+
                 State2*ActProbData[1,4]+
                 State3*ActProbData[2,4]+
                 State4*ActProbData[3,4]+
                 State5*ActProbData[4,4]';

311.     /Update the number of elements in the bridge
312.     ONRELEASE E1 ASSIGN AssembleBridges.TotElems AssembleBridges.TotElems+1;

313.     /*****************************************************************************

314.     /* Assembly of resources in AssembleBridges

315.     /Update the properties of the bridge

316.     ONASSEMBLY AssembleBridges ASSIGN RepairCost '

317.     AssembleBridges.Element.RepairCst.SumVal';

318.     ONASSEMBLY AssembleBridges ASSIGN HI '

319.     AssembleBridges.Element.TEV.SumVal!=0?

320.     (AssembleBridges.Element.CEV.SumVal/AssembleBridges.Element.TEV.SumVal)*100:0';
321.     ONASSEMBLY AssembleBridges ASSIGN FHI '

322.     AssembleBridges.Element.TEV.SumVal!=0?

323.     (AssembleBridges.Element.FEV.SumVal/AssembleBridges.Element.TEV.SumVal)*100:0';

324.     ONASSEMBLY AssembleBridges ASSIGN LastRepairedTm -(Delay+1);

325.     *****************************************************************************

326.     /* Entry of resources into BridgeNetwork
```

```
327.     ONENTRY BridgeNetwork ASSIGN QueueFrm BridgeNetwork;

328.     VARPROP Bridge NonDelayedHI '( (SimTime-LastRepairedTm>=Delay) & (RepairCost<=Funds) )?
                                        a. IC:-200';

329.     /*****************************************************************************

330.     /* Entry of resources into BridgesToCreate

331.     ONENTRY BridgesToCreate ASSIGN BrID BR__INFO[ResNum-1];

332.     /*****************************************************************************

333.     /* Release of disassembled components in DisassemBridges

334.     /*****************************************************************************

335.     /* Entry of resources into Elements

336.     /*****************************************************************************

337.     /* Entry of resources into InWrkBridges

338.     ONENTRY InWrkBridges ASSIGN QueueFrm InWrkBridges;

339.     /*****************************************************************************

340.     /* Statements to assist in the definition of attributes of RepairElements and its related
   links

341.     /*****************************************************************************

342.     /* Startup of RepairElements
```

```
343.     DURATION          RepairElements '15/365';

344.     /*************************************************************************************

345.     /* Termination of RepairElements

346.     ONRELEASE E4 ASSIGN S1 State1;

347.     ONRELEASE E4 ASSIGN S2 State2;

348.     ONRELEASE E4 ASSIGN S3 State3;

349.     ONRELEASE E4 ASSIGN S4 State4;

350.     ONRELEASE E4 ASSIGN S5 State5;


351.     /Get the transition probability data
352.     ONRELEASE E4 ASSIGN ActProbData 0 0 '

353.     PrefMaintAct[ID,0]==2?
354.     (GetArrayElement[MaintAct2Data[ID],0,0]):

355.     PrefMaintAct[ID,0]==1?

356.     (GetArrayElement[MaintAct1Data[ID],0,0]):

357.     (GetArrayElement[TransProbData[ID],0,0])';

358.     ONRELEASE E4 ASSIGN ActProbData 0 1 '

359.     PrefMaintAct[ID,0]==2?

360.     (GetArrayElement[MaintAct2Data[ID],0,1]):

361.     PrefMaintAct[ID,0]==1?
```

```
362.     (GetArrayElement[MaintAct1Data[ID],0,1]):
363.     (GetArrayElement[TransProbData[ID],0,1])';
364.     ONRELEASE E4 ASSIGN ActProbData 0 2 '
365.     PrefMaintAct[ID,0]==2?
366.     (GetArrayElement[MaintAct2Data[ID],0,2]):
367.     PrefMaintAct[ID,0]==1?
368.     (GetArrayElement[MaintAct1Data[ID],0,2]):
369.     (GetArrayElement[TransProbData[ID],0,2])';
370.     ONRELEASE E4 ASSIGN ActProbData 0 3 '
371.     PrefMaintAct[ID,0]==2?
372.     (GetArrayElement[MaintAct2Data[ID],0,3]):
373.     PrefMaintAct[ID,0]==1?
374.     (GetArrayElement[MaintAct1Data[ID],0,3]):
375.     (GetArrayElement[TransProbData[ID],0,3])';
376.     ONRELEASE E4 ASSIGN ActProbData 0 4 '
377.     PrefMaintAct[ID,0]==2?
378.     (GetArrayElement[MaintAct2Data[ID],0,4]):
379.     PrefMaintAct[ID,0]==1?
380.     (GetArrayElement[MaintAct1Data[ID],0,4]):
```

```
381.     (GetArrayElement[TransProbData[ID],0,4])';
382.     ONRELEASE E4 ASSIGN ActProbData 1 0 '
383.     PrefMaintAct[ID,1]==2?
384.     (GetArrayElement[MaintAct2Data[ID],1,0]):
385.     PrefMaintAct[ID,1]==1?
386.     (GetArrayElement[MaintAct1Data[ID],1,0]):
387.     (GetArrayElement[TransProbData[ID],1,0])';
388.     ONRELEASE E4 ASSIGN ActProbData 1 1 '
389.     PrefMaintAct[ID,1]==2?
390.     (GetArrayElement[MaintAct2Data[ID],1,1]):
391.     PrefMaintAct[ID,1]==1?
392.     (GetArrayElement[MaintAct1Data[ID],1,1]):
393.     (GetArrayElement[TransProbData[ID],1,1])';
394.     ONRELEASE E4 ASSIGN ActProbData 1 2 '
395.     PrefMaintAct[ID,1]==2?
396.     (GetArrayElement[MaintAct2Data[ID],1,2]):
397.     PrefMaintAct[ID,1]==1?
398.     (GetArrayElement[MaintAct1Data[ID],1,2]):
399.     (GetArrayElement[TransProbData[ID],1,2])';
```

```
400.     ONRELEASE E4 ASSIGN ActProbData 1 3 '
401.     PrefMaintAct[ID,1]==2?
402.     (GetArrayElement[MaintAct2Data[ID],1,3]):
403.     PrefMaintAct[ID,1]==1?
404.     (GetArrayElement[MaintAct1Data[ID],1,3]):
405.     (GetArrayElement[TransProbData[ID],1,3])';
406.     ONRELEASE E4 ASSIGN ActProbData 1 4 '
407.     PrefMaintAct[ID,1]==2?
408.     (GetArrayElement[MaintAct2Data[ID],1,4]):
409.     PrefMaintAct[ID,1]==1?
410.     (GetArrayElement[MaintAct1Data[ID],1,4]):
411.     (GetArrayElement[TransProbData[ID],1,4])';
412.     ONRELEASE E4 ASSIGN ActProbData 2 0 '
413.     PrefMaintAct[ID,2]==2?
414.     (GetArrayElement[MaintAct2Data[ID],2,0]):
415.     PrefMaintAct[ID,2]==1?
416.     (GetArrayElement[MaintAct1Data[ID],2,0]):
417.     (GetArrayElement[TransProbData[ID],2,0])';
418.     ONRELEASE E4 ASSIGN ActProbData 2 1 '
```

```
419.     PrefMaintAct[ID,2]==2?
420.     (GetArrayElement[MaintAct2Data[ID],2,1]):
421.     PrefMaintAct[ID,2]==1?
422.     (GetArrayElement[MaintAct1Data[ID],2,1]):
423.     (GetArrayElement[TransProbData[ID],2,1])';
424.     ONRELEASE E4 ASSIGN ActProbData 2 2 '
425.     PrefMaintAct[ID,2]==2?
426.     (GetArrayElement[MaintAct2Data[ID],2,2]):
427.     PrefMaintAct[ID,2]==1?
428.     (GetArrayElement[MaintAct1Data[ID],2,2]):
429.     (GetArrayElement[TransProbData[ID],2,2])';
430.     ONRELEASE E4 ASSIGN ActProbData 2 3 '
431.     PrefMaintAct[ID,2]==2?
432.     (GetArrayElement[MaintAct2Data[ID],2,3]):
433.     PrefMaintAct[ID,2]==1?
434.     (GetArrayElement[MaintAct1Data[ID],2,3]):
435.     (GetArrayElement[TransProbData[ID],2,3])';
436.     ONRELEASE E4 ASSIGN ActProbData 2 4 '
437.     PrefMaintAct[ID,2]==2?
```

```
438.     (GetArrayElement[MaintAct2Data[ID],2,4]):
439.     PrefMaintAct[ID,2]==1?
440.     (GetArrayElement[MaintAct1Data[ID],2,4]):
441.     (GetArrayElement[TransProbData[ID],2,4])';
442.     ONRELEASE E4 ASSIGN ActProbData 3 0 '
443.     PrefMaintAct[ID,3]==2?
444.     (GetArrayElement[MaintAct2Data[ID],3,0]):
445.     PrefMaintAct[ID,3]==1?
446.     (GetArrayElement[MaintAct1Data[ID],3,0]):
447.     (GetArrayElement[TransProbData[ID],3,0])';
448.     ONRELEASE E4 ASSIGN ActProbData 3 1 '
449.     PrefMaintAct[ID,3]==2?
450.     (GetArrayElement[MaintAct2Data[ID],3,1]):
451.     PrefMaintAct[ID,3]==1?
452.     (GetArrayElement[MaintAct1Data[ID],3,1]):
453.     (GetArrayElement[TransProbData[ID],3,1])';
454.     ONRELEASE E4 ASSIGN ActProbData 3 2 '
455.     PrefMaintAct[ID,3]==2?
456.     (GetArrayElement[MaintAct2Data[ID],3,2]):
```

```
457.     PrefMaintAct[ID,3]==1?
458.     (GetArrayElement[MaintAct1Data[ID],3,2]):
459.     (GetArrayElement[TransProbData[ID],3,2])';
460.     ONRELEASE E4 ASSIGN ActProbData 3 3 '
461.     PrefMaintAct[ID,3]==2?
462.     (GetArrayElement[MaintAct2Data[ID],3,3]):
463.     PrefMaintAct[ID,3]==1?
464.     (GetArrayElement[MaintAct1Data[ID],3,3]):
465.     (GetArrayElement[TransProbData[ID],3,3])';
466.     ONRELEASE E4 ASSIGN ActProbData 3 4 '
467.     PrefMaintAct[ID,3]==2?
468.     (GetArrayElement[MaintAct2Data[ID],3,4]):
469.     PrefMaintAct[ID,3]==1?
470.     (GetArrayElement[MaintAct1Data[ID],3,4]):
471.     (GetArrayElement[TransProbData[ID],3,4])';
472.     ONRELEASE E4 ASSIGN ActProbData 4 0 '
473.     PrefMaintAct[ID,3]==2?
474.     (GetArrayElement[MaintAct2Data[ID],4,0]):
475.     PrefMaintAct[ID,4]==1?
```

```
476.     (GetArrayElement[MaintAct1Data[ID],4,0]):
477.     (GetArrayElement[TransProbData[ID],4,0])';
478.     ONRELEASE E4 ASSIGN ActProbData 4 1 '
479.     PrefMaintAct[ID,3]==2?
480.     (GetArrayElement[MaintAct2Data[ID],4,1]):
481.     PrefMaintAct[ID,4]==1?
482.     (GetArrayElement[MaintAct1Data[ID],4,1]):
483.     (GetArrayElement[TransProbData[ID],4,1])';
484.     ONRELEASE E4 ASSIGN ActProbData 4 2 '
485.     PrefMaintAct[ID,3]==2?
486.     (GetArrayElement[MaintAct2Data[ID],4,2]):
487.     PrefMaintAct[ID,4]==1?
488.     (GetArrayElement[MaintAct1Data[ID],4,2]):
489.     (GetArrayElement[TransProbData[ID],4,2])';
490.     ONRELEASE E4 ASSIGN ActProbData 4 3 '
491.     PrefMaintAct[ID,3]==2?
492.     (GetArrayElement[MaintAct2Data[ID],4,3]):
493.     PrefMaintAct[ID,4]==1?
494.     (GetArrayElement[MaintAct1Data[ID],4,3]):
```

```
495.     (GetArrayElement[TransProbData[ID],4,3])';

496.     ONRELEASE E4 ASSIGN ActProbData 4 4 '

497.     PrefMaintAct[ID,3]==2?

498.     (GetArrayElement[MaintAct2Data[ID],4,4]):

499.     PrefMaintAct[ID,4]==1?

500.     (GetArrayElement[MaintAct1Data[ID],4,4]):

501.     (GetArrayElement[TransProbData[ID],4,4])';

502.     Update the state information of the element
503.     ONRELEASE E4 ASSIGN State1 '

                 S1*ActProbData[0,0]+
                 S2*ActProbData[1,0]+
                 S3*ActProbData[2,0]+
                 S4*ActProbData[3,0]+
                 S5*ActProbData[4,0]';

504.     ONRELEASE E4 ASSIGN State2 '

                 S1*ActProbData[0,1]+
                 S2*ActProbData[1,1]+
                 S3*ActProbData[2,1]+
                 S4*ActProbData[3,1]+
                 S5*ActProbData[4,1]';

505.     ONRELEASE E4 ASSIGN State3 '

                 S1*ActProbData[0,2]+
                 S2*ActProbData[1,2]+
                 S3*ActProbData[2,2]+
                 S4*ActProbData[3,2]+
```

```
                S5*ActProbData[4,2]';

506.    ONRELEASE E4 ASSIGN State4 '

                S1*ActProbData[0,3]+
                S2*ActProbData[1,3]+
                S3*ActProbData[2,3]+
                S4*ActProbData[3,3]+
                S5*ActProbData[4,3]';

507.    ONRELEASE E4 ASSIGN State5 '

                S1*ActProbData[0,4]+
                S2*ActProbData[1,4]+
                S3*ActProbData[2,4]+
                S4*ActProbData[3,4]+
                S5*ActProbData[4,4]';

508.    /Update the future state information based on the preferred maintenance activities
509.    ONRELEASE E4 ASSIGN FState1 '

                State1*ActProbData[0,0]+
                State2*ActProbData[1,0]+
                State3*ActProbData[2,0]+
                State4*ActProbData[3,0]+
                State5*ActProbData[4,0]';

510.    ONRELEASE E4 ASSIGN FState2 '

                State1*ActProbData[0,1]+
                State2*ActProbData[1,1]+
                State3*ActProbData[2,1]+
                State4*ActProbData[3,1]+
                State5*ActProbData[4,1]';

511.    ONRELEASE E4 ASSIGN FState3 '
```

```
                State1*ActProbData[0,2]+
                State2*ActProbData[1,2]+
                State3*ActProbData[2,2]+
                State4*ActProbData[3,2]+
                State5*ActProbData[4,2]';

512.    ONRELEASE E4 ASSIGN FState4 '

                State1*ActProbData[0,3]+
                State2*ActProbData[1,3]+
                State3*ActProbData[2,3]+
                State4*ActProbData[3,3]+
                State5*ActProbData[4,3]';

513.    ONRELEASE E4 ASSIGN FState5 '

                State1*ActProbData[0,4]+
                State2*ActProbData[1,4]+
                State3*ActProbData[2,4]+
                State4*ActProbData[3,4]+
                State5*ActProbData[4,4]';


514.    /*****************************************************************************************

515.    /* Entry of resources into RprdElements


516.    /*****************************************************************************************

517.    /* Statements to assist in the definition of attributes of AssembleElems and its related
   links

518.    FILTER RprdBridges Bridge 1;

519.    FILTER RprdElems Element
   'RprdBridges.HasCursor?(Source==RprdBridges.BrID):(Source==B7.BrID)';
```

```
520.     FILTEREXP RprdBridges 'RprdElements.RprdElems.Count==TotElems';

521.     /Check if all the elements in the bridge have been repaired
522.     VARPROP Bridge FnshdRepair 'RprdElements.RprdElems.Count==TotElems';

523.     /Variable to match elements and the corresponding bridges
524.     VARPROP Element FnshdBridge 'Source==AssembleElems.Bridge.BrID';

525.     /*****************************************************************************

526.     /* Startup of AssembleElems

527.     ENOUGH                  B7 'InWrkBridges.RprdBridges.Count';

528.     DRAWUNTIL         B7 'AssembleElems.Bridge.Count';

529.     DRAWWHERE         B7 'FnshdRepair';

530.     DRAWUNTIL         E5 'RprdElements.FnshdBridge.AveVal==0';

531.     DRAWWHERE         E5 'FnshdBridge';

532.     *****************************************************************************

533.     /* Termination of AssembleElems

534.     RELEASEAMT        IC2 1;

535.     *****************************************************************************

536.     /* Assembly of resources in AsmRprdBridges

537.     /Update the properties of the bridge
```

```
538.     ONASSEMBLY AsmRprdBridges ASSIGN LastRepairedTm SimTime;
539.     ONASSEMBLY AsmRprdBridges ASSIGN RepairCost '

540.     AsmRprdBridges.Element.RepairCst.SumVal';

541.     ONASSEMBLY AsmRprdBridges ASSIGN HI '

542.     AsmRprdBridges.Element.TEV.SumVal!=0?
      (AsmRprdBridges.Element.CEV.SumVal/

543.     AsmRprdBridges.Element.TEV.SumVal*100):0';

544.     ONASSEMBLY AsmRprdBridges ASSIGN FHI '

545.     AsmRprdBridges.Element.TEV.SumVal!=0?
      (AsmRprdBridges.Element.FEV.SumVal/

546.     AsmRprdBridges.Element.TEV.SumVal*100):0';


547.     /*****************************************************************************

548.     /* Startup of OneYear

549.     PRIORITY          OneYear '300';

550.     DURATION          OneYear '1';

551.     /*****************************************************************************

552.     /* Termination of OneYear

553.     RELEASEAMT C5 1;

554.     /*****************************************************************************

555.     /* Entry of resources into YrCntr
```

```
556.     /*****************************************************************************
557.     /* Entry of resources into DeteriorateCntr
558.     /*****************************************************************************
559.     /* Startup of UpdateBridges
560.     PRIORITY UpdateBridges 200;
561.     ENOUGH B10        1;
562.     DRAWWHERE         B10 'SimTime-LastRepairedTm>=1';
563.     DRAWUNTIL         B10 0;
564.     /*****************************************************************************
565.     /* Termination of UpdateBridges
566.     /*****************************************************************************
567.     /* Entry of resources into Brdgs
568.     /*****************************************************************************
569.     /* Startup of SeperateBridges
570.     PRIORITY SeperateBridges 150;
571.     DRAWUNTIL         B12 'SeperateBridges.Bridge.Count';
572.     /*****************************************************************************
573.     /* Termination of SeperateBridges
574.     /*****************************************************************************
575.     /* Release of disassembled components in DisassemBridge
```

```
576.     ONRELEASE E7 ASSIGN S1 State1;

577.     ONRELEASE E7 ASSIGN S2 State2;

578.     ONRELEASE E7 ASSIGN S3 State3;

579.     ONRELEASE E7 ASSIGN S4 State4;

580.     ONRELEASE E7 ASSIGN S5 State5;

581.     /Update the transition probabilities
582.     ONRELEASE E7 ASSIGN ActProbData 0 0 '

583.     PrefMaintAct[ID,0]==2?
584.     (GetArrayElement[MaintAct2Data[ID],0,0]):

585.     PrefMaintAct[ID,0]==1?

586.     (GetArrayElement[MaintAct1Data[ID],0,0]):

587.     (GetArrayElement[TransProbData[ID],0,0])';

588.     ONRELEASE E7 ASSIGN ActProbData 0 1 '

589.     PrefMaintAct[ID,0]==2?

590.     (GetArrayElement[MaintAct2Data[ID],0,1]):

591.     PrefMaintAct[ID,0]==1?

592.     (GetArrayElement[MaintAct1Data[ID],0,1]):

593.     (GetArrayElement[TransProbData[ID],0,1])';

594.     ONRELEASE E7 ASSIGN ActProbData 0 2 '

595.     PrefMaintAct[ID,0]==2?
```

```
596.     (GetArrayElement[MaintAct2Data[ID],0,2]):
597.     PrefMaintAct[ID,0]==1?
598.     (GetArrayElement[MaintAct1Data[ID],0,2]):
599.     (GetArrayElement[TransProbData[ID],0,2])';
600.     ONRELEASE E7 ASSIGN ActProbData 0 3 '
601.     PrefMaintAct[ID,0]==2?
602.     (GetArrayElement[MaintAct2Data[ID],0,3]):
603.     PrefMaintAct[ID,0]==1?
604.     (GetArrayElement[MaintAct1Data[ID],0,3]):
605.     (GetArrayElement[TransProbData[ID],0,3])';
606.     ONRELEASE E7 ASSIGN ActProbData 0 4 '
607.     PrefMaintAct[ID,0]==2?
608.     (GetArrayElement[MaintAct2Data[ID],0,4]):
609.     PrefMaintAct[ID,0]==1?
610.     (GetArrayElement[MaintAct1Data[ID],0,4]):
611.     (GetArrayElement[TransProbData[ID],0,4])';
612.     ONRELEASE E7 ASSIGN ActProbData 1 0 '
613.     PrefMaintAct[ID,1]==2?
614.     (GetArrayElement[MaintAct2Data[ID],1,0]):
```

```
615.     PrefMaintAct[ID,1]==1?
616.     (GetArrayElement[MaintAct1Data[ID],1,0]):
617.     (GetArrayElement[TransProbData[ID],1,0])';
618.     ONRELEASE E7 ASSIGN ActProbData 1 1 '
619.     PrefMaintAct[ID,1]==2?
620.     (GetArrayElement[MaintAct2Data[ID],1,1]):
621.     PrefMaintAct[ID,1]==1?
622.     (GetArrayElement[MaintAct1Data[ID],1,1]):
623.     (GetArrayElement[TransProbData[ID],1,1])';
624.     ONRELEASE E7 ASSIGN ActProbData 1 2 '
625.     PrefMaintAct[ID,1]==2?
626.     (GetArrayElement[MaintAct2Data[ID],1,2]):
627.     PrefMaintAct[ID,1]==1?
628.     (GetArrayElement[MaintAct1Data[ID],1,2]):
629.     (GetArrayElement[TransProbData[ID],1,2])';
630.     ONRELEASE E7 ASSIGN ActProbData 1 3 '
631.     PrefMaintAct[ID,1]==2?
632.     (GetArrayElement[MaintAct2Data[ID],1,3]):
633.     PrefMaintAct[ID,1]==1?
```

```
634.     (GetArrayElement[MaintAct1Data[ID],1,3]):
635.     (GetArrayElement[TransProbData[ID],1,3])';
636.     ONRELEASE E7 ASSIGN ActProbData 1 4 '
637.     PrefMaintAct[ID,1]==2?
638.     (GetArrayElement[MaintAct2Data[ID],1,4]):
639.     PrefMaintAct[ID,1]==1?
640.     (GetArrayElement[MaintAct1Data[ID],1,4]):
641.     (GetArrayElement[TransProbData[ID],1,4])';
642.     ONRELEASE E7 ASSIGN ActProbData 2 0 '
643.     PrefMaintAct[ID,2]==2?
644.     (GetArrayElement[MaintAct2Data[ID],2,0]):
645.     PrefMaintAct[ID,2]==1?
646.     (GetArrayElement[MaintAct1Data[ID],2,0]):
647.     (GetArrayElement[TransProbData[ID],2,0])';
648.     ONRELEASE E7 ASSIGN ActProbData 2 1 '
649.     PrefMaintAct[ID,2]==2?
650.     (GetArrayElement[MaintAct2Data[ID],2,1]):
651.     PrefMaintAct[ID,2]==1?
652.     (GetArrayElement[MaintAct1Data[ID],2,1]):
```

```
653.     (GetArrayElement[TransProbData[ID],2,1])';
654.     ONRELEASE E7 ASSIGN ActProbData 2 2 '
655.     PrefMaintAct[ID,2]==2?
656.     (GetArrayElement[MaintAct2Data[ID],2,2]):
657.     PrefMaintAct[ID,2]==1?
658.     (GetArrayElement[MaintAct1Data[ID],2,2]):
659.     (GetArrayElement[TransProbData[ID],2,2])';
660.     ONRELEASE E7 ASSIGN ActProbData 2 3 '
661.     PrefMaintAct[ID,2]==2?
662.     (GetArrayElement[MaintAct2Data[ID],2,3]):
663.     PrefMaintAct[ID,2]==1?
664.     (GetArrayElement[MaintAct1Data[ID],2,3]):
665.     (GetArrayElement[TransProbData[ID],2,3])';
666.     ONRELEASE E7 ASSIGN ActProbData 2 4 '
667.     PrefMaintAct[ID,2]==2?
668.     (GetArrayElement[MaintAct2Data[ID],2,4]):
669.     PrefMaintAct[ID,2]==1?
670.     (GetArrayElement[MaintAct1Data[ID],2,4]):
671.     (GetArrayElement[TransProbData[ID],2,4])';
```

```
672.     ONRELEASE E7 ASSIGN ActProbData 3 0 '
673.     PrefMaintAct[ID,3]==2?
674.     (GetArrayElement[MaintAct2Data[ID],3,0]):
675.     PrefMaintAct[ID,3]==1?
676.     (GetArrayElement[MaintAct1Data[ID],3,0]):
677.     (GetArrayElement[TransProbData[ID],3,0])';
678.     ONRELEASE E7 ASSIGN ActProbData 3 1 '
679.     PrefMaintAct[ID,3]==2?
680.     (GetArrayElement[MaintAct2Data[ID],3,1]):
681.     PrefMaintAct[ID,3]==1?
682.     (GetArrayElement[MaintAct1Data[ID],3,1]):
683.     (GetArrayElement[TransProbData[ID],3,1])';
684.     ONRELEASE E7 ASSIGN ActProbData 3 2 '
685.     PrefMaintAct[ID,3]==2?
686.     (GetArrayElement[MaintAct2Data[ID],3,2]):
687.     PrefMaintAct[ID,3]==1?
688.     (GetArrayElement[MaintAct1Data[ID],3,2]):
689.     (GetArrayElement[TransProbData[ID],3,2])';
690.     ONRELEASE E7 ASSIGN ActProbData 3 3 '
```

```
691.     PrefMaintAct[ID,3]==2?
692.     (GetArrayElement[MaintAct2Data[ID],3,3]):
693.     PrefMaintAct[ID,3]==1?
694.     (GetArrayElement[MaintAct1Data[ID],3,3]):
695.     (GetArrayElement[TransProbData[ID],3,3])';
696.     ONRELEASE E7 ASSIGN ActProbData 3 4 '
697.     PrefMaintAct[ID,3]==2?
698.     (GetArrayElement[MaintAct2Data[ID],3,4]):
699.     PrefMaintAct[ID,3]==1?
700.     (GetArrayElement[MaintAct1Data[ID],3,4]):
701.     (GetArrayElement[TransProbData[ID],3,4])';
702.     ONRELEASE E7 ASSIGN ActProbData 4 0 '
703.     PrefMaintAct[ID,3]==2?
704.     (GetArrayElement[MaintAct2Data[ID],4,0]):
705.     PrefMaintAct[ID,4]==1?
706.     (GetArrayElement[MaintAct1Data[ID],4,0]):
707.     (GetArrayElement[TransProbData[ID],4,0])';
708.     ONRELEASE E7 ASSIGN ActProbData 4 1 '
709.     PrefMaintAct[ID,3]==2?
```

```
710.     (GetArrayElement[MaintAct2Data[ID],4,1]):
711.     PrefMaintAct[ID,4]==1?
712.     (GetArrayElement[MaintAct1Data[ID],4,1]):
713.     (GetArrayElement[TransProbData[ID],4,1])';
714.     ONRELEASE E7 ASSIGN ActProbData 4 2 '
715.     PrefMaintAct[ID,3]==2?
716.     (GetArrayElement[MaintAct2Data[ID],4,2]):
717.     PrefMaintAct[ID,4]==1?
718.     (GetArrayElement[MaintAct1Data[ID],4,2]):
719.     (GetArrayElement[TransProbData[ID],4,2])';
720.     ONRELEASE E7 ASSIGN ActProbData 4 3 '
721.     PrefMaintAct[ID,3]==2?
722.     (GetArrayElement[MaintAct2Data[ID],4,3]):
723.     PrefMaintAct[ID,4]==1?
724.     (GetArrayElement[MaintAct1Data[ID],4,3]):
725.     (GetArrayElement[TransProbData[ID],4,3])';
726.     ONRELEASE E7 ASSIGN ActProbData 4 4 '
727.     PrefMaintAct[ID,3]==2?
728.     (GetArrayElement[MaintAct2Data[ID],4,4]):
```

```
729.     PrefMaintAct[ID,4]==1?

730.     (GetArrayElement[MaintAct1Data[ID],4,4]):

731.     (GetArrayElement[TransProbData[ID],4,4])';

732.     /Update the condition state information of the element
733.     ONRELEASE E7 ASSIGN State1 '

             S1*GetArrayElement[TransProbData[ID],0,0]+
             S2*GetArrayElement[TransProbData[ID],1,0]+
             S3*GetArrayElement[TransProbData[ID],2,0]+
             S4*GetArrayElement[TransProbData[ID],3,0]+
             S5*GetArrayElement[TransProbData[ID],4,0]';

734.     ONRELEASE E7 ASSIGN State2 '

             S1*GetArrayElement[TransProbData[ID],0,1]+
             S2*GetArrayElement[TransProbData[ID],1,1]+
             S3*GetArrayElement[TransProbData[ID],2,1]+
             S4*GetArrayElement[TransProbData[ID],3,1]+
             S5*GetArrayElement[TransProbData[ID],4,1]';

735.     ONRELEASE E7 ASSIGN State3 '

             S1*GetArrayElement[TransProbData[ID],0,2]+
             S2*GetArrayElement[TransProbData[ID],1,2]+
             S3*GetArrayElement[TransProbData[ID],2,2]+
             S4*GetArrayElement[TransProbData[ID],3,2]+
             S5*GetArrayElement[TransProbData[ID],4,2]';

736.     ONRELEASE E7 ASSIGN State4 '

             S1*GetArrayElement[TransProbData[ID],0,3]+
             S2*GetArrayElement[TransProbData[ID],1,3]+
             S3*GetArrayElement[TransProbData[ID],2,3]+
             S4*GetArrayElement[TransProbData[ID],3,3]+
```

```
            S5*GetArrayElement[TransProbData[ID],4,3]';

737.    ONRELEASE E7 ASSIGN State5 '

            S1*GetArrayElement[TransProbData[ID],0,4]+
            S2*GetArrayElement[TransProbData[ID],1,4]+
            S3*GetArrayElement[TransProbData[ID],2,4]+
            S4*GetArrayElement[TransProbData[ID],3,4]+
            S5*GetArrayElement[TransProbData[ID],4,4]';

738.    /Update the future state information of the element
739.    ONRELEASE E7 ASSIGN FState1 '
                State1*ActProbData[0,0]+
                State2*ActProbData[1,0]+
                State3*ActProbData[2,0]+
                State4*ActProbData[3,0]+
                State5*ActProbData[4,0]';

740.    ONRELEASE E7 ASSIGN FState2 '
                State1*ActProbData[0,1]+
                State2*ActProbData[1,1]+
                State3*ActProbData[2,1]+
                State4*ActProbData[3,1]+
                State5*ActProbData[4,1]';

741.    ONRELEASE E7 ASSIGN FState3 '
                State1*ActProbData[0,2]+
                State2*ActProbData[1,2]+
                State3*ActProbData[2,2]+
                State4*ActProbData[3,2]+
                State5*ActProbData[4,2]';


742.    ONRELEASE E7 ASSIGN FState4 '
                State1*ActProbData[0,3]+
                State2*ActProbData[1,3]+
                State3*ActProbData[2,3]+
                State4*ActProbData[3,3]+
```

```
                State5*ActProbData[4,3]';

743.    ONRELEASE E7 ASSIGN FState5 '
                State1*ActProbData[0,4]+
                State2*ActProbData[1,4]+
                State3*ActProbData[2,4]+
                State4*ActProbData[3,4]+
                State5*ActProbData[4,4]';


744.    /*************************************************************************************

745.    /* Assembly of resources in AssemBridge

746.    /Updating the condition information of the bridge
747.    ONASSEMBLY AssemBridge ASSIGN RepairCost '

748.    AssemBridge.Element.RepairCst.SumVal';

749.    ONASSEMBLY AssemBridge ASSIGN HI '

     AssemBridge.Element.TEV.SumVal!=0?(
750.    (AssemBridge.Element.CEV.SumVal/

751.    AssemBridge.Element.TEV.SumVal)*100):0';
752.    ONASSEMBLY AssemBridge ASSIGN FHI '

     AssemBridge.Element.TEV.SumVal!=0?(
753.    (AssemBridge.Element.FEV.SumVal/

754.    AssemBridge.Element.TEV.SumVal)*100):0';



755.    /*************************************************************************************

756.    /* Statements to assist in the definition of attributes of RepairBridges and its related
   links
```

```
757.     /*****************************************************************************
758.     /* Startup of RepairBridges
759.     FILTER BestExc Bridge 'NonDelayedHI==BridgeNetwork.NonDelayedHI.MaxVal & NonDelayedHI!=-
   100';
760.     SEMAPHORE RepairBridges ' (BridgeNetwork.CurCount+InWrkBridges.CurCount==Br__Count) ';
761.     PRIORITY          RepairBridges 20;
762.     ENOUGH B4 'BridgeNetwork.BestExc.Count & BridgeNetwork.BestExc.RepairCost.AveVal<=Funds';
763.     DRAWWHERE B4 'NonDelayedHI==BridgeNetwork.NonDelayedHI.MaxVal & NonDelayedHI!=-100 &
   RepairCost<=Funds';
764.     DRAWUNTIL B4 RepairBridges.Bridge.Count;
765.     DRAWORDER B4 -NonDelayedHI;
766.     ONDRAW                 B4 ASSIGN Funds Funds-RepairCost;
767.     *****************************************************************************
768.     /* Termination of RepairBridges
769.     *****************************************************************************
770.     /* Entry of resources into ElementsToCreat
771.     /Enter the information of the elements upon creation
772.     ONENTRY ElementsToCreat ASSIGN ID El__INFO[ResNum-1,0];
773.     ONENTRY ElementsToCreat ASSIGN Qty El__INFO[ResNum-1,2]+El__INFO[ResNum-
   1,4]+El__INFO[ResNum-1,6]+El__INFO[ResNum-1,8]+El__INFO[ResNum-1,10];
```

```
774.     ONENTRY ElementsToCreat ASSIGN State1 El__INFO[ResNum-1,1]/100;

775.     ONENTRY ElementsToCreat ASSIGN State2 El__INFO[ResNum-1,3]/100;

776.     ONENTRY ElementsToCreat ASSIGN State3 El__INFO[ResNum-1,5]/100;

777.     ONENTRY ElementsToCreat ASSIGN State4 El__INFO[ResNum-1,7]/100;

778.     ONENTRY ElementsToCreat ASSIGN State5 El__INFO[ResNum-1,9]/100;

779.     ONENTRY ElementsToCreat ASSIGN Source El__INFO[ResNum-1,11];

780.     ONENTRY ElementsToCreat ASSIGN statecnt StateCnt[ID];


781.     /*****************************************************************************

782.     /* Statements to assist in the definition of attributes of Report and its related links

783.     /Statistical collector for getting the information about the bridges
784.     BINCOLLECTOR HICollector 100 0 100;


785.     ENOUGH                  R3 '1';
786.     ENOUGH                  R1 1;
787.     DRAWUNTIL         R1 '0';
788.     ONDRAW R1 COLLECT HICollector HI;
789.     DRAWUNTIL         R3 '0';

790.     ONDRAW R3 COLLECT HICollector HI;

791.     *****************************************************************************

792.     /* Startup of Report


793.     /Outputs the appropriate information to the screen every year of simulation time
```

```
794.     PRINT StdOutput
   "Year\t\tAverageHI\t\tExcellent\t\tGood\t\tFair\t\tPoor\t\tVPoor\tFundsAllocated\tFundsLft\n";
795.     PRIORITY          Report '100';

796.     BEFOREDRAWS       Report CALL Reset[HICollector];
797.     ONSTART Report PRINT StdOutput "%4.0f %15.2f%15.0f%12.0f%12.0f%12.0f%12.0f\t"
798.     SimTime
799.     HICollector.AveVal
800.     HICollector.nSamples-HitsAtOrBelowBin[HICollector,Excellent]
801.     HitsAtOrBelowBin[HICollector,Excellent]-HitsAtOrBelowBin[HICollector,Good]
802.     HitsAtOrBelowBin[HICollector,Good]-HitsAtOrBelowBin[HICollector,Fair]
803.     HitsAtOrBelowBin[HICollector,Fair]-HitsAtOrBelowBin[HICollector,Poor]
804.     HitsAtOrBelowBin[HICollector,Poor];
805.     ONSTART Report PRINT StdOutput "%12.0f%12.0f\n" (SimTime!=0?FUNDSMTRX[SimTime-1]:0)
   Funds;

806.     /*************************************************************************************

807.     /* Termination of Report

808.     /Restore the bridges back in the network
809.     RELEASEWHERE      R2 'QueueFrm==BridgeNetwork';

810.     RELEASEWHERE      R4 'QueueFrm==InWrkBridges';


811.     /Restore the funds at the beginning of every simulation year
812.     ONEND                   Report ASSIGN Funds FUNDSMTRX[SimTime];


813.     *************************************************************************************

814.     /* Entry of resources into RCntr

815.     *************************************************************************************

816.     /* Initialization of Queues, Running the Simulation, Presenting Results
```

```
817.     /Create the bridges

818.     INIT BridgesToCreate Br__Count;

819.     /Create the elements
820.     INIT ElementsToCreat El__Count;


821.     INIT YrCntr 1;
822.     INIT RCntr 1;

823.     /Maximum number of bridges which can be worked on simultaneously
824.     INIT BridgeCount MaxBridgeCount;


825.     SIMULATEUNTIL SimTime>YrsToSimulate;
826.     /REPORT;
```

APPENDIX E

Listing Of Dll Code

```
1. //  ADD-ON DLL to Stroboscope Simulation Software
2. //
3. //  Programmer:    Srinath Devulapali
4. //  OS:            Windows 2000 Professional
5. //  System:        Pentium III 500, 128 MB Memory
6. //  Compiler:      Visual C++ 6.0, Service Pack 4
7. //  Last modified: March 27th, 2002
8. //
9. ////This is an add-on to Stroboscope which enables it to
10.      //read data from predifined files
11.      // The list of Stroboscope supported commands are
12.      // ReadElementData()
13.      //CloseInputFile()
14.      //IsEOF()
15.      // ReadElementData(double dFileIdentifier);
16.      // GetBridgeInfo(double dFileIdentifier);
17.      // GetElementInfo(double dFileIdentifier);
18.      // GetStateCount(double dFileIdentifier);
19.      // GetFailureCost(double dFileIdentifier);
20.      // GetPolicyInfo(double dFileIdentifier);

21.      //Assumptions: The InputFiles are present in the working
   directory
22.      //If not the full path of the files needs to be sepcified
23.      //The InputFiles are in the Pontis Data Interchange format
24.      //This format is based on the PDI files generated by
25.      //PONTIS software Version 4.
26.      //Five Arrays called TransProbData, MaintAct1Data,
   MaintAct2Data
27.      // FailrCostData StateCnt have been defined in the
   Stroboscope model
28.      //calling this DLL
29.      //The maximum number of brisges supported is 10000 and is
   defined by MaxBridge
30.      //The maximum length of the Bridge ID supported is given by
   MaxIDLength=50


31.      #include<stdio.h>
32.      #include<string.h>
33.      #include<fstream.h>
34.      #include<afxwin.h>
35.      #include<afxcoll.h>
36.      #include<afxtempl.h>
37.      //#include "s:\win32app\strobos\sdk\strobosc.h"
38.      //#include <strobosc.h>
39.      #include "C:\Program Files\strobosc\SDK\strobosc.h"
40.      #include<io.h>
```

```
41.      //Function prototypes for the functions registered by the
   AddOn

42.      double _stdcall OpenInputFile(double InputFileName);
43.      double _stdcall CloseInputFile(double InputFileName);
44.      double _stdcall IsEOF(double FileIdentifier);

45.      double _stdcall ReadElementData(double dFileIdentifier);
46.      double _stdcall GetBridgeInfo(double dFileIdentifier);
47.      double _stdcall GetElementInfo(double dFileIdentifier);
48.      double _stdcall GetStateCount(double dFileIdentifier);
49.      double _stdcall GetFailureCost(double dFileIdentifier);
50.      double _stdcall GetPolicyInfo(double dFileIdentifier);

51.      int _stdcall DiscardLine(const char* szArguments);


52.      //defining the global variables
53.      //ifstream* fstreampointer;
54.      char szScratch[MAX_STATEMENT_LENGTH];
55.      char szScratch2[MAX_STATEMENT_LENGTH];


56.      //Class for converting a Double to an ifstream pointer and
   vice versa
57.      //

58.      class CDoubleAndIfstreamPtr
59.      {
60.      private:
61.      union
62.      {     double m_d;
63.      struct
64.      {     ifstream* m_ptr;
65.      int m_checkVal;
66.      };
67.      };
68.      public:
69.      CDoubleAndIfstreamPtr(double d)
70.      :     m_d(d)
71.      {}

72.      CDoubleAndIfstreamPtr(ifstream* ptr)
73.      :     m_ptr(ptr)
74.      {}

75.      operator double () const {return m_d;}
76.      operator ifstream* () const{return m_ptr;}
77.      };

78.      double glb_DblTransfer;
79.      double _stdcall DblTransfer(){return glb_DblTransfer;}
```

```
80.      //Default function where all the functions and statements
   need to be registered
81.      int _stdcall StroboAddOnInit(const char* szModelName)
82.      {
83.      //Print in the error device that Stroboscope has loaded the
   Addon
84.      PrintToStdError("\nInputFile Add-On for Stroboscope");
85.      PrintToStdError("\nBy Srinath Devulapalli and Julio
   Martinez\n");

86.      //Register with Stroboscope the statements
87.      RegStatement("DISCARDLINE",DiscardLine);

88.      //Register with Stroboscope the functions
89.      RegFunction("OpenInputFile",OpenInputFile,1,TRUE);
90.      RegFunction("CloseInputFile",CloseInputFile,1,FALSE);
91.      RegFunction("IsEOF",IsEOF,1,FALSE);
92.      RegFunction("DblTransfer__",DblTransfer,0,FALSE);

93.      RegFunction("ReadElementData",ReadElementData,1,FALSE);
94.      RegFunction("GetBridgeInfo",GetBridgeInfo,1,FALSE);
95.      RegFunction("GetElementInfo",GetElementInfo,1,FALSE);
96.      RegFunction("GetStateCount",GetStateCount,1,FALSE);
97.      RegFunction("GetFailureCost",GetFailureCost,1,FALSE);
98.      RegFunction("GetPolicyInfo",GetPolicyInfo,1,FALSE);

99.      return true;
100.     }//end StroboAddOnInit

101.     //Class for managing the open input files
102.     //Maintains a list of all open input files.
103.     //Closes any open input files using the destructor

104.     class ManageFilePointers
105.     {

106.     private:
107.     ifstream** FileStreamArray;   //Array of open file pointers
108.     int count;

109.     public:
110.     ManageFilePointers();
111.     ~ManageFilePointers();
112.     ifstream* AddFile(double InputFileName);
113.     double CloseFile(double InputFileName);
114.     int ReturnPosition(ifstream* fstreampointer);

115.     } FPointerArray;

116.     //Default constructor ManageFilePointers
117.     //Parameters:     None
118.     //Calls:          None
```

```
119.     //CalledBy:       None
120.     //Pre:                  None
121.     //Post:                 The object is initialized
122.     ManageFilePointers::ManageFilePointers()
123.     {
124.     FileStreamArray=NULL;
125.     count=0;
126.     };

127.     //Default destructor
128.     //Parameters:     None
129.     //Calls:          None
130.     //CalledBy:       None
131.     //Pre:                  ManageFilePointers object exists
132.     //Post:                 The object has been dereferenced
   properly

133.     ManageFilePointers::~ManageFilePointers()
134.     {
135.     if(FileStreamArray!=NULL)
136.     {
137.     //Close the open files
138.     for(int i=0;i<count;++i)
139.     (FileStreamArray[i])->close();

140.     //dereference objects
141.     delete[] FileStreamArray;
142.     FileStreamArray=NULL;
143.     count=0;
144.     }
145.     };


146.     //ManageFilePointers::AddFile(double InputFileName)
147.     //Parameters:     InputFileName double value of file name
   passed from Stroboscope
148.     //Calls:          None
149.     //CalledBy:       OpenInputFile
150.     //Pre:                  ManageFilePointers object exists.
151.     //Post:                 The Inputfile has been opened and
   added to the array

152.     ifstream* ManageFilePointers::AddFile(double InputFileName)
153.     {
154.     strncpy(szScratch,ConvertDoubleToString(InputFileName),MAX_
   STATEMENT_LENGTH);

155.     ifstream* fstreampointer = new ifstream;

156.     //Open the file and add its pointer to the list of open
   files
157.     (*fstreampointer).open(szScratch,ios::nocreate|ios::in);
158.     if ((*fstreampointer).fail())
159.     {
```

```
160.     if(strlen(szScratch)==(MAX_STATEMENT_LENGTH-1))
161.     PrintToStdError("\nToo Long a file name.Specify a file name
   shorter than 4096 characters");
162.     else
163.     PrintToStdError("\nInput File not found");
164.     return NULL;

165.     };

166.     ifstream** temp=new ifstream*[count+1];

167.     if(temp==NULL)
168.     {
169.     PrintToStdError("\nToo many open input files. Unable to
   allocate memory");
170.     return NULL;
171.     }

172.     //Add the new file to the array
173.     for(int i=0;i<count;++i)
174.     temp[i]=FileStreamArray[i];
175.     if(FileStreamArray!=NULL)
176.     delete[] FileStreamArray;
177.     FileStreamArray=temp;
178.     temp[count]=fstreampointer;
179.     count++;
180.     return fstreampointer;

181.     };


182.     //ManageFilePointers::ReturnPosition(ifstream*
   fstreampointer)
183.     //Returns the position of the file pointer in the array if
   the
184.     //file has been opened. -1 otherwise
185.     //Parameters:           fstreampointer
186.     //
187.     int ManageFilePointers::ReturnPosition(ifstream*
   fstreampointer)
188.     {
189.     int position=-1;
190.     for(int i=0;i<count;++i)
191.     if(FileStreamArray[i]==fstreampointer)
192.     position=i;
193.     return position;
194.     };


195.     //ManageFilePointers::CloseFile()
196.     //Parameters:     InputFileName double value of file
   pointer passed by Stroboscope
197.     //Calls:          None
198.     //CalledBy:       CloseInputFile
```

```
199.     //Pre:                  ManageFilePointers object exists.
200.     //Post:                 The Inputfile has been closed and
   removed from the array

201.     double ManageFilePointers::CloseFile(double InputFileName)
202.     {
203.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(InputFileName);
204.     int position=ReturnPosition(fstreampointer);

205.     //check if file is open
206.     if(position==-1)
207.     {
208.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
209.     StrbMathError("CloseInputFile",STR_DOMAIN);

210.     return false;
211.     };

212.     //close the file
213.     fstreampointer->close();

214.     //Remove its pointer from the array
215.     ifstream** temp=new ifstream*[count-1];
216.     if(temp==NULL)
217.     return false;
218.     for(int i=0;i<position;++i)
219.     temp[i]=FileStreamArray[i];
220.     for(i=position;i<count;++i)
221.     temp[i]=FileStreamArray[i+1];
222.     delete[] FileStreamArray;
223.     FileStreamArray=temp;
224.     count--;
225.     return true;

226.     };

227.     //The function registered with Stroboscope as OpenInputFile
228.     //Takes the name of the file as INPUT and returns the
   fstream pointer
229.     //after typecasting it as a double. To use the pointer
   again need to
230.     //type cast the double back to ifstream*
231.     //Parameter:            InputFileName     Stroboscope name
   of the input file
232.     //Returns:              A double which when typecast to
   ifstream gives the
233.     //                            ifstream pointer to the input
   file
234.     //Calls:                ManageFilePointers::AddFile


235.     double _stdcall OpenInputFile(double InputFileName)
```

```
236.     {

237.     ifstream*
   fstreampointer=FPointerArray.AddFile(InputFileName);

238.     if(fstreampointer==NULL)
239.     {
240.     StrbMathError("OpenInputFile",STR_DOMAIN);
241.     return false;
242.     }
243.     return (CDoubleAndIfstreamPtr)fstreampointer;

244.     }//end OPENINPUTFILE



245.     //The function registered with Stroboscope as
   CloseInputFile

246.     double _stdcall CloseInputFile(double InputFileName)
247.     {
248.     double status=FPointerArray.CloseFile(InputFileName);

249.     if(status==0)
250.     {
251.     StrbMathError("CloseInputFile",STR_DOMAIN);
252.     return false;
253.     }
254.     return true;
255.     }


256.     //The function registered with Stroboscoope as ISEOF
257.     //Parameters:           FileIdentifier    When typecast to
   ifstream gives
258.     //                                                    a
   pointer to the file
259.     //Returns:              true if EOF false otherwise

260.     double _stdcall IsEOF(double dFileIdentifier)
261.     {
262.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(dFileIdentifier);

263.     int position=FPointerArray.ReturnPosition(fstreampointer);

264.     //Check if file has not been opened
265.     if(position==-1)
266.     {
267.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
268.     StrbMathError("ReadData",STR_DOMAIN);
```

```
269.     return false;
270.     };

271.     //ifstream*
   fstreampointer=CDoubleAndIfstreamPtr(dFileIdentifier);

272.     (*fstreampointer).eatwhite();
273.     if((*fstreampointer).peek()==EOF)
274.     return 1;
275.     return 0;
276.     }


277.     //The statement registered with Stroboscope as DISCARDLINE
278.     //Function:             Removes a line from the input file
   stream
279.     //                           Can be used to ignore
   comments
280.     //
281.     int _stdcall DiscardLine(const char* szArguments)
282.     {

283.     char *szFileIdentifier,*Scratch;
284.     strncpy(szScratch,szArguments,MAX_STATEMENT_LENGTH);
285.     Scratch=szScratch;
286.     szFileIdentifier = ExtractArgument(Scratch);
287.     if(!strlen(szFileIdentifier))
288.     {
289.     PrintToStdError("\nMissing File Identifier");
290.     return false;
291.     }

292.     //ifstream*
   fstreampointer=CDoubleAndIfstreamPtr(EvaluateExpression(szFileIde
   ntifier));
293.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(EvaluateExpression(szFileIdentif
   ier));

294.     int position=FPointerArray.ReturnPosition(fstreampointer);
295.     if(position==-1)
296.     {
297.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
298.     StrbMathError("ReadData",STR_DOMAIN);

299.     return false;
300.     };

301.     (*fstreampointer).ignore(MAX_STATEMENT_LENGTH,'\n');
302.     return 1;
303.     }
```

```
304.     //The function registered with Stroboscope as
   ReadElementData
305.     //Parameters:           None
306.     //Assumptions:          The data files are present in the
   current directory.
307.     //                            The element data files have a
   '.elm' extension
308.     //Returns:              If the operation was succesful or
   not

309.     double _stdcall ReadElementData(double dFileIdentifier)
310.     {

311.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(dFileIdentifier);

312.     int position=FPointerArray.ReturnPosition(fstreampointer);

313.     //check if file has not been opened
314.     if(position==-1)
315.     {
316.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
317.     StrbMathError("ReadData",STR_DOMAIN);

318.     return false;
319.     };


320.     //The temporary arrays in which the values read from the
   files are stored.
321.     //Once the file is read the values are sent to Stroboscope
   using the
322.     //appropriate Stroboscope statements

323.     double (*TransProbData)[5][6];
324.     double (*MaintAct1Data)[5][8];
325.     double (*MaintAct2Data)[5][8];

326.     TransProbData=new double[800][5][6];
327.     MaintAct1Data=new double[800][5][8];
328.     MaintAct2Data=new double[800][5][8];

329.     //Initialize the arrays
330.     for(int i=0;i<800;++i)
331.     for(int j=0;j<5;++j)
332.     for(int k=0;k<6;++k)
333.     TransProbData[i][j][k]=0;

334.     for(i=0;i<800;++i)
335.     for(int j=0;j<5;++j)
336.     for(int k=0;k<8;++k)
337.     MaintAct1Data[i][j][k]=0;
```

```
338.     for(i=0;i<800;++i)
339.     for(int j=0;j<5;++j)
340.     for(int k=0;k<8;++k)
341.     MaintAct2Data[i][j][k]=0;


342.     (*fstreampointer).eatwhite();
343.     char sValue[MAX_STATEMENT_LENGTH];
344.     (*fstreampointer)>>sValue;
345.     //fstreampointer->getline(sValue,MAX_STATEMENT_LENGTH);

346.     //check for the beginning of the data table
347.     while(strcmp(sValue,"ACTMODLS")!=0)
348.     (*fstreampointer)>>sValue;

349.     while(!IsEOF(dFileIdentifier))
350.     {
351.     double elemID,stateID,actionID,envID;
352.     double prob1,prob2,prob3,prob4,prob5,cost,optimumfrac;
353.     char ch;

354.     //Read the appropriate data from the data file

355.     (*fstreampointer)>>sValue;
356.     (*fstreampointer)>>ch;
357.     //(*fstreampointer)>>ch;
358.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
359.     elemID=atof(sValue);
360.     //(*fstreampointer)>>elemID;
361.     (*fstreampointer)>>ch;
362.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
363.     stateID=atof(sValue);
364.     (*fstreampointer)>>ch;
365.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
366.     actionID=atof(sValue);
367.     (*fstreampointer)>>ch;
368.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
369.     envID=atof(sValue);

370.     (*fstreampointer)>>ch;
371.     (*fstreampointer)>>sValue;
372.     (*fstreampointer)>>sValue;
373.     (*fstreampointer)>>ch;

374.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
375.     prob1=atof(sValue);
376.     (*fstreampointer)>>ch;
377.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
378.     prob2=atof(sValue);
379.     (*fstreampointer)>>ch;
380.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
381.     prob3=atof(sValue);
382.     (*fstreampointer)>>ch;
```

```
383.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
384.     prob4=atof(sValue);
385.     (*fstreampointer)>>ch;
386.     fstreampointer>getline(sValue,MAX_STATEMENT_LENGTH,'"');
387.     prob5=atof(sValue);


388.     (*fstreampointer)>>ch;
389.     fstreampointer->getline(sValue,MAX_STATEMENT_LENGTH,'"');
390.     cost=atof(sValue);
391.     (*fstreampointer)>>sValue;
392.     (*fstreampointer)>>ch;
393.     fstreampointer->getline(sValue,MAX_STATEMENT_LENGTH,'"');
394.     optimumfrac=atof(sValue);
395.     (*fstreampointer)>>ch;
396.     fstreampointer->getline(sValue,MAX_STATEMENT_LENGTH,'"');
397.     double ltcost=atof(sValue);


398.     //Now place this data in the appropriate array
399.     if(envID==2)
400.     {
401.     /*if(optimumfrac>0.8)
402.     {
403.     sprintf(szScratch,"ASSIGN PrefMaintAct [%d] [%d]
   %d",int(elemID),int(stateID),int(actionID));
404.     ExecuteStatement(szScratch);
405.     }*/

406.     if(actionID==2)
407.     {
408.     MaintAct2Data[int(elemID)][int(stateID)-1][0]=prob1/100;
409.     MaintAct2Data[int(elemID)][int(stateID)-1][1]=prob2/100;
410.     MaintAct2Data[int(elemID)][int(stateID)-1][2]=prob3/100;
411.     MaintAct2Data[int(elemID)][int(stateID)-1][3]=prob4/100;
412.     MaintAct2Data[int(elemID)][int(stateID)-1][4]=prob5/100;
413.     //if(cost>0)
414.     MaintAct2Data[int(elemID)][int(stateID)-1][5]=cost;
415.     //if(ltcost>0)
416.     //    MaintAct2Data[int(elemID)][int(stateID)-1][6]=ltcost;
417.     }
418.     else if(actionID==1)
419.     {
420.     MaintAct1Data[int(elemID)][int(stateID)-1][0]=prob1/100;
421.     MaintAct1Data[int(elemID)][int(stateID)-1][1]=prob2/100;
422.     MaintAct1Data[int(elemID)][int(stateID)-1][2]=prob3/100;
423.     MaintAct1Data[int(elemID)][int(stateID)-1][3]=prob4/100;
424.     MaintAct1Data[int(elemID)][int(stateID)-1][4]=prob5/100;
425.     //if(cost>0)
426.     MaintAct1Data[int(elemID)][int(stateID)-1][5]=cost;
427.     //if(ltcost>0)
428.     //    MaintAct1Data[int(elemID)][int(stateID)-1][6]=ltcost;
429.     }
430.     else if(actionID==0)
```

```
431.     {
432.     TransProbData[int(elemID)][int(stateID)-1][0]=prob1/100;
433.     TransProbData[int(elemID)][int(stateID)-1][1]=prob2/100;
434.     TransProbData[int(elemID)][int(stateID)-1][2]=prob3/100;
435.     TransProbData[int(elemID)][int(stateID)-1][3]=prob4/100;
436.     TransProbData[int(elemID)][int(stateID)-1][4]=prob5/100;
437.     //if(ltcost>0)
438.     //    TransProbData[int(elemID)][int(stateID)-1][5]=ltcost;

439.     }
440.     else
441.     { }

442.     };

443.     (*fstreampointer)>>sValue;

444.     //skip until beginning of next table is found
445.     while(strcmp(sValue,"ACTMODLS")!=0 &&
   !IsEOF(dFileIdentifier))
446.     (*fstreampointer)>>sValue;

447.     };

448.     //Write appropriate Stroboscope statements to transfer the
   data read
449.     //into Stroboscope
450.     //For TransProbData
451.     for(i=0;i<800;++i)
452.     {
453.     //Create the 2-d array
454.     sprintf(szScratch,"ARRAY trans__%d 5 6",i);
455.     ExecuteStatement(szScratch);
456.     for(int j=0;j<5;++j)
457.     for(int k=0;k<=5;++k)
458.     {
459.     sprintf(szScratch,"ASSIGN trans__%d %d %d
   %f",i,j,k,TransProbData[i][j][k]);
460.     ExecuteStatement(szScratch);

461.     }
462.     sprintf(szScratch,"ASSIGN TransProbData %d trans__%d",i,i);
463.     ExecuteStatement(szScratch);

464.     };

465.     //For MaintAct1Data
466.     for(i=0;i<800;++i)
467.     {
468.     //create the appropriate 2-d array
469.     sprintf(szScratch,"ARRAY maint1__%d 5 7",i);
470.     ExecuteStatement(szScratch);
471.     for(int j=0;j<5;++j)
472.     for(int k=0;k<7;++k)
```

```
473.     {
474.     sprintf(szScratch,"ASSIGN maint1__%d %d %d
   %f",i,j,k,MaintAct1Data[i][j][k]);
475.     ExecuteStatement(szScratch);

476.     }
477.     sprintf(szScratch,"ASSIGN MaintAct1Data %d
   maint1__%d",i,i);
478.     ExecuteStatement(szScratch);
479.     };

480.     //For MaintAct2Data
481.     for(i=0;i<800;++i)
482.     {
483.     sprintf(szScratch,"ARRAY maint2__%d 5 7",i);
484.     ExecuteStatement(szScratch);
485.     for(int j=0;j<5;++j)
486.     for(int k=0;k<7;++k)
487.     {
488.     sprintf(szScratch,"ASSIGN maint2__%d %d %d
   %f",i,j,k,MaintAct2Data[i][j][k]);
489.     ExecuteStatement(szScratch);
490.     }
491.     sprintf(szScratch,"ASSIGN MaintAct2Data %d
   maint2__%d",i,i);
492.     ExecuteStatement(szScratch);
493.     };


494.     delete []TransProbData;
495.     delete []MaintAct1Data;
496.     delete []MaintAct2Data;

497.     return true;
498.     }

499.     //Counts the number of bridges and creates them in
   Stroboscope

500.     const MaxBridges=10000;//defines the maximum number of
   brisges supported
501.     const MaxIDLength=50;
502.     int Brcount=0;    //keeps track of the number of bridges
   created
503.     char BrID[MaxBridges][MaxIDLength]; //storest the Bridge
   ID's

504.     double _stdcall GetBridgeInfo(double dFileIdentifier)
505.     {

506.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(dFileIdentifier);

507.     int position=FPointerArray.ReturnPosition(fstreampointer);
```

```
508.     //Check if file has not been opened
509.     if(position==-1)
510.     {
511.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
512.     StrbMathError("ReadData",STR_DOMAIN);

513.     return false;
514.     };


515.     (*fstreampointer).eatwhite();
516.     char sValue[MAX_STATEMENT_LENGTH];
517.     (*fstreampointer)>>sValue;

518.     while(strcmp(sValue,"BRIDGE")!=0)
519.     (*fstreampointer)>>sValue;

520.     while(!IsEOF(dFileIdentifier))
521.     {
522.     //char temp[100];
523.     char ch;
524.     //fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
525.     (*fstreampointer)>>ch;
526.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
527.     strcpy(BrID[Brcount],szScratch);
528.     Brcount++;
529.     (*fstreampointer)>>szScratch;
530.     while(!IsEOF(dFileIdentifier) &&
   (strcmp(szScratch,"BRIDGE")!=0))
531.     (*fstreampointer)>>szScratch;
532.     }

533.     sprintf(szScratch,"ARRAY BR__INFO %d",Brcount);
534.     ExecuteStatement(szScratch);

535.     //write the bridge information to Stroboscope
536.     for(int i=0;i<Brcount;++i)
537.     {
538.     sprintf(szScratch,"ASSIGN BR__INFO %d \"%s\"",i,BrID[i]);
539.     ExecuteStatement(szScratch);
540.     };

541.     sprintf(szScratch,"SAVEVALUE Br__Count %d",Brcount);
542.     ExecuteStatement(szScratch);

543.     return 1;

544.     }


545.     //Extracts the element information from the input file
```

```
546.     //Returns 1 if succesful, - if not
547.     //Parameters:     dFileIdentifies - the Stroboscope
   identifier of the file
548.     double _stdcall GetElementInfo(double dFileIdentifier)
549.     {

550.     const MaxElements=500000;
551.     const MaxBridgeIDLt=50;
552.     double (*ElementInfo)[11];
553.     ElementInfo = new double[MaxElements][11];
554.     char (*BridgeID)[MaxBridgeIDLt] = new
   char[MaxElements][MaxBridgeIDLt];
555.     int count=0;

556.     int pos=0;

557.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(dFileIdentifier);

558.     int position=FPointerArray.ReturnPosition(fstreampointer);

559.     //Check if the file is not open
560.     if(position==-1)
561.     {
562.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
563.     StrbMathError("ReadData",STR_DOMAIN);

564.     return false;
565.     };

566.     (*fstreampointer).eatwhite();
567.     char sValue[MAX_STATEMENT_LENGTH];
568.     (*fstreampointer)>>sValue;

569.     while(strcmp(sValue,"ELEMINSP")!=0)
570.     (*fstreampointer)>>sValue;

571.     while(!IsEOF(dFileIdentifier))
572.     {
573.     double state;
574.     //double elementID,pctstate1,qtystate1,pctstate2,qtystate2;
575.     //double
   pctstate3,qtystate3,pctstate4,qtystate4,pctstate5,qtystate5;
576.     char ch;

577.     (*fstreampointer)>>ch;
578.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
579.     strcpy(BridgeID[count],szScratch);

580.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');
```

```
581.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');
582.     (*fstreampointer)>>ch;
583.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
584.     ElementInfo[count][0]=atof(szScratch);

585.     (*fstreampointer)>>ch;
586.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
587.     state=atof(szScratch);

588.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');
589.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');
590.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');
591.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');
592.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');
593.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'\n');

594.     (*fstreampointer)>>ch;
595.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
596.     ElementInfo[count][1]=atof(szScratch);

597.     (*fstreampointer)>>ch;
598.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
599.     ElementInfo[count][2]=atof(szScratch);

600.     (*fstreampointer)>>ch;
601.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
602.     ElementInfo[count][3]=atof(szScratch);

603.     (*fstreampointer)>>ch;
604.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
605.     ElementInfo[count][4]=atof(szScratch);

606.     (*fstreampointer)>>ch;
607.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
608.     ElementInfo[count][5]=atof(szScratch);

609.     (*fstreampointer)>>ch;
610.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
611.     ElementInfo[count][6]=atof(szScratch);
```

```
612.     (*fstreampointer)>>ch;
613.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
614.     ElementInfo[count][7]=atof(szScratch);

615.     (*fstreampointer)>>ch;
616.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
617.     ElementInfo[count][8]=atof(szScratch);

618.     (*fstreampointer)>>ch;
619.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
620.     ElementInfo[count][9]=atof(szScratch);

621.     (*fstreampointer)>>ch;
622.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
623.     ElementInfo[count][10]=atof(szScratch);

624.     count++;

625.     (*fstreampointer)>>sValue;
626.     while(strcmp(sValue,"ELEMINSP")!=0 &&
   !IsEOF(dFileIdentifier))
627.     (*fstreampointer)>>sValue;
628.     }

629.     sprintf(szScratch,"ARRAY El__INFO %d 13",count);
630.     ExecuteStatement(szScratch);

631.     for(int i=0;i<count;++i)
632.     {
633.     for(int j=0;j<11;++j)
634.     {
635.     sprintf(szScratch,"ASSIGN El__INFO %d %d
   %f",i,j,ElementInfo[i][j]);
636.     ExecuteStatement(szScratch);
637.     }
638.     sprintf(szScratch,"ASSIGN El__INFO %d 11
   \"%s\"",i,BridgeID[i]);
639.     ExecuteStatement(szScratch);

640.     };


641.     delete []ElementInfo;
642.     delete []BridgeID;

643.     sprintf(szScratch,"SAVEVALUE El__Count %d",count);
644.     ExecuteStatement(szScratch);

645.     return 1;
```

```
646.     }

647.     //Extracts the state information for each element type
648.     //Parameters:     The Stroboscope identifier for the input
   file
649.     //
650.     double _stdcall GetStateCount(double dFileIdentifier)
651.     {
652.     int StateCount[800];

653.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(dFileIdentifier);

654.     int position=FPointerArray.ReturnPosition(fstreampointer);
655.     if(position==-1)
656.     {
657.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
658.     StrbMathError("ReadData",STR_DOMAIN);

659.     return false;
660.     };

661.     (*fstreampointer).eatwhite();
662.     char sValue[MAX_STATEMENT_LENGTH];
663.     (*fstreampointer)>>sValue;

664.     while(strcmp(sValue,"ELEMDEFS")!=0)
665.     (*fstreampointer)>>sValue;

666.     while(!IsEOF(dFileIdentifier))
667.     {
668.     int ID;
669.     char ch;
670.     //(*fstreampointer)>>sValue;
671.     (*fstreampointer)>>ch;
672.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
673.     ID=atof(szScratch);

674.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
675.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
676.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
677.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
678.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
679.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
680.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
681.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
682.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
683.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
684.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
685.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
```

```
686.     (*fstreampointer)>>ch;
687.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
688.     StateCount[ID]=atof(szScratch);

689.     (*fstreampointer)>>sValue;
690.     while(strcmp(sValue,"ELEMDEFS")!=0 &&
   !IsEOF(dFileIdentifier))
691.     (*fstreampointer)>>sValue;

692.     };

693.     for(int i=0;i<800;++i)
694.     {
695.     sprintf(szScratch,"ASSIGN StateCnt %d
   %d",i,StateCount[i]>0?StateCount[i]:0);
696.     ExecuteStatement(szScratch);
697.     }

698.     return 1;
699.     }


700.     //Extracts the failure cost information from the input file
701.     //Parameters:     dFileIdentifier - The Stroboscope
   identifier of the input file
702.     //Returns:  true if succesful, false otherwise
703.     //
704.     double _stdcall GetFailureCost(double dFileIdentifier)
705.     {
706.     double FailureCost[800]={0};

707.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(dFileIdentifier);

708.     int position=FPointerArray.ReturnPosition(fstreampointer);
709.     if(position==-1)
710.     {
711.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
712.     StrbMathError("ReadData",STR_DOMAIN);

713.     return false;
714.     };

715.     (*fstreampointer).eatwhite();
716.     char sValue[MAX_STATEMENT_LENGTH];
717.     (*fstreampointer)>>sValue;

718.     //while(strcmp(sValue,"EXPCNDUC")!=0)
719.     while(strcmp(sValue,"CONDUMDL")!=0 &&
   !IsEOF(dFileIdentifier))
720.     (*fstreampointer)>>sValue;
```

```
721.     while(!IsEOF(dFileIdentifier))
722.     {
723.     char ch;
724.     int ID;

725.     //(*fstreampointer)>>sValue;
726.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
727.     (*fstreampointer)>>ch;
728.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
729.     ID=atof(szScratch);

730.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
731.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
732.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
733.     (*fstreampointer)>>ch;
734.     fstreampointer-
   >getline(szScratch,MAX_STATEMENT_LENGTH,'"');
735.     if(atof(szScratch)>0)
736.     FailureCost[ID]=atof(szScratch);

737.     (*fstreampointer)>>sValue;
738.     while(strcmp(sValue,"CONDUMDL")!=0 &&
   !IsEOF(dFileIdentifier))
739.     (*fstreampointer)>>sValue;

740.     }

741.     for(int i=0;i<800;++i)
742.     {
743.     sprintf(szScratch,"ASSIGN FailrCostData %d
   %f",i,FailureCost[i]>0?FailureCost[i]:0);
744.     ExecuteStatement(szScratch);

745.     }

746.     return 1;

747.     }

748.     //double _stdcall GetPolicyInfo(double dFileIdentifier)
749.     //Reads the policy information from the input file
750.     //Parameters:           dFileIdentifier - The file pointer
   from which
751.     //                              the formatted policy
   information is read
752.     //Pre:                          dFileIdentifier points to an
   open Policy file
753.     //                              The format of the policy file
   is ACTID '\t' Action Action Action Action Action
754.     //                              The specified action
   information must be in actmodls file
755.     double _stdcall GetPolicyInfo(double dFileIdentifier)
756.     {
```

```
757.     ifstream* fstreampointer=
   (ifstream*)CDoubleAndIfstreamPtr(dFileIdentifier);

758.     int position=FPointerArray.ReturnPosition(fstreampointer);
759.     if(position==-1)
760.     {
761.     PrintToStdError("\nNot previously Open Input File. Use
   OpenInputFile");
762.     StrbMathError("ReadData",STR_DOMAIN);

763.     return false;
764.     };

765.     (*fstreampointer).eatwhite();
766.     char ch=(*fstreampointer).peek();

767.     while( !IsEOF(dFileIdentifier) )
768.     {
769.     while(ch=='/')
770.     {
771.     fstreampointer->getline(szScratch,MAX_STATEMENT_LENGTH);
772.     fstreampointer->eatwhite();
773.     ch=fstreampointer->peek();
774.     };

775.     int ID,action1,action2,action3,action4,action5;
776.     (*fstreampointer)>>ID>>action1>>action2>>action3>>action4>>
   action5;
777.     sprintf(szScratch,"ASSIGN PrefMaintAct %d 0
   %d",ID,action1);
778.     ExecuteStatement(szScratch);
779.     sprintf(szScratch,"ASSIGN PrefMaintAct %d 1
   %d",ID,action2);
780.     ExecuteStatement(szScratch);
781.     sprintf(szScratch,"ASSIGN PrefMaintAct %d 2
   %d",ID,action3);
782.     ExecuteStatement(szScratch);
783.     sprintf(szScratch,"ASSIGN PrefMaintAct %d 3
   %d",ID,action4);
784.     ExecuteStatement(szScratch);
785.     sprintf(szScratch,"ASSIGN PrefMaintAct %d 4
   %d",ID,action5);
786.     ExecuteStatement(szScratch);

787.     fstreampointer->eatwhite();
788.     ch=fstreampointer->peek();

789.     };

790.     return true;

791.     };
```

APPENDIX F

Vdot Element Level Policies

	Unit Costs
"Bare Concrete Deck"	0
12 1 0 Do Nothing	89.02
12 1 1 Add a protective system	
12 2 0 Do Nothing	0
12 2 1 Repair spalls and delaminations	284.17
12 2 2 Add a protective system	73.63
12 3 0 Do Nothing	0
12 3 1 Repair spalls and delaminations	307.74
12 3 2 Repair spalls and delaminations and add a protective system on entire deck	129.17
12 4 0 Do Nothing	0
12 4 1 Repair spalls and delaminations	158.01
12 4 2 Repair spalls and delaminations and add a protective system on entire deck	151.23
12 5 0 Do Nothing	0
12 5 1 Repair spalls and delaminations and add/or a protective system on entire deck	242.19
12 5 2 Replace deck	301.5
---	---
"Unp Conc Deck/AC Ovl"	
13 1 0 Do Nothing	0
13 2 0 Do Nothing	
13 2 1 Repair potholes and substrate	337.99
13 3 0 Do Nothing	0
13 3 1 Repair potholes and substrate	180.3
13 3 2 Replace overlay and repair substrate	226.04
13 4 0 Do Nothing	0
13 4 1 Repair potholes and substrate	96.55
13 4 2 Repair substrate and replace overlay	186.43
13 5 0 Do Nothing	0
13 5 1 Repair substrate and replace overlay	67.27
13 5 2 Replace deck	331.1
---	---
"P Conc Deck/Rigid Ov"	
22 1 0 Do Nothing	0

22 2 0 Do Nothing	0
22 2 1 Repair spalls and delaminations	29.82
22 3 0 Do Nothing	0
22 3 1 Repair spalls and delaminations	68.89
22 4 0 Do Nothing	0
22 4 1 Repair spalls and delaminations	95.58
22 4 2 Replace overlay	115.39
22 5 0 Do Nothing	0
22 5 1 Replace overlay	80.73
22 5 2 Replace deck	346.6
----------	----------
"Conc Deck/Coatd Bars"	
26 1 0 Do Nothing	0
26 1 1 Add a protective system	180
26 2 0 Do Nothing	0
26 2 1 Repair spalls and delaminations	52.53
26 2 2 Add a protective system	180
26 3 0 Do Nothing	0
26 3 1 Repair spalls and delaminations	30.68
26 3 2 Repair spalls and delaminations and add a protective system on entire deck	79.65
26 4 0 Do Nothing	0
26 4 1 Repair spalls and delaminations	83.96
26 4 2 Repair spalls and delaminations and add a protective system on entire deck	113.77
26 5 0 Do Nothing	0
26 5 1 Repair spalls and delaminations and add/or a protective system on entire deck	108.28
26 5 2 Replace deck	187.62
----------	----------
"P/S Conc Closed Web/Box Girder"	
104 1 0 Do Nothing	0
	328.08
104 2 0 Do Nothing	0
104 2 1 Seal cracks and minor patching	82.02
	328.08
104 3 0 Do Nothing	0
104 3 1 Clean steel and patch (and/or seal)	164.04
	328.08

104 4 0 Do Nothing	0
104 4 1 Rehab unit	350
104 4 2 Replace unit	328.08
----------	----------
"Unpainted Steel Open Girder/Beam"	
106 1 0 Do Nothing	0
	108.27
	902.23
106 2 0 Do Nothing	0
106 2 1 Clean and paint	108.27
	110
106 3 0 Do Nothing	0
106 3 1 Clean and paint	108.27
	1000
106 4 0 Do Nothing	0
106 4 1 Rehab unit	108.27
106 4 2 Replace unit	1246.72
----------	----------
"Painted Steel Open Girder/Beam"	
107 1 0 Do Nothing	0
107 1 1 Surface clean	215.28
	1000
107 2 0 Do Nothing	0
107 2 1 Surface clean	215.28
107 2 2 Clean and paint	220
107 3 0 Do Nothing	0
107 3 1 Spot blast, clean, and paint	215.28
	220
107 4 0 Do Nothing	0
107 4 1 Spot blast, clean, and paint	215.28
107 4 2 Replace paint system	500
107 5 0 Do Nothing	0
107 5 1 Rehab unit	215.28
107 5 2 Replace unit	1500
----------	----------
"P/S Conc Open Girder/Beam"	
109 1 0 Do Nothing	0
	328.08

109 2 0 Do Nothing	0
109 2 1 Seal cracks and minor patching	82.02
	328.08
109 3 0 Do Nothing	0
109 3 1 Clean steel and patch (and/or seal)	164.04
	328.08
109 4 0 Do Nothing	0
109 4 1 Rehab unit	350
109 4 2 Replace unit	328.08
--- ---	--- ---
"Reinforced Conc Open Girder/Beam"	
110 1 0 Do Nothing	0
	328.08
110 2 0 Do Nothing	0
110 2 1 Seal cracks and minor patching	82.02
	328.08
110 3 0 Do Nothing	0
110 3 1 Clean rebar and patch (and/or seal)	164.04
	328.08
110 4 0 Do Nothing	0
110 4 1 Rehab unit	350
110 4 2 Replace unit	328.08
--- ---	
"Painted Steel Column or Pile Extension"	
202 1 0 Do Nothing	0
202 1 1 Surface clean	275
	2000
202 2 0 Do Nothing	0
202 2 1 Surface clean	275
202 2 2 Clean and paint	280
202 3 0 Do Nothing	0
202 3 1 Spot blast, clean, and paint	275
	280
202 4 0 Do Nothing	0
202 4 1 Spot blast, clean, and paint	275
202 4 2 Replace paint system	500

202 5 0 Do Nothing	0
202 5 1 Rehab unit	275
202 5 2 Replace unit	1000
----------	----------
"Reinforced Conc Column or Pile Extension"	
205 1 0 Do Nothing	0
	2000
205 2 0 Do Nothing	0
205 2 1 Seal cracks and minor patching	500
	2000
205 3 0 Do Nothing	0
205 3 1 Clean rebar and patch (and/or seal)	750
	2000
205 4 0 Do Nothing	0
205 4 1 Rehab unit	1000
205 4 2 Replace unit	2000
----------	----------
"Reinforced Conc Pier Wall"	
210 1 0 Do Nothing	0
	9742.78
210 2 0 Do Nothing	0
210 2 1 Seal cracks and minor patching	1075.23
	9742.78
210 3 0 Do Nothing	0
210 3 1 Clean rebar and patch (and/or seal)	1552.17
	9742.78
210 4 0 Do Nothing	0
210 4 1 Rehab unit	3857.05
210 4 2 Replace unit	9742.78
----------	----------
"Reinforced Conc Abutment"	
215 1 0 Do Nothing	0
	2560.17
215 2 0 Do Nothing	0
215 2 1 Seal cracks and minor patching	330.81
	2560.17
215 3 0 Do Nothing	0

215 3 1 Clean rebar and patch (and/or seal)	767.62
	2560.17
215 4 0 Do Nothing	0
215 4 1 Rehab unit	1700
215 4 2 Replace unit	2560.17
----------	----------
"Reinforced Conc Cap"	
234 1 0 Do Nothing	0
	328.08
234 2 0 Do Nothing	0
234 2 1 Seal cracks and minor patching	82.02
	328.08
234 3 0 Do Nothing	0
234 3 1 Clean rebar and patch (and/or seal)	164.04
	328.08
234 4 0 Do Nothing	0
234 4 1 Rehab unit	350
234 4 2 Replace unit	328.08
----------	----------
---	---
"Timber Cap"	
235 1 0 Do Nothing	0
	328.08
235 2 0 Do Nothing	0
235 2 1 Rehab and/or protect unit	16.4
	328.08
235 3 0 Do Nothing	0
235 3 1 Rehab unit	164.04
235 3 2 Replace unit	328.08
235 4 0 Do Nothing	0
235 4 1 Rehab unit	350
235 4 2 Replace unit	328.08
----------	----------
"Pourable Joint Seal"	
301 1 0 Do Nothing	0
	196.85
301 2 0 Do Nothing	0
301 2 1 Clean joint and replace seal	65.62
	196.85

301 3 0 Do Nothing	0
301 3 1 Clean joint, patch spalls, and replace seal	114.83

"Compression Joint Seal"	
302 1 0 Do Nothing	0
	328.08
302 2 0 Do Nothing	0
302 2 1 Patch/remove and reseal/clean joint	82.02
	328.08
302 3 0 Do Nothing	0
302 3 1 Replace gland and/or patch spalls	147.64
302 3 2 Replace joint	328.08
----------	----------
"Assembly Joint/Seal (modular)"	
303 1 0 Do Nothing	0
	1500
303 2 0 Do Nothing	0
303 2 1 Rehab unit	328.08
	1500
303 3 0 Do Nothing	0
303 3 1 Rehab unit	656.17
303 3 2 Replace unit	1500
----------	----------
"Open Expansion Joint"	
304 1 0 Do Nothing	0
	196.85
304 2 0 Do Nothing	0
304 2 1 Rehab unit	82.02
	196.85
304 3 0 Do Nothing	0
304 3 1 Rehab unit	147.64
304 3 2 Replace unit	196.85
----------	----------
"Elastomeric Bearing"	
310 1 0 Do Nothing	0
	325
310 2 0 Do Nothing	0
310 2 1 Reset bearings	75

	325
310 3 0 Do Nothing	0
310 3 1 Reset bearings	350
310 3 2 Replace unit and reset girders	325
----------	----------
"Moveable Bearing (roller, sliding, etc.)"	0
311 1 0 Do Nothing	150
	400
311 2 0 Do Nothing	0
311 2 1 Clean and paint or reset bearings and/or rehab supports	150
	200
311 3 0 Do Nothing	0
311 3 1 Rehab supports or bearings	150
311 3 2 Replace unit	250
----------	----------
"Enclosed/Concealed Bearing"	
313 1 0 Do Nothing	0
	150
	400
313 2 0 Do Nothing	0
313 2 1 Clean and paint or reset bearings and/or rehab supports	150
	200
313 3 0 Do Nothing	0
313 3 1 Rehab supports or bearings	150
313 3 2 Replace unit	250
----------	----------
"Reinforced Conc Approach Slab w/ or w/o AC Ovly"	
321 1 0 Do Nothing	0
	575
	1500
321 2 0 Do Nothing	0
321 2 1 Perform mudjacking operations	575
	750
321 3 0 Do Nothing	0
321 3 1 Place overlay	575
321 3 2 Replace unit	750
321 4 0 Do Nothing	0

321 4 1 Replace unit	575
	1000

"Metal Bridge Railing - Uncoated"	
330 1 0 Do Nothing	0
	16.4
	164.04
330 2 0 Do Nothing	0
330 2 1 Clean and restore coating	16.4
	50
330 3 0 Do Nothing	0
330 3 1 Clean and restore coating	16.4
330 3 2 Replace unit	75
330 4 0 Do Nothing	0
330 4 1 Rehab unit	16.4
330 4 2 Replace unit	100
----------	----------
"Reinforced Conc Bridge Railing"	
331 1 0 Do Nothing	0
	328.08
331 2 0 Do Nothing	
331 2 1 Seal cracks, minor patching	32.81
	328.08
331 3 0 Do Nothing	0
331 3 1 Clean rebar and patch (and/or seal)	82.02
	328.08
	0
331 4 0 Do Nothing	0
331 4 1 Rehab unit	164.04
331 4 2 Replace unit	328.08
----------	----------
"Other Bridge Railing"	
333 1 0 Do Nothing	0
	350
333 2 0 Do Nothing	0
333 2 1 Rehab unit	62.04
	350
333 3 0 Do Nothing	0
333 3 1 Rehab unit	164.04

333 3 2 Replace unit	350
---	---
"Metal Bridge Railing - Coated"	
334 1 0 Do Nothing	0
	16.4
	164.04
334 2 0 Do Nothing	0
334 2 1 Clean and coat	16.4
	25
334 3 0 Do Nothing	0
334 3 1 Clean and coat	16.4
	50
334 4 0 Do Nothing	0
334 4 1 Rehab unit	16.4
334 4 2 Replace unit	100
334 5 0 Do Nothing	
334 5 1 Rehab unit	
334 5 2 Replace unit	
---	---

APPENDIX G

Functions Provided By Dll

Function Name	Purpose/Use
OpenInputFile( )	Opens a file for reading data from it. Requires the name of the file to be opened as a parameter. Returns a Stroboscope handle (pointer) for extracting the data from the file
CloseInputFile( )	Closes a file which has been opened for reading. Requires the Stroboscope handle of the open file as a parameter
IsEOF( )	Checks if the end of file has been reached while reading the input file
ReadElementData( )	Reads the PONTIS element data definitions from the specified file and imports this data to Stroboscope. Assumes that the input file has been opened and the file uses PONTIS Data Interchange (PDI) format.
GetBridgeInfo( )	Extracts the bridge network information from the specified file. Assumes that the file has been opened and it uses PDI format.
GetElementInfo( )	Extracts the individual element condition information from the inspection reports. Assumes that the input file has been opened and uses PDI format for storing information
GetStateCount( )	Extracts the number of condition states defined for each element definition
GetFailureCost( )	Extracts the failure cost information for the element condition states
GetPolicyInfo( )	Extracts the element level policies to be implemented in the network from the specified file and imports this information to the model. Assumes that the file has been opened before and the format matches the format of the Policy Analysis guidelines specified.

APPENDIX H

Listing Of Policy Analysis File

```
/ "Bare Concrete Deck"
12 0 0 0 2 1
/ "Unp Conc Deck/AC Ovl"
13 0 0 0 2 1
/ "P Conc Deck/AC Ovly"
14 0 0 0 2 1
/ "P Conc Deck/Thin Ovl"
18 0 0 0 1 1
/ "P Conc Deck/Rigid Ov"
22 0 0 0 1 1
/ "Conc Deck/Coatd Bars"
26 0 0 0 2 1
/ "Conc Deck/Cathodic"
27 0 1 2 1 1
/ "Steel Deck/Open Grid"
28 0 0 2 2 1
/ "Steel Deck/Conc Grid"
29 0 0 1 1 1
/ "Steel Deck - Corrugated/Orthotropic/Etc."
30 0 0 2 1 1
/ "Timber Deck - Bare"
31 0 0 1 0 0
/ "Timber Deck - w/ AC Overlay"
32 0 1 1 0 0
/ "Concrete Slab - Bare"
38 0 1 2 1 1
/ "Concrete Slab - Unprotected w/ AC Overlay"
39 0 0 1 0 1
/ "Concrete Slab - Protected w/ AC Overlay"
40 0 0 1 1 1
/ "Concrete Slab - Protected w/ Thin Overlay"
44 0 0 0 1 1
/ "Concrete Slab - Protected w/ Rigid Overlay"
48 0 0 0 1 1
/ "Concrete Slab - Protected w/ Coated Bars"
52 0 1 2 1 1
/ "Concrete Slab - Protected w/ Cathodic System"
53 0 1 2 1 1
/ "Timber Slab"
54 0 0 1 0 0
/ "Timber Slab - w/ AC Overlay"
55 0 0 1 0 0
/ "Reinforced Concrete Sidewalk"
92 0 0 0 0 0
/ "Timber Sidewalk"
94 0 0 0 0 0
/ "Defines those open grid steel sidewalks protected"
```

```
98 0 0 0 0 0
/ "Unpainted Steel Closed Web/Box Girder"
101 1 1 1 1 0
/ "Painted Steel Closed Web/Box Girder"
102 1 1 1 1 1
/ "Aluminum Closed Girder"
103 0 0 0 0 0
/ "P/S Conc Closed Web/Box Girder"
104 0 0 1 2 0
/ "Reinforced Concrete Closed Webs/Box Girder"
105 0 1 1 2 0
/ "Unpainted Steel Open Girder/Beam"
106 0 0 1 1 0
/ "Painted Steel Open Girder/Beam"
107 0 0 0 2 1
/ "P/S Conc Open Girder/Beam"
109 0 0 1 2 0
/ "Reinforced Conc Open Girder/Beam"
110 0 0 1 1 0
/ "Timber Open Girder/Beam"
111 0 1 1 2 0
/ "Unpainted Steel Stringer"
112 1 1 1 1 0
/ "Painted Steel Stringer"
113 1 1 1 1 1
/ "P/S Conc Stringer"
115 0 1 1 2 0
/ "Reinforced Conc Stringer"
116 0 1 1 2 0
/ "Timber Stringer"
117 0 1 1 2 0
/ "Unpainted Steel Bottom Chord Thru Truss"
120 1 1 1 1 0
/ "Painted Steel Bottom Chord Thru Truss"
121 1 1 1 1 1
/ "Unpainted Steel Thru Truss (excl. bottom chord)"
125 1 1 1 1 0
/ "Painted Steel Thru Truss (excl. bottom chord)"
126 1 1 1 1 1
/ "Unpainted Steel Deck Truss"
130 1 1 1 1 0
/ "Painted Steel Deck Truss"
131 1 1 1 1 1
/ "Timber Truss/Arch"
135 0 1 1 2 0
/ "Unpainted Steel Arch"
140 1 1 1 1 0
/ "Painted Steel Arch"
141 1 1 1 1 1
/ "P/S Conc Arch"
143 0 1 1 2 0
```

```
/ "R/Conc Arch"
144 0 1 1 2 0
/ "Other Arch"
145 0 1 1 2 0
/ "Cable - Uncoated (not embedded in concrete)"
146 1 1 1 1 0
/ "Misc Cable Coated"
147 1 1 1 1 1
/ "Unpainted Steel Floor Beam"
151 1 1 1 1 0
/ "Painted Steel Floor Beam"
152 1 1 1 1 1
/ "P/S Conc Floor Beam"
154 0 1 1 2 0
/ "Reinforced Conc Floor Beam"
155 0 1 1 2 0
/ "Timber Floor Beam"
156 0 1 1 2 0
/ "Unpainted Steel Pin and/or Pin and Hanger Assembly"
160 1 1 1 1 0
/ "Painted Steel Pin and/or Pin and Hanger Assembly"
161 1 1 1 1 1
/ "Unpainted Steel Column or Pile Extension"
201 1 1 1 1 0
/ "Painted Steel Column or Pile Extension"
202 0 0 0 2 1
/ "P/S Conc Column or Pile Extension"
204 0 1 1 1 0
/ "Reinforced Conc Column or Pile Extension"
205 0 0 1 1 0
/ "Timber Column or Pile Extension"
206 0 1 1 1 0
/ "Reinforced Conc Pier Wall"
210 0 0 1 1 0
/ "Other Material Pier Wall"
211 0 1 1 1 0
/ "Reinforced Conc Abutment"
215 0 0 1 1 0
/ "Timber Abutment"
216 0 1 1 1 0
/ "Other Material Abutment"
217 0 1 1 1 0
/ "Reinforced Conc Submerged Pile Cap/Footing"
220 0 1 1 1 0
/ "Unpainted Steel Submerged Pile"
225 1 1 1 1 0
/ "P/S Conc Submerged Pile"
226 0 1 1 1 0
/ "Reinforced Conc Submerged Pile"
227 0 1 1 1 0
/ "Timber Submerged Pile"
```

```
228 0 1 1 1 0
/ "Unpainted Steel Cap"
230 1 1 1 1 0
/ "Painted Steel Cap"
231 1 1 1 1 1
/ "P/S Conc Cap"
233 0 1 1 2 0
/ "Reinforced Conc Cap"
234 0 0 1 1 0
/ "Timber Cap"
235 0 0 1 1 0
/ "Unpainted Steel Culvert"
240 0 1 1 1 0
/ "Reinforced Concrete Culvert"
241 0 1 1 1 0
/ "Timber Culvert"
242 0 1 1 1 0
/ "Other Culvert"
243 0 1 1 1 0
/ "Protected Slope"
285 0 0 0 0 0
/ "Unprotected Slope"
286 0 0 0 0 0
/ "Reinforced Concrete Wingwalls"
295 0 0 0 0 0
/ "Timber Wingwalls"
296 0 0 0 0 0
/ "Other Material Wingwalls"
297 0 0 0 0 0
/ "Strip Seal Expansion Joint"
300 0 1 2 0 0
/ "Pourable Joint Seal"
301 0 1 2 0 0
/ "Compression Joint Seal"
302 0 1 2 0 0
/ "Assembly Joint/Seal (modular)"
303 0 1 2 0 0
/ "Open Expansion Joint"
304 0 1 2 0 0
/ "Elastomeric Bearing"
310 0 0 2 0 0
/ "Moveable Bearing (roller, sliding, etc.)"
311 0 0 1 0 0
/ "Enclosed/Concealed Bearing"
312 0 0 1 0 0
/ "Fixed Bearing"
313 0 0 1 0 0
/ "Pot Bearing"
314 0 0 1 0 0
/ "Disk Bearing"
315 0 0 1 0 0
```

```
/ "P/S Concrete Approach Slab w/ or w-o/AC Ovly"
320 0 0 1 1 0
/ "Reinforced Conc Approach Slab  w/ or w/o AC Ovly"
321 0 0 0 1 0
/ "Metal Bridge Railing - Uncoated"
330 0 0 1 1 0
/ "Reinforced Conc Bridge Railing"
331 0 0 0 1 0
/ "Timber Bridge Railing"
332 0 1 1 0 0
/ "Other Bridge Railing"
333 0 0 1 0 0
/ "Metal Bridge Railing - Coated"
334 0 0 0 1 2
/ "Steel Fatigue SmFlag"
356 0 0 0 0 0
/ "Pack Rust Smart Flag"
357 0 0 0 0 0
/ "Deck Cracking SmFlag"
358 0 0 0 0 0
/ "Soffit of Concrete Deck or Slab"
359 0 0 0 0 0
/ "Settlement SmFlag"
360 0 0 0 0 0
/ "Scour Smart Flag"
361 0 0 0 0 0
/ "Traf Impact SmFlag"
362 0 0 0 0 0
/ "Section Loss SmFlag"
363 0 0 0 0 0
/ "Utilities"
701 0 0 0 0 0
/ "Drains Smart Flag"
702 0 0 0 0 0
/ "Lighting"
703 0 0 0 0 0
/ "Roadway Over Culvert"
704 0 0 0 0 0
/ "Concrete Deck Overhang Under-side"
706 0 0 0 0 0
/ "Soffit with Stay In Place Forms"
707 0 0 0 0 0
/ "Concrete slab Covered with Fill"
738 0 0 0 0 0
```

APPENDIX I

Calculation of Maintainability Index

MI=ADTS+ STTS+ DCTS+ SCTS+ SBCTS+ PCTS+ RCTS+ DATS+ TCCTS+ 3RTS+ PTS+ FPTS

Where

ADTS is Annual Daily Traffic Total Score and is given by

ADTS = ADT/500 (Annual Daily Traffic obtained from survey information)

STTS is Superstructure Type Total Score and is given by

STTS = Monolithic Score + Pin/Link Score + Fracture Critical Score + Truss Score + Timber Deck Score

$$\left.\begin{array}{l} Monolithic\ \ Score \\ Pin/Link\ \ Score \\ Fracture\ \ Critical\ \ Score \\ Truss\ \ Score \\ Timber\ \ Deck\ \ Score \end{array}\right\} = \begin{array}{l} -15\ if\ \ condition\ \ exists \\ 0\ if\ \ not \end{array}$$

DCTS is Deck Condition Total Score and is given by

*DCTS = Deck Rating*3 (Deck Rating is obtained from NBI inspection data)*

SCTS is Superstructure Condition Total Score and is given by

*SCTS = Superstructure Rating*3 (Superstructure Rating obtained from NBI)*

SBCTS is Substructure Condition Total Score and is given by

*SBCTS = Substructure Condition Rating*3 (Substructure rating obtained from NBI data)*

PCTS is P/C Score (Paint & Scour) and is given by

PCTS = Scour Score + Paint Score + Joint Score + Wearing Surface Score

$$Scour\ Score = \begin{cases} 10\ if\ \ Scour\ \ Rating = 3 \\ 0\ if\ \ Scour\ \ Rating >= 5 \\ 5\ otherwise \end{cases}$$

$$\left.\begin{array}{l} Paint\ Score \\ Joint\ Score \\ Wearing\ Surface\ Score \end{array}\right\} = \begin{cases} 10\ if\ \ Rating = C \\ 5\ if\ \ Rating = P \\ 0\ otherwise \end{cases}$$

RCTS is Repair Cost Total Score and is given by

RCTS = -Repair Cost/10,000 (Repair Cost is estimated repair cost

DATS is Deck Area Total Score and is given by

*DATS = Existing Length*Existing Width/1000*

TCCTS is Traffic Control Conditions Total Score and is given by

$$TCCTS = \begin{cases} 5\ if\ \ Traffic\ Control\ \ Conditions = G \\ 3\ if\ \ Traffic\ Control\ \ Conditions = A \\ 0\ if\ \ Traffic\ Control\ \ Conditions = P \end{cases}$$

3RTS is 3R Total Score and is given by

$$3RTCS = \begin{cases} 5\ if\ \ meets\ 3R\ condition \\ 3\ if\ \ does\ not\ meet\ traffic\ condition\ but\ \ can\ be\ converted\ \ to\ 3R \\ 0\ if\ \ cannot\ \ be\ converted\ \ to\ 3R \end{cases}$$

PTS is Posting Total Score and is given by

$$PTS = \begin{cases} 5\ if\ \ Posting\ is\ N \\ 3\ if\ \ Posting\ is\ Y\ \ and\ \ Strengthening\ Candidate\ is\ Y \\ 0\ otherwise \end{cases}$$

FPTS is Future Plans Total Score and is given by

$$FPTS = \begin{cases} 5\ if\ \ \text{Re}\ placement\ \ Planned\ \ is\ N \\ 3\ if\ \ \text{Re}\ placement\ \ Planned\ \ is\ P \\ 0\ if\ \ \text{Re}\ placement\ \ Planneed\ \ is\ U \end{cases}$$

VITA

Srinath Devulapalli was born in Machilipatnam, India on July 3rd 1979. He received his primary and secondary education at Aurobindo High school in Hyderabad. After graduating from high school in 1996, he started to pursue his baccalaureate degree in civil engineering at the Indian Institute of Technology, Madras. During the course of his undergraduate education he became interested in simulation and project management aspects of construction. After graduating in 2000, he chose to pursue his academic interests at Virginia Tech and worked towards his M.S degree. At the time of completing his thesis, he hopes to continue his academic endeavor by pursuing a master's degree in computer science.