3-29-2004

Using neural networks for goal driven simulation

Maria F. Clavijo
Florida International University

DOI: 10.25148/etd.FI14060853

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

USING NEURAL NETWORKS FOR GOAL DRIVEN SIMULATION

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

INDUSTRIAL AND SYSTEMS ENGINEERING

by

Maria F. Clavijo

2004

To: Dean Vish Prasad
College of Engineering

This thesis, written by Maria F. Clavijo, and entitled Using Neural Networks for Goal Driven Simulation, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

Kia Makki

Armando Barreto

Martha A. Centeno, Major Professor

Date of Defense: March 29, 2004

The thesis of Maria F. Clavijo is approved.

Dean Vish Prasad
College of Engineering

Dean Douglas Wartzok
University Graduate School

Florida International University, 2004

DEDICATION

To my parents, *Yolanda* and *José Clavijo*, for their unconditional support throughout these years, as it has been very difficult to be away from home. I have missed them very much. I hope my success compensates for the time we have been apart. I love you both.

To my boyfriend, *Christopher Demchalk*, for **always** being there. I could not have done it without him. I love you very much "Mi Amor".

DEDICATORIA

A mis padres, *Yolanda* y *José Clavijo*, por su apoyo incondicional durante todos estos años, ha sido muy dificil estar lejos de casa y los he extrañado mucho. Espero mi exito pague el tiempo que hemos estado distanciados. Los quiero mucho a ambos.

A mi novio, *Christopher Demchalk*, por **siempre** estar ahí. No hubiera podido conseguirlo sin él. Te amo mucho "Mi Amor".

ACKNOWLEDGMENTS

First of all, I have to thank my major professor and **dear** friend *Dr. Martha A. Centeno*. Thank you very much for being there for me all the time, anyday, anywhere. Without you this thesis would not have been as enjoyable and gratifying as it was. I will never forget you. ☺

A special thanks goes to my friends from the ARISE Lab, especially *Ramki Sundaram* and *Mansi Srivastava*. *Mansi*, thanks for your endless patience and for giving me confidence everyday. *Ramki*, thanks for the midnight music; it inspired me ☺, and for staying late at the Lab even when you had finished your thesis.

Thanks to my friends, *Laura Biaggini*, *Gertrudis Achecar* and *Melissa Carrion*, for taking me out to dinner and rescuing me, once in a while, from the Lab. Your friendship is one of the best things that have happened to me in Miami.

Last but not least, I would like to thank the committee members for their participation and valuable advice in this effort, and the ISE Department staff, especially *Nancy Urbina* and *Sylvia Sharpe*, for all those little details that made my life a lot simpler.

ABSTRACT OF THE THESIS

USING NEURAL NETWORKS FOR GOAL DRIVEN SIMULATION

by

Maria F. Clavijo

Florida International University, 2004

Miami, Florida

Professor Martha A. Centeno, Major Professor

An integration framework for Neural Networks (NN) and Goal Driven Simulation (GDS) has been designed. It offers no constraints regarding number of variables ($n>3$) and it does not have domain restrictions. The effectiveness of the framework was tested by observing the computational time required for obtaining responses and for training, and by assessing its accuracy for different scenarios. This framework has achieved the automation objective set by GDS under a shorter time frame, as it reduces the time from more than 42 hours to less than 14. A trained NN generates responses to queries almost instantaneously. However, it requires time re-building and re-training new NNs when changes are made to the system represented by the model. If these changes are rare, the payoff is worthy as this approach gives users more flexibility.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1[1]

INTRODUCTION

Goal Driven Simulation (GDS) does not seek to optimize results; rather it seeks a feasible solution, which satisfies a preset objective or goal. In many real world instances, management may not be interested in the true optimal, rather it may be interested in determining the conditions under which it can offer a predefined level of service. In response to this, research efforts have led to various approaches for Goal Driven Simulation (GDS). However, these approaches have presented various difficulties during its real world implementation. For instance, Alsugair and Chang (1994) developed a system for civil engineering called Goal Driven Simulation System, which is not applicable to other domains. Reyes (1998), Peñaloza (1999), and Jones (1999) developed a more general heuristic for both terminating and non-terminating systems regardless of domain, but it deals with only one variable. Sevimoglu (2002) extended Reyes's work to include three variables, but she found out that the process of adding new decision variables is not simple. Yet, general Goal Driven Simulation requires multivariable heuristics to be of real value.

In addition to these drawbacks, it is necessary to include in the list of challenges the complexity of the systems being studied. In complex systems the number of possible

[1] Supported in part by NASA grants NAG10-0212 and NAG010-321.

configurations is large, and the search for the best one is time consuming, expensive, and ineffective (Schewetman 2000).

This study has been based on the challenges of Goal Driven Simulation and on the possibilities offered by Neural Networks technology to overcome them, namely to facilitate the handling of *n* variables (n>3) with ease. Artificial Neural Networks (ANN) have proven effective at dealing with hard-to-define problems (Issacson 1998) and are considered masters in pattern recognition, which makes them an appealing technology to perform Goal Driven Simulation in a more efficient fashion that the current algorithmic approaches. Lee (2002) and Sevimoglu (2002) did an extensive study of the response surface of a healthcare system and of a manufacturing system. Their study clearly shows that there is a pattern, rather complex, but a pattern nonetheless that explains the relationship between the inputs and outputs (I/O) of these systems even though these systems are very distinct. The existence of similar patterns among diverse systems leads one to pursue automated means to exploit these patterns in designing new systems. In pursuing system configurations that yield a specific output, knowing precisely this relationship is a must. It is here where the Neural Networks capabilities may help significantly in recognizing the existing n-dimensional relationships found in many systems that are modeled using simulation. The ANN technology has been applied to situations where algorithmic solutions have fallen short solving mostly non-procedural type of problems. Likewise, Goal Driven Simulation applications require handling scenarios for which there are no algorithms to represent the I/O relationships.

An integration framework has been designed using a simple yet robust methodology to integrate GDS and ANN. The resulting decision support system will allow a non-expert user to make decisions with ease. An experimental prototype was created to test this integration framework, which showed satisfactory results in achieving the level of automation and transparency goals required by GDS. Experiments were done to verify the accuracy of the results as well as the computational efficiency of the methodology. Results from these experiments indicated that a GDS using NN is able to predict the parameters for a given goal set almost instantaneously. However, some time investment is needed when changes to the original system are reflected in the simulation model; in these cases the NNs have to be built new. Yet again, the process remains transparent to the user.

The remaining of Chapter 1 explains in detail the problem addressed, the goal and the specific objectives of this effort. Chapter 2 provides a literature review on Goal Driven Simulation, Neural Networks, and Neural Networks in Simulation. Chapter 3 describes the methodology used in this effort as well as the tools used. Chapter 4 illustrates the experimental prototype that was built following the proposed methodology. Chapter 5 provides details of the experimentation and its results. Lastly, Chapter 6 summarizes the results and suggests extensions to this work.

1.1 PROBLEM STATEMENT

Computer simulation has evolved over the last 40 years, thanks to the research done to improve its capabilities and to reduce its drawbacks (Kelton, et al. 1998). One of the areas in which experts have been working is Goal Driven Simulation also known as Goal Oriented Simulation. GDS can be achieved by means of two ways:

1) Embedding the simulation model within a high level programming language application, or
2) Embedding the high-level program language GDS code within the simulation model (Reyes 1998.)

Thus, far the second approach is the one that has been used the most. This has been due in part to the tendency of simulation packages to be closed, not embeddable systems. They do have connectivity windows that allow the package to control other applications. Except for a few packages, such as Silk (Kilgore 2001), commercial simulation packages cannot function as simple simulation engines. However, given the pace at which software technologies are evolving, it is reasonable to suspect that vendors of simulation packages will give customers the "engine only" option in a non-distant future.

In the meantime, various research efforts have sought to make GDS a feasible and robust approach for both, one-time simulation studies and every day simulation-based decision-making. But although these efforts have yielded valuable knowledge, there are still some challenges to meet to realize the full potential of GDS. Among these challenges are:

A) How to automate GDS?
B) How to make GDS "knowledgeable"?

First of all, GDS must be automated (Sanchez 1994); in other words, any heuristic for GDS must know how to drive the simulation model towards the goal. In one-time simulation studies, GDS automation may not be critical as the simulation expert actually looks at the outputs and is able to make an assessment of the situation and modify the inputs accordingly. But, for simulation-based decision-making, automated GDS is critical because, in most instances, the user of the simulation-based decision support system (SBDSS) is not a simulation expert; hence, he or she may not know how to change the inputs to the model to attain better results. The challenge then is: How do we automate GDS?

In the 1980's, the challenges of automation were concentrated on hardware technology and software incompatibility. Since then, hardware has evolved so much that it is no longer a significant problem. Software has also evolved to a point in which heterogeneous packages can communicate with each other. Thus, the challenges of GDS automation now reside in making the GDS heuristic knowledgeable enough, so that it can make decisions similar to those of a human expert.

A GDS heuristic must be a back-end that emulates the knowledge of a simulation expert, including statistical knowledge (Jones 1999). To recreate this knowledge, it is necessary to analyze the decision-making technique followed by the experts. In non-automated GDS, once the simulation has been executed the expert examines the outputs and determines if the goal was achieved or not. In the case when the goals are not achieved, the expert has to decide which parameters to change and by how much (Shannon and Prakash, 1990.) In automated GDS this process has been modeled by using rules that relates the system's responses with an input parameter(s) change (Umpress and Poch, 1987; Sevimoglu 2002.) These rules have been derived by empirical observation of the response surface of the system under study. For example, if the goal is to have a utilization of 70% or more, and the simulation outputs are yielding 45% utilization, the inputs that need changing are number of resources and/or system demand.

Reyes et al. (1999) stated a set of questions that must be answered in order to make GDS knowledgeable and useful in the real world. Answering these questions also leads to a better understanding of how to automate GDS (challenge A). These questions are:

1) Is the goal set feasible?
2) When should the simulation execution be halted to check the direction of the results?
3) How to assess the direction of the results?,
4) If the results are going in the wrong direction, how should the inputs be changed?, and
5) Where in the time line should the simulation be restarted?

Regarding question number 1, Reyes (1998) stated that, other than the scenarios where the set goals have some degree of overlapping in the response surface, achieving all the goals simultaneously may be infeasible. A GDS heuristic should know how to determine the feasibility of the solution. In GDS, feasibility refers to the ability of satisfying the goals of each and every one of the variables in the goal set ($n>3$) simultaneously. No formal research has been done in this regard, but Reyes (1998) and Sevimoglu (2002) have recognized that the nature of some of the most commonly used output variables require contradictory adjustments to the input set. For example, utilization of resources versus time in the system; usually, it is desirable to have a high utilization of resources and a low time in the system, but the relation between them is generally inverse.

Some of the previous research efforts have focused on improving the computational aspect of GDS by predicting if the simulation outputs would reach the targeted values without executing the entire simulation replication (Reyes 1998) or even the entire run (Jones 1999). Reyes et al. (1999) Suggested that there are three ways to determine when to halt the simulation run: 1) based on simulation length, 2) based on the number of observations, and 3) based on the correlation coefficient value. The idea behind halting the simulation run to predict if the goals would be achieved was to avoid wasting time and computational resources. Even though computers are really fast nowadays, the computational savings achieved by Reyes et al. (1999) exceed 75%, which make them significant.

Assessing the direction of the results has been explored using statistical methods such as confidence intervals and forecasting techniques. Reyes (1998) and Jones (1999) used confidence intervals around the mean of the output values and specified that if 75% of this interval does not overlap the goal range set by the user, the simulation should be stopped and the inputs parameters should be changed. Although this approach seems to work well, Sharda and Rampal (2000) have stated that ANN may be a better alternative to perform this statistical task because ANN use a variety of techniques to "deduce a result," such as, regression, discriminant analysis, logistic regression, and forecasting, which could offer an option to determine the direction of the results during the simulation execution.

Sevimoglu (2002), derived rules that relate the system response with input parameters changes. These rules allow the determination of the step size, required by the GDS technique to execute a change. In this particular aspect, ANN learning capabilities may improve GDS by deriving the rules automatically and intrinsically, without the participation of the modeler.

Question number 5 addresses the issue of where in the time line the simulation should be restarted. In theory, if the system was performing adequately at time *t*, and poorly at time *t+1*, then it should be restarted from the last known stoppage point with adequate performance, in this case *t* (Jones 1999). Reyes et al. (1999) state that a knowledgeable GDS should assist the user in the process of finding an alternative route from the new starting point. No research has been done in regards to this issue. The intent of not having to run the simulation again from time zero is based on time savings. Perhaps because the computational speed has increased so much in the last few years, the priority of this research question has declined significantly. With ANN, this question becomes irrelevant because once the ANN is trained, there is no need to re-run the model.

Existing Goal Driven Simulation approaches present a limitation in handling the search space because there are able to evaluate just a fraction of the immense range of options available (Glover et al. 1999). In this aspect, ANN may be useful because they are able to filter out solutions that are likely to perform poorly when the simulation is executed (Laguna and Marti 2002). Thus, a more extensive examination of the search space can be done in an efficient manner.

ANN mimic human neurons (Isaacson 1998); thus, they are able to learn by receiving inputs with their correspondent set of outputs. The learning capabilities of ANN offer an appealing medium to achieve the automation required by the GDS technique. ANN have proven themselves useful in various domains (See Chapter 2). The question now is *how can Artificial Neural Network facilitate and improve Goal*

Driven Simulation? To answer this question, it is necessary to examine the five questions posed by Reyes et al. (1999). In this examination, one must keep in mind that the objective of GDS is to find a set of inputs that satisfies the constraints of the system and achieves the desired set of outputs (goals). But a Neural Network asks for "inputs" to derive "outputs"; thus, for an ANN to work adequately in the context of GDS, the first thing that must happen is that the "inputs" to the ANN must be the goal (outputs), and its "outputs" must be the configuration (inputs). Therefore, the ANN has to be trained using the outputs of the simulation model as "inputs" and the inputs of the simulation model as "outputs" (Figure 1).

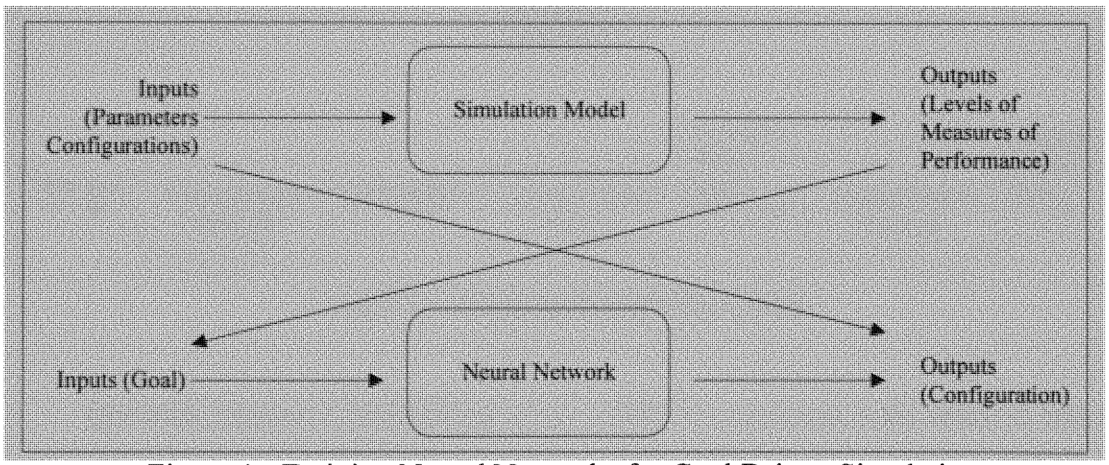

Inputs (Parameters Configurations)
Simulation Model
Outputs (Levels of Measures of Performance)
Inputs (Goal)
Neural Network
Outputs (Configuration)

Figure 1: Training Neural Networks for Goal Driven Simulation

A trained ANN possesses a minimal reaction time; thus, the "outputs" could be generated in a fraction of a second. This computational speed becomes handy when checking the feasibility of the goal set (question 1). The feasibility of it depends on the system constraints. So, it is necessary to include an Add-On subroutine written in a programming language, such as Visual Basic for Applications (VBA) or similar, to test the "outputs" against the users' requirements and the system' constraints that have been defined in advance. In case of infeasibility, one of two scenarios may occur: 1) the set of "outputs" given by the ANN differs notoriously from the values established by the user as constraints of the system, or 2) the "outputs" given by the ANN are located within a "reasonable" range from the values established by the user (Figure 2). The definition of "reasonable" is subject to discussion; however, its intention is to describe the "neighborhood" as used in optimization problems.

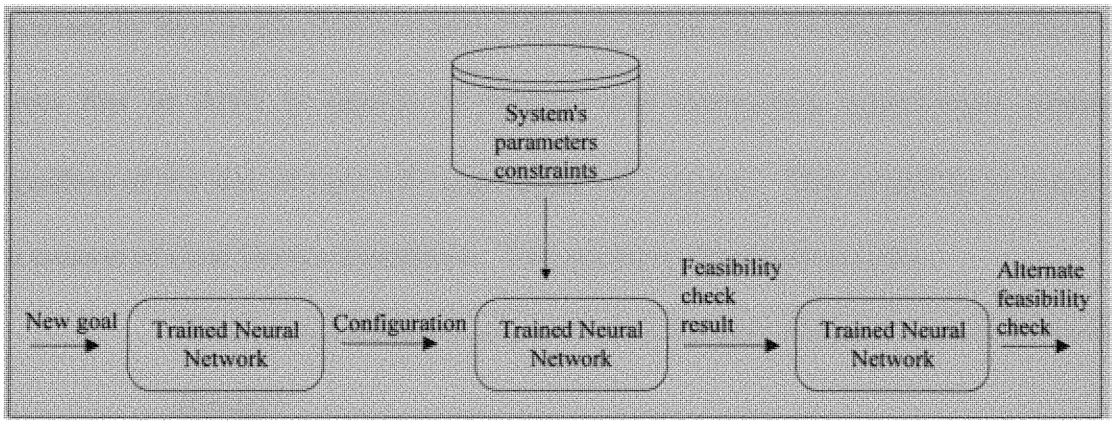


Figure 2: Alternate Feasibility Check

Question number 2 from Reyes et al. (1999) is concerned with when to halt the execution of the simulation model. Because a trained ANN is knowledgeable enough to predict "output" values just by entering a set of "input" values, running the simulation model becomes unnecessary; therefore, this question becomes irrelevant when using ANN. Similarly, question number 3 depends on the partial results obtained when the simulation is halted. If the simulation is not run, this question becomes irrelevant as well.

Although question number 4 is also related to halting the simulation run, it is still relevant even when using ANN. As it was discussed previously, once it is detected than the "outputs" are infeasible, there are two possible scenarios. Concentrating in the second scenario, where the "outputs" are located within a "reasonable" range from the original values established by the user, there is a possibility of presenting these results to the user as an alternate option, once he/she is willing to modify the current values of one or more of the system's constraints.

Question 5 has lost its relevance in the last couple of years because of the increase in computational speed, but speed is still a mandatory requirement for all of GDS efforts.

One aspect of GDS briefly mentioned by Reyes et al. (1999) and thoroughly discussed by Sevimoglu (2002) is that of the goal set having more than three variables. It seems that an ANN is capable of learning and determining Input/Output relations regardless of the size of Input ($\vec{\mathbf{I}}$) and Output ($\vec{\mathbf{O}}$) vectors. This needed to be tested in the context of Goal Driven Simulation. Two questions have high priority:

I. What would happen if one or more structural characteristics of the model change?

II. Would the knowledge acquired by the Neural Network allow an early detection of infeasibilities?

For question I, most likely it would be necessary to re-train the Neural Network by using data generated from the modified version of the model. In the research done by Sevimoglu (2002) a similar situation was presented, and she stated the high level of difficulty of re-discovering the rules that relate inputs and outputs. For automation purposes, mainly Simulation Based Decision Support Systems (SBDSS), if the ANN has to be re-trained this process must take place on-line and in a timely fashion. On-line training means that as soon as the modified simulation model generates outputs, these are fed to the ANN and the training takes place without any user intervention. For question II, it would be necessary to experiment with a trained ANN by feeding known infeasible goals and analyzing the response of the ANN.

1.2 Goal and Objectives

The goal of this research is to develop a methodology to integrate Neural Network technologies to simulation models to achieve a robust and flexible GDS framework.

To achieve the goal of this study, it was necessary to perform the following tasks:

1) Investigate Neural Networks in the context of simulation.
2) Develop experimental models. Two simulation models were built to serve as test beds. One of the models was the one used by Sevimoglu (2002) (non-terminating). The second one was the one used by Jones (1999) (terminating).
3) Design experiments. Three experiments were designed to answer the following questions:
    i) Can Neural Networks handle better the multivariable search space in the context of Goal Driven Simulation?
    ii) How difficult is it to train and re-train Neural Networks?
    iii) How difficult is it to implement Neural Networks-Base Goal Driven Simulation support systems?
4) Run experiments and draw conclusions.

Details on the exact methods and tools used to meet these objectives are given in Chapter 3. Details on the experimentation and results are given in Chapters 4 and 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Goal Driven Simulation

Simulation is arguably the most versatile and general-purpose tool available today for modeling complex systems (Panayioutou et al. 2000). Computer simulation has evolved over the last 40 years, thanks to the research done to improve its capabilities and to reduce its drawbacks (Kelton, et al. 1998). One of the areas in which experts have been working on is Goal Driven Simulation or Goal Oriented Simulation.

Goal Driven Simulation (GDS) refers to simulation modeling where a goal is to be met in an automated fashion (Reyes 1998). In GDS, the inputs to the model are the goals to be met, and the outputs are the required configurations to achieve those goals. The importance of automation in GDS is based on the necessity of speeding up the evaluation process in order to study real-world problems (Panayioutou et al. 2000).

Initial approaches to GDS have been classified as at-end GDS (changes to the input parameters are made at the completion of the simulation run) and as on-line GDS (ability to check the directions of the outputs of the model at intermediate points of the simulation run; in other words, these approaches do not need to run the simulation to completion unless the results are converging to the predefined goals) (Jones 1999). By avoiding running the model to completion with the wrong set of input parameters on-line GDS reduces the total execution time (Peñaloza 1999, Reyes 1998).

On-line GDS has been applied to terminating systems (Jones 1999) and to non-terminating systems (Reyes 1998), together with different statistical approaches including confidence intervals and forecasting techniques (Peñaloza 1999). Prototypes have been developed for different scenarios with the purpose of validating the method and establishing new procedures. Initial approaches to on-line GDS were applied to terminating and non-terminating systems but manipulated a single variable. These approaches although very valuable gave just partial results; "in the real world no system is analyzed on one variable" (Sevimoglu 2002). Most realistic systems are of high order and/or nonlinear, which precludes a direct solution (Sparks and Maghami 1998).

Sevimoglu (2002) applied on-line two variable GDS to a non-terminating system. The results of this effort show that the simulation execution time is significantly smaller when this approach is used instead of similar two variable approaches in at-end GDS. Sevimoglu's effort also showed the high level of difficulty of adding new decision variables to an existing model. Her on-line GDS two variable approach used code written in Visual Basic for Applications (VBA) to assist the user in finding the solution that satisfies the goal. This code describes the relations between inputs and outputs. When the simulation conditions change the relations between inputs and outputs must be reevaluated and, therefore, most of this code must be re-written.

2.2 NEURAL NETWORKS

The areas of application of ANN go from statistical and optimization problems to music understanding. Artificial Neural Networks (ANN) were developed mimicking human neurons, so they are able to learn by example receiving a set of inputs with their correspondent set of outputs. ANN are considered masters in pattern recognizition and have been used in hard-to-define type of problems. According to Krishnaswamy et al. (2002), ANN are best applied when: 1) one can specify particular influences on a phenomenon whose outcome is known with certainty, 2) the relationship cannot be described, 3) the relationship is not necessarily linear, and 4) there are no known models.

According to Rich (1991), recent ANN architectures are characterized by:

- A large number of neuron like processing elements called perceptrons.
- A large number of weighted connections between the elements. The weights in the connections represent in some manner the knowledge acquired by the network.
- Distributed control.
- Emphasis in the automation of the learning process of internal representations.

ANN and their prediction capabilities are based on the training process performed in the system. To perform a successful training, the data used for this purpose must be gathered carefully, following closely the statement "garbage in-garbage out". In addition, it is important to account with a spectrum of data scenarios from different types of situations and present them to the ANN. This spectrum must focus on the points where small changes of the inputs generate big changes in the outputs, if these points are known. The selection and the amount of input and output pairs will depend on the final goal of the training process and on the availability of the data itself.

An ANN must count with knowledge before it could participate in any type of experiment. The knowledge in the ANN is defined as the weights located in each one of the existing connections between nodes. By means of the training process the ANN determines the weight between elements to obtain an output from a specific input; in other words, it acquires knowledge. These weights can take positive or negative values. If the weight is positive, it will push the activation of the target element upward. This is called an "excitatory" connection. If the weight is negative, it will push the activation of the element downward and it is called and "inhibitory" connection (Isaacson 1998).

The ANN learning process can be classified in one of two types, supervised or non-supervised. The main difference between the two types of training is the presence of training data or the lack of it. The supervised training provides the system with initial training data, and once the training process is over, it provides the ANN with testing data to validate the knowledge. On the other hand, for the non-supervised training, no training data is given to the network. In this case, the ANN will use all the given information and will group similar input values together. Later, by modifying the weights in the connections, it will attempt to make the outputs of the similar inputs be similar as well (Reed 1998). The more number of pairs used, the better will be the knowledge learned by the ANN. Nevertheless, it is necessary to establish a trade-off between selecting enough points for good training and keeping the number of training points down to practical levels of computation (Sparks et al. 1998). Over-training the ANN is not desirable.

Similarly, ANN training can take place off-line from data available in a database, or on-line where the process that uses the neural network generates and feeds the training data to the ANN without user intervention (Laguna 2002).

2.3 NEURAL NETWORKS AND SIMULATION

An increasing volume of research has combined Artificial Intelligence (AI) and simulation in the past decade (Cho 2002). The integration between AI and Simulation has been attempted using different approaches. These efforts have established that AI could provide significant benefits in two main simulation areas: 1) the development of the simulation model, and 2) the interpretation of the simulation results.

An example of interaction between AI and Simulation, in the scenario of GDS, occurs when the simulation model interrogates the Knowledge Based System (KBS) to find out if the current solution satisfied the preset objectives/goals or not (Centeno and Standridge, 1992). A reverse way of interaction occurs when the KBS looks for the solution, stops the search, and asks the simulation model for an answer. Later the AI portion of the model goes back to a search process if the targeted value was not achieved (Centeno and Standridge, 1992).

According to Erraguntla et al. (1994), the goals of the KBS Engines are:

- To support the designing process of a simulation model, which changes from a description of the system and description of the user concerns to a question that needs to be answered, and
- To support the interpretation of the simulation outputs.

From these two goals the second one is the most difficult to achieve. There are different types of user concerns and the exact type of concern must be identified prior to any other action. The different types of user concerns can be grouped as:

1) Concerns that can be answered directly by the model execution: These are concerns related with measures of performance.
2) Concerns that require comparison of results with different simulation models: this takes place when more than one alternative is considered, and it is necessary to select the best among them.
3) Concerns related to optimization processes. This last group could be considered the most difficult to deal with because in addition to an analysis of the measures of performance, it also requires a heuristic analysis.

One type of AI is ANN technology, which has been used in conjunction with simulation in some experiments. ANN captures the relationships existing between inputs and outputs sets; this feature is able to represent the nature of a simulation model in a more efficient computational way (Panayiotou et al. 2000). ANN are metamodels, which may speedup the simulation process; therefore, a wider range of real-world situations can be studied (Panayiotou et al. 2000).

ANN has been used in conjunction with continuous simulation for outcome prediction in pallidotomy for Parkinson's disease (Hamilton et al. 2000). Because of the limited number of operations performed in this field, most of the data to train the ANN came from simulation outputs. Nonetheless, the network kept the flexibility to respond appropriately to unusual inputs. No major work has been done to marry ANN and discrete event simulation modeling.

CHAPTER 3

INTEGRATION OF NEURAL NETWORKS AND GDS

This chapter discusses how ANN can be used for GDS. A thorough literature review helped to determine how best to integrate ANN and GDS. A software review was also done to select a package to experiment with and test proposed concepts.

3.1 Conceptual Framework for the Integration

A conceptual framework for the integration of ANN and GDS is given in Figure 3. This framework has been designed with one objective in mind: *to facilitate GDS for the non-expert user of simulation to make accurate decisions with ease.*

The framework supports:

1) Two modes of operation: training and daily usage.
2) Two types of users: expert and non-expert.

To provide this support, the framework requires several commercial of the shelf (COTS) software. It needs:

1) A simulation engine.
2) A neural network engine.
3) A set of routines to interact with each of the two types of users.
4) A set of routines to receive, present, and transfer data in the appropriate format.

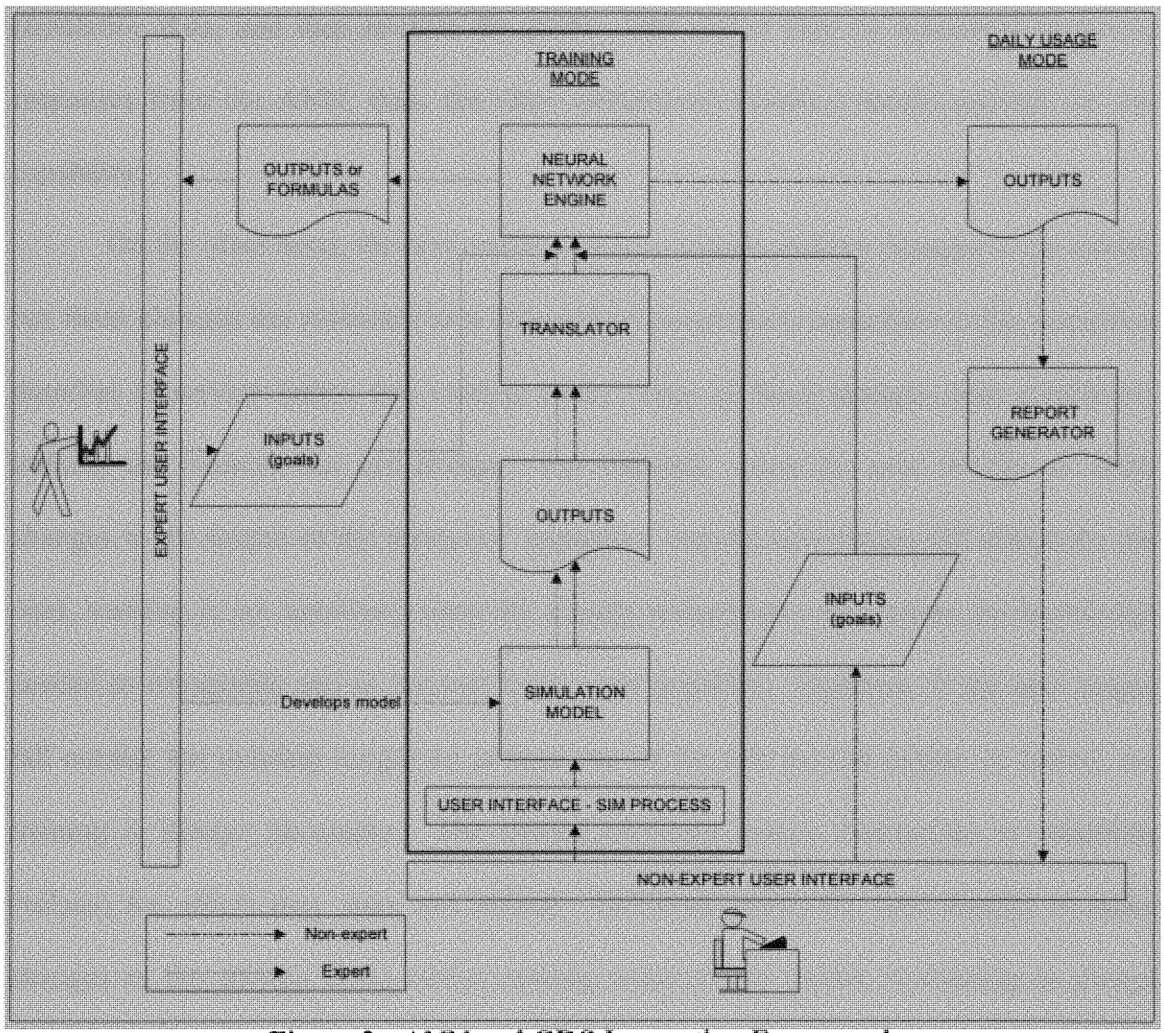


Figure 3: ANN and GDS Integration Framework

The designation of expert refers to a knowledgeable person in the simulation field. The expert user designs and builds the simulation model according to the system's specifications and uses a customized interface to train the ANN with simulation data. A translator is embedded in the framework to take the outputs of the simulation model and feed the ANN. This translator takes care of the necessary communication protocol in order to exchange information between the two packages. If the expert decides to make changes to the current system, s/he must first modify the simulation model, generate new simulation data, and then re-train the ANN. The expert may be interested in getting responses from the ANN; thus, the same customized interface should provide for this capability.

On the other hand, the non-expert user may require making changes to the model, but these changes are restricted to a finite set of parameters: capacities of the resources, duration of service times, and routing times. Any of these changes *may* involve re-training the network but not always. For example, when the user decides to make changes in the ranges of the resources' capacities, and the new ranges fall within the old ones, training is not necessary since the already trained networks posses the knowledge needed to intrapolate and to respond to the user inquiries. However, if the new ranges fall outside the existing ranges on at least one of the two bounds (upper or lower), re-training is always necessary. The trained networks cannot extrapolate to situations (values) for which they were not trained. Similarly, when the user decides to make changes to the service times and/or routing times of the model, re-training is always necessary since these are given as specific constant values and not as ranges. All these situations must be supported 100% by the framework, which should advice the user whether re-training the ANN is needed. In the case when it is concluded that re-training is required, new simulation data must be produced in order to re-train the ANN. Thus, the framework should provide this user with the capability of transparently modifying the simulation model and re-training the NN.

The daily usage is primarily for the non-expert user. The idea is that the non-expert will query the NN-based GDS application on a regular basis (daily, weekly) to determine how many of each resource should be on the schedule to meet a desired service level. So, the non-expert user will provide the goals (desired level of service) through a customized user interface; then, the ANN processes this information and provides the user, in a nice report format, with the system's configuration that meets the desired level of service.

The expert user has the capability of generating transformation equations. These equations can be utilized to implement the prediction capabilities of a trained ANN even in the absence of a specific NN engine.

The proposed design seeks to incorporate the most relevant aspects of GDS and ANN according to the published literature (Chapter 2). Among these aspects are:

- GDS automation.
- Handling n>3 variables.

GDS must be automated, which means that all necessary processes should remain transparent to the end-user (expert and non-expert). In the case of the non-expert user all processes are 100% seamless and transparent because this user is involved exclusively in the daily usage of the framework, and it does not need to be aware of any of the activities taking place behind the scenes. For the expert user, the level of automation is narrowed to management of the neural network engine and full access to the simulation package. In case there are structural modifications to the existent model, the expert user must edit and run the simulation model in order to generate data to re-train the ANN.

An important aspect of designing a GDS framework is the ability of it to handle several variables simultaneously and easily. ANNs are able to handle multivariable scenarios with ease after they have been designed and trained. Thus, in this effort all that was required was an effective user interface to capture the desired number of variables, including the set of goals, and a number of parameters used to define the configuration of the system at any specific moment.

The last aspect considered in the design of the framework was the nature and availability of the training data. In order to have a NN that mimics realistic situations, it is necessary to present high quality data and a wide spectrum of scenarios. Since in this effort a simulation model is being used to train the NN, the quality and quantity of the data is fully controllable, and all statistical guidelines can be met with ease.

Based on the requirements of both types of users, it was established that the framework requires:

a) A flexible simulation package with capabilities of interacting with other packages.

b) A NN package, which could be embedded under a controlling application. Integrating an ANN is dependent on the selection of the training technique. It was concluded that a supervised training algorithm was the best alternative for GDS because GDS requires the very good prediction capabilities obtained under supervised training.

c) A high level programming language to implement the customized interfaces, the translators of data, and the report generation.

3.2 TOOLS

A software review was done to determine which commercial packages were more suitable to prototype the proposed framework. Two commercial packages had to be selected to prototype the framework: one package to build the ANN, and one package to build the simulation model.

One of Rockwell Software Inc. most popular and successful products is ARENA. This product has been used by well-known organizations worldwide as a support tool for their decision-making process. The ARISE Center at the Industrial and Systems Engineering Department counts with one of the latest versions of this software. Further ARENA possesses a VBA processor, and since the framework requires a high level programming language, it was considered that ARENA VBA was sufficient to develop the interfaces and the communication protocols required by the framework. Hence, ARENA was selected as the COTS package to build the simulation model and the integration routines.

The selection of the ANN package took a little more analysis because there was not previous experience in this area. After conducting a preliminary evaluation of capabilities, the selection was narrowed down to three packages: Neural Studio, Neuro Solutions, and NeuroShell 2. After comparing the characteristics of the three packages, NeuroShell 2 was identified as the software of choice because of its simplicity of use and its assisted design module. A brief description of each one of the two other packages is given in Appendix A.

NeuroShell 2 is one of Ward Systems experimental kits, which provides sixteen classic Neural Networks architectures to choose from, and it is strongly recommended for academic and research purposes. NeuroShell 2 is divided in three modules: beginners, advanced, and runtime facilities.

The beginners' module allows the user to build simple networks with only one hidden layer and using back propagation as the learning methodology. This does not necessarily mean that this type of networks can be used exclusively to solve simple problems; on the contrary, the resultant networks are complete and powerful prediction and classification tools. The main assistance given by this module is the presetting of default values for each one of the network's parameters such as, learning rate, momentum, and number of hidden neurons. These network's parameters have the following definitions:

Learning rate: Each time a pattern is presented to the network, the weights that lead to an output node are modified slightly during training in order to produce a smaller error the next time that the same pattern is presented. The amount of weight modification is the learning rate times the error. In other words, learning rate is the level of adjustment in the weights at each step of the training.

Momentum: If large learning rates are used the oscillation of weight changes increases, and either the learning process is never completed or the model converges to a solution that is not optimal. One way to allow fast learning without much oscillation is by making the weight change a function of the previous weight change, which provides a smoothing effect. Momentum is the proportion of the last weight change that is added to the new weight change.

Number of hidden neurons: The default number of hidden neurons used by NeuroShell 2 for a three-layer network (one input layer, one hidden layer, and one output layer) is computed with the following formula (Ward Systems, 2003):

$$No.\ hidden\ neurons = \frac{1}{2}\left(No.\ inputs + No.\ outputs\right) + \sqrt{No.\ patterns}$$

Pattern: A single record of data used with training purposes.

The advanced module of NeuroShell 2 allows the user to design ANN by selecting anyone of the sixteen types of architectures distributed under five different learning paradigms: Back Propagation (BP), Unsupervised Kohonen (UK), Probabilistic Neural Networks (PNN), General Regression Neural Networks (GRNN), and Group Method of Data Handling or Polynomial Nets (GMDH). Within this option, it is possible to set the scaling function for all types of networks but UK, and the activation function for the BP networks exclusively. The purpose of scaling functions is to "scale" the input data loaded into the ANN into a numeric range for which the ANN works efficiently. Usually, these ranges are either binary [0,1] or bipolar [-1,1]. On the other hand, the purpose of the activation function is to propagate data from a preceding layer to a succeeding layer in an ANN. Numerous research efforts have been done in order to find the ideal activation function; but yet there is no general agreement out of these efforts. The activation functions most commonly used include logistic (sigmoid binary and bipolar), linear, and hyperbolic tangent. Some characteristics that are important for BP networks at the time of selecting an activation function are continuity, differentiability, non-decreasing monotonic, and to approach asymptotically to the maximum and minimum values.

The advanced module enables the user to generate various graphs for all type of architectures, and it provides contribution factor values and bar graphs for BP networks. *Contribution factors* are a rough measure of the importance of a specific input variable in predicting the network's output, relative to the other input variables in the same network (Ward Systems Group, 2003). These factors were used to compare the ANN results to the conclusions drawn by Sevimoglu (2002), Correa (1999), and Jones (1999). This module is also able to translate alphanumeric strings or symbols into values that can be used by the ANN. This capability was not needed for this effort.

The third module (runtime facilities) allows the user to fire a trained NN from a different program, in real time.

The initial experimentation to integrate ANN with GDS was done using the beginners' module in order to acquire knowledge about how the software works and the possibilities of interaction with other applications. After the first experimentation was over, the advanced module was used to explore new options, namely new architectures. Finally, the runtime facilities module was used to integrate the two technologies in a working prototype.

3.3 Macro Steps of the Integration Methodology

Based on the requirements of the conceptual integration and the selected tools, the methodology to integrate ANN and GDS is as follows:

1) Develop simulation model. In this particular case, it is an ARENA model; however, it can be a model in any simulation package that allows the

interaction with a high level programming language. The difference resides on whether the model is the driver of the framework or it is just embedded in it. In the case of ARENA, the model is actually the driver as it executes VBA routines at various events of its execution as explain in section 4.3.

2) Develop the ANN. An initial ANN must be provided by the designer, but also appropriate VBA subroutines should be given so that the ANN can be re-built when certain changes are made to the model as required by the design (section 3.1) and prototype (section 4.2).

3) Develop (program) routines to automate training of ANN. These routines know how to get inputs from the user to change the model, how to re-run the model, and how to re-build the ANN. For this effort these routines were written in VBA for ARENA.

4) Develop (program) high level programming language routines to present and manage the user dependent user interface.

CHAPTER 4

EXPERIMENTAL PROTOTYPE

This chapter describes a prototype of the integration. The prototype required three main components: a simulation model, an ANN, and VBA routines. It is important to keep in mind that the prototype has been developed with two goals in mind:

1) Experimental tool: Embedding an ANN can be done in several ways. Recommending a "best" way for it in the context of GDS requires that a series of experiments be done; thus, the need of an experimental prototype.
2) Feasibility assessment: An outcome of this effort is a theoretical framework that others can use, its feasibility needs to be assessed.

The discussion of the experimental prototype is sectioned by the main components of the integration framework (Figure 3):

a) Simulation model
b) Neural networks
c) Integration routines: discussed from the perspective of each user.

4.1 THE SIMULATION MODEL

Initially, two simulation models were considered for the present effort. The first one corresponds to the model used by Sevimoglu (2002), and the second one corresponds to the model used by Jones (1999). The selection of these two models obeyed first, to the availability of simulation data, and second, to the possibility of studying and analyzing two different types of systems. Sevimoglu (2002) studied an Emergency Room (ER), which is considered a non-terminating type of system, whereas Jones (1999) studied a manufacturing facility, which is considered a terminating type of system. Terminating systems are easier to analyze than non-terminating ones because the starting and ending conditions are defined by the nature of the system, and the initial condition is fixed. On the other hand, non-terminating systems experience two phases, an initial transient phase that depends on the starting conditions and an unchanging distribution phase that is independent of the starting conditions (steady state). The understanding of the latter phase, steady state, is the main interest during research because it leads to the understanding of the system's behavior. Terminating systems rarely achieve steady state. For the purposes of this research either a terminating or a non-terminating system could have been chosen, but we decided for the non-terminating system used by Sevimoglu (2002). It was expected that any conclusion that could be drawn from this complex system could be generalized to a less complex system.

The ER model used by Sevimoglu (2002) classifies its patients according to their illness condition in four categories, emergent, urgent, non-urgent, and stable. Regardless the type of patient, they can arrive to the ER by any of the following means, by ambulance, by fire rescue, or by themselves. If arriving by ambulance or by fire rescue, a bed has to be available for the patient upon arrival; if not, the patient has to be taken to a different hospital. If the patient arrived by its own means, a triage nurse initially evaluates the patient and assigns a priority to it. The priority scale is as follows:

- Category 1: These patients are immediately placed on a bed after the Triage nurse and the registered nurse perform an initial evaluation.
- Categories 2 to 4: These patients are taken to a registration representative after the initial assessment done by the Triage nurse. Patients' category 2 wait for a regular or an extra bed, patients category 3 wait for a regular, extra or a fast track bed, patients category 4 wait for an extra or fast track bed.

After the registration process is over patients category 2 to 4 wait in the waiting area until one bed is made available. Once a bed is made available, the patients wait for a registered nurse to perform the initial evaluation. Then, the registered nurse informs the doctor that there is a patient ready and explains to him/her the details of the initial evaluation. Next, the doctor makes an examination and decides which further procedures are required, including if the patient should be admitted, observed or discharged. If the patient is to be admitted, some paperwork has to be done while the patient waits, and a nurse or an assistant from the unit to admit the patient should come and escort the patient. If the decision is to discharge the patient, the patient waits until the required paperwork is processed. Finally, if the patient has to be observed the registered nurse must check on it at regular basis. Specific numeric data for this model can be found in the work of Sevimoglu (2002). A flow chart of the process of this test bed and detailed information about the ARENA 5.0 simulation model is presented in Appendix B.

4.2 The ANN Model

The ANN component of the prototype has been designed and built using NeuroShell 2. For the framework, designing and building an ANN in NeuroShell 2 is a relatively simple task because the software automatically generates the layers and neurons needed. The framework is expected to have a trained ANN for the everyday user; however, changes to the system's parameters may require the ANN to be re-trained. Thus, the framework has been given the capability of automatically training an ANN from scratch.

Whether manually or automatically, the steps to design and build an ANN are the same. There are four elements that need to be provided in order to design and build an ANN using NeuroShell 2:

1) A set of I/O scenarios.
2) Definition of independent and dependent variables.
3) Architecture to be used to build the actual layers.
4) Learning process.

Each of these elements requires the system designer to select the best option for the GDS context.

4.2.1 Setting I/O Scenarios

Providing the set of I/O scenarios requires deciding how the set will be fed to NeuroShell 2. There are two options: 1) import a data file, or 2) input the set manually. For GDS, the set must be provided via file because since a simulation model generates the scenarios, the number of scenarios is large. Thus, manual input is not an option. NeuroShell 2 offers the alternative of importing ASCII files, binary files, or spreadsheet files. Because the simulation package has a natural capability of exporting data to an ASCII file, importing files on this format is the best alternative. The use of an ASCII file is the best alternative when using ARENA; however, it is possible that other packages may by default send their outputs to other types of files, e.g. Excel. The feasibility of the framework does not rest on the I/O set being fed via ASCII, rather on the I/O set being imported into the NN package, regardless of the file format chosen.

In the case of the ER model, generating the ASCII files meant running the model for certain number of replications; each replication for a given length, and ensuring that the outputs of this run are written in the required NeuroShell 2 format. A subroutine, programmed in VBA for ARENA (Appendix D.7), takes care of sending the outputs to the ASCII file. More details on this routine are given in section 4.3. Regarding the requirements for the ASCII file, NeuroShell 2 calls for organizing the input and output variables in independent labeled columns. Preferably the label names should not include spaces because this could create a problem when importing the file into NeuroShell 2. When importing the ASCII file into NeuroShell 2, it is necessary to specify if spaces, tabs or commas separate the data. In our experimental prototype the data was separated by using spaces; hence, including spaces in the columns label names would generate errors at the time of importing the data file. The variables to be used as "outputs" of the ANN should appear first in the ASCII file, and the variables to be used as "inputs" of the ANN should appear immediately after. The objective of this layout is to facilitate the automation of the experimental prototype; *this is not a NeuroShell 2 requirement.* It is important to be aware that the order in which the variables are presented to NeuroShell 2 during training is kept constant for firing purposes. Each data row is expected to have the same number of variables to ensure proper supervised learning process. NeuroShell 2 considers missing data as errors, and it ignores the row if one of the values is missing. An example of the ASCII file distribution and labeling is presented in Figure 4.

```
NoBeds        NoDoc       NoNur             AveQNur                   UtilNur
 10             1            6         6.41904641059947E-02       0.763303951618937
 18             1            6         6.78311940431272E-02       0.765156307010476
 26             1            6         6.71479355835947E-02       0.766309049704611
 34             1            6         6.67546140139281E-02       0.763607174700303
 40             1            6         6.69715227827895E-02       0.766033743094173
 10             2            6         0.067565694927857          0.765421042333161
```

Figure 4: Portion of ASCII file to be fed to NeuroShell 2

Importing the data file into NeuroShell 2 takes place by executing the NeuroShell 2's application **Impascii.exe**. This application is executed by using the *shell* statement in a VBA for ARENA routine (Appendix D.4.1). The process to import the data takes place by sending keystrokes to it, using the *sendkeys* statement.

4.2.2 Defining ANN Variables

Once the data has been imported into NeuroShell 2, it is necessary to define which variables are the independent variables ("inputs"), and which are the dependent variables ("outputs"). Upon importing the data, NeuroShell 2 creates a data grid where the columns are named using the labels of the columns in the ASCII data file.

To define the variable type, we must insert in each cell of the first row one of three letters: *I* for Input, *A* for Actual output, or *U* for Unused (a blank cell is also read as an unused variable). To give more flexibility to the experimental prototype, it was decided that the user would have the option to select which and how many "inputs" to use to predict the value of the "outputs". The ER model has 6 possible "inputs"; each one with the possibility of being defined as variable type *I* or *U*. By the Multiplication Rule, the total number of possible combinations is 64 (2^6), but the scenario where all the "inputs" are defined as Unused (U) is not a valid scenario, leaving only 63 combinations.

Once trained, the ANN acquired knowledge is dependent and valid only for the combination of I/O for which it was trained. Therefore, if we wanted the ANN to work for all 63 combinations, an array of 63 ANN has to be trained independently. It is important to note that the simulation model is run only once, and that only one ASCII file is generated. The 63 combinations use the same file, and use only the variables in the combination, skipping the other ones. This process gave as final result the generation of 63 different trained networks, but the end user is never aware of this. To automate this process, a VBA subroutine was written (Appendix D.3.2).

4.2.3 ANN Architecture

The next step in building an ANN is to select the type of architecture to be used in the actual layers of the network as well as the learning paradigm. NeuroShell 2 offers 16 different architectures grouped in five major learning paradigms. These five groups are: Back Propagation (BP), Unsupervised Kohonen (UK), Probabilistic Neural Networks (PNN), General Regression Neural Networks (GRNN), and Group Method of Data Handling or Polynomial Nets (GMDH). Among them, the most popular one is the BP learning paradigm. BP networks are one type of supervised networks and are known for their ability to generalize well. For this effort, the BP architecture to build the actual layers of the ANN was chosen.

4.2.4 Learning Process

The process of designing and building the ANN finalizes with the actual learning procedure (training), where the ANN learns the relationships existing between the "inputs" and the "outputs". In NeuroShell 2 the duration of this process is not preset and the network keeps training until the user interrupts the process. The learning process (training) is perhaps the most critical activity while building an ANN model. A critical question is: *how can it be determined that the ANN has learned enough?* In real world, there are basically two situations: 1) the training set consists of all possible sample cases that will be encountered by the network, and 2) the number of possible sample cases is infinite or at least very large, and where the training set is only a representative of this enormous number of sample cases. In the first case, the learning process should continue until it fails to make any perceptible progress. For the second case, the correct action is to stop the learning process when the network gives the "best possible results" to patterns that were not used during training (Ward Systems Group, 2003). The experimental prototype used in this effort represents the second case since only a small group of all the possible scenarios (sample cases) are used during training. Hence, a training stoppage condition must be set in order to achieve automation. The number of scenarios used for training is discussed further in Section 4.3.

One way of setting the stoppage condition is by stopping the training once the "optimal point" has been reached. The *optimal point* is defined as the point where the error obtained with the test set slowly begins to increase after reaching its lowest point. The only problem with this approach is that the duration of the training process cannot be correctly estimated, and such estimate is critical to automatically train the network. Therefore, this approach is not convenient.

Another approach to detect the training stopping point is to use a sample of the data set available, generating two different sets: one for training (*training set*) and one for testing the level of training reached (*test set*). NeuroShell 2 offers the possibility of extracting a *testing set* out of the original ASCII file. The purpose of the testing set is to use it for testing during the training process (simultaneously) to achieve a balance between the generalization and memorization capabilities of the network. This balance is known as *Calibration,* and it avoids over training the networks. When using this testing set during training, NeuroShell 2 displays the values of two very important variables for training purposes: minimum average error (MAE) and number of epochs elapsed since the last minimum occurred. Essentially, the goal is to minimize the average error without consuming large amounts of time and while keeping the total number of epochs low. The greater the number of epochs, the greater the risk of memorization. For BP networks, NeuroShell 2 recommends to continue the training process until the numbers of events since the minimum error (calculated in the testing set) is greater than 20,000 and preferably lower than 40,000.

There are different methods to extract the testing set. Among the most popular are random extraction of the Nth percent of the total sample, selecting one pattern every

Nth pattern, extracting all the patterns from row N thru M (where N and M are any two rows in the data file), and selecting the testing set by a row marker, if any. There is no preferred or recommended method to perform the testing set extraction. The only recommendation is that it should not include patterns that are part of the training set. Thus, this effort used the Nth percent random extraction method, for which, NeuroShell 2 recommends choosing a testing set size between 10 and 40 percentage of the total sample. It is necessary to take into consideration the number of data points available. For instance, the ER model has around 100 data points (scenarios), which prevents a large testing set. In this experimental prototype, an N equal to 20 percent was selected.

In order to automate this task, it was necessary to conduct an experiment to determine the adequate training time. Table 1 provides the results of this experiment, where the number of events was set to be greater or equal than 20,000 since the last minimum error on the testing set. It is clear from Table 1 that the minimum average error for the training set decreases as the training time increases, while the error for the testing set remains basically constant. Although the error for the training set could reach a lower value with a higher training time, it was decided that a training time of 45 seconds is good enough to reach a MAE, while reducing the risk of memorization. With this training time the average number of events since the last minimum was 112,000, although this number looks high if compare with the set value of 20,000, it was used in order to improve the performance of the network with the training set (reduce the MAE for the training set).

With a training time of 45 seconds, it will take 47.25 minutes to perform the training required by the 63 ANN (45 seconds x 63 networks). Although it may seem as a long time, it is necessary to remember that rebuilding the ANN macro network will occur only when there are significant changes made to the simulation model parameters. After the training process is over, the ANN is ready to be used (fired).

Table 1: Results with Different Training Times

Training Time (seconds)	**Minimum Average Error (Training)**	**Minimum Average Error (Testing)**	**Events since last minimum (Testing)**
15	0.07969298	0.22547542	78186
30	0.0737239	0.22547549	95514
45	**0.06807132**	**0.22547542**	**112274**
60	0.06621125	0.22547542	122822

Besides the training time, the network parameters that were defined in Section 3.2 have to be set before initiating the learning process. NeuroShell 2 provides default values for these parameters according with the complexity of problem. Since the system being modeled (ER) is a real world system, the level of complexity was set to **complex**. The following are the recommended values given by NeuroShell 2 and used in the experimental prototype:

a) *Learning Rate (LR)* = 0.1

b) *Momentum* = 0.1

c) *Hidden Neurons* = Depend on the number of inputs and outputs.

d) *Pattern Selection* = Random

4.3 INTEGRATION ROUTINES

The goal of the integration routines presented in this section is *to achieve the automation of the entire process with minimal to none user interaction.* Further, the framework calls for support two types of users: expert (rare usage) and the non-expert (daily usage). Therefore, a variety of VBA routines are needed to have a functional framework. These routines are classified as:

1) User Interaction Routines (UIR): Manage the interaction with each type of user, offering appropriate options and reports to each one.

2) ARENA Control Routines (ACR): Manage the modification of the simulation model and run it as needed.

3) NeuroShell 2 Control Routines (NCR): Manage the running of the ANN either for querying a trained network, or rebuilding a macro ANN. Translation routines belong to this category (Figure 3).

A list of these routines and their particular functionality is given in Table 2. Because these routines are integrating two heterogeneous tools and the user, some of them may belong to more than one category, but they are "mostly" of one. In the column "called by" in Table 2, routines with and "(A)" are actually automatically triggered by the stated ARENA events. In addition, a set of global variables and constants had to be defined (Appendix D.1), and their values stored in the **VARIABLES** element whenever necessary.

An overview of how the GDS_Main routine triggers different routines to enable the integration is presented in Figure 5. The ARENA simulation model has the master control; hence, each one of the routines is executed at a particular event experienced by the model as indicated in Figure 5 and Table 2. The RunBegin event invokes the GDS_main routine (Appendix D.2.1),which decides what subroutines to call based on user type (expert or non-expert) and the type of interaction selected by the user (to make changes to the model, to fire an already trained network, or to quit the program). In doing so, it displays the forms shown in Figure 6. Once the user makes a selection in the UserType form, a global variable is set to the type of user: expert (1), or non-expert (2). If the selection is to quit the program, this global variable takes the value of zero and the subroutine CancelRun (Appendix D.2.2) is called.

Before continuing the discussion of the various routines, it is necessary to point out that when the Cancel Button is chosen, the user will be taken either to the previous user form or completely out of the system. Also, the design of the framework (Figure 3) clearly shows that there are activities shared by both users. Hence, its prototype is described from each user's perspective.

Table 2: Integration Routines

Routine Type	**Routines Names**	**Functionality**	**Called by**
UIR (Appendix D.2)	GDS_main	Coordinates routines to call based on user type and choices.	RunBegin (A)
	CancelRun	Cancels the program setting the number of replications and replication length to 1.	GDS_main
ACR (Appendix D.3)	ChangesMain	Oversees changes done to the model.	GDS_main
	CheckingRanges	Checks if re-training is needed by comparing the old and new capacities' ranges	ChangesMain
	SetScenarios	Calculates and generates all combinations of capacities values to generate training data.	GDS_main
	CheckReplications	Every 15 replications stores the value of the measures of performance in an ASCII file and updates the variables of the model.	RunEndReplication (A)
NCR (Appendix D.4)	NN2Using	Opens the selected NN and coordinates all actions required to fire the NN, and return results to the user.	GDS_main
	NN2Training	Creates description file and imports the ASCII file into NeuroShell. Extracts testing set, sets the inputs, trains the NN, creates the ".def" file, and generates the VB code.	GDS_main
	SelectingNetwork	Selects appropriate network to use based on user's inputs	GDS_main
	RunNewANN	Opens the selected NN and coordinates all actions required to fire the NN after the training is completed.	RunEnd (A)
	NN2Querying	Loads the ENTRY form for user reference and calls the subroutines to use the already trained NNs.	GDS_main

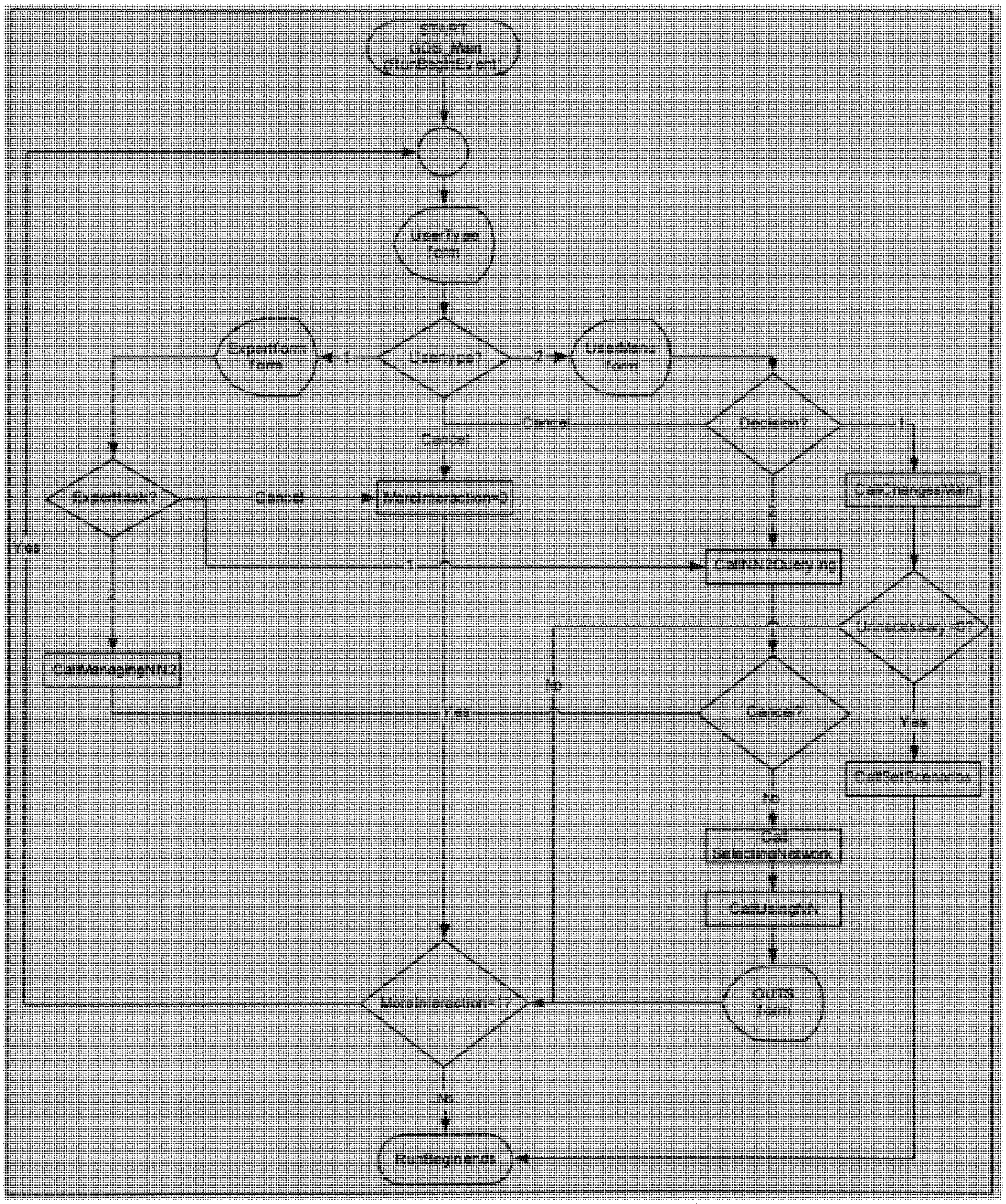


Figure 5: GDS_Main Routine Interaction Flowchart

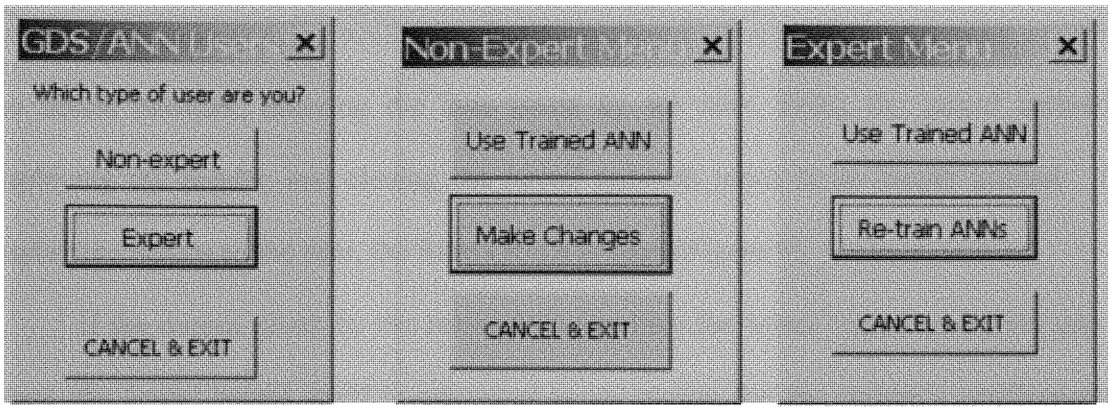


Figure 6: UserType and Users Menus forms

4.3.1 Non-Expert User

If the user is a non-expert, two options are given by the UserMenu form (Figure 6): 1) use the existing macro network (63 networks), or 2) make changes to the input parameters that were used to train the existing networks. Since ARENA is the master application in the prototype, the various routines are triggered from specific ARENA model events as shown in Figure 7.

For using the existing set of networks, the form shown in Figure 8 is displayed. The ENTRY form provides guidance and instructions for the user, in order to avoid mistakes related with firing networks, as well as erroneous results due to the use of a macro ANN that does not model the current characteristic of the system, i.e. the ranges of the resources' capacities for which it was trained are no longer valid. Since this form is asking for user inputs, a series of subroutines to check existence and consistency of the data were developed (Appendix D.5). Once the user's goals have been collected, the subroutine SelectingNetworks is called (Appendix D.4.2). This subroutine matches the user input to a corresponding trained network; then, the subroutine NN2Using takes control (Appendix D.4.3). NN2Using is a subroutine in charge of several tasks: opening the neural network, entering the data given by the user, firing the neural network, displaying the results, and closing the neural network when is no longer needed. To display the results to the user, the form OUTS is used (Figure 9).

If the output given by the network is less than the lower bound set by the user's range, the lower bound itself is presented as an answer for the system's configuration because if it is possible to meet the goal with a level of resources below the range set by the user, it is clear that the goal will be met with a level of resources equal to the lower bound of the user's range. However, if the situation is the opposite, the output given by the network is higher than the upper bound set by the user, the goals are infeasible under the current ranges set by the user. In this case, a warning message regarding the infeasible scenario is presented to the user along with the output given by the neural network. *It is important to remark that this detection of infeasibilities is possible if and only if the ANN has been trained for a wider range than the one set by the user.*

The Changesform form (Figure 10) offers 5 different options to make changes: 1) to the resources' capacities, 2) to the service times, 3) to the routing times, 4) to continue the process by submitting the changes, and 5) to cancel this action and go back to the UserType form. Each option leads to other forms, capturing the values from the user and transferring them to the ARENA model (Figure 11, Figure 12, and Figure 13). The subroutine that manages all these activities is ChangesMain, its code is included in Appendix D.3.1 and its flowchart is included in Figure 14.

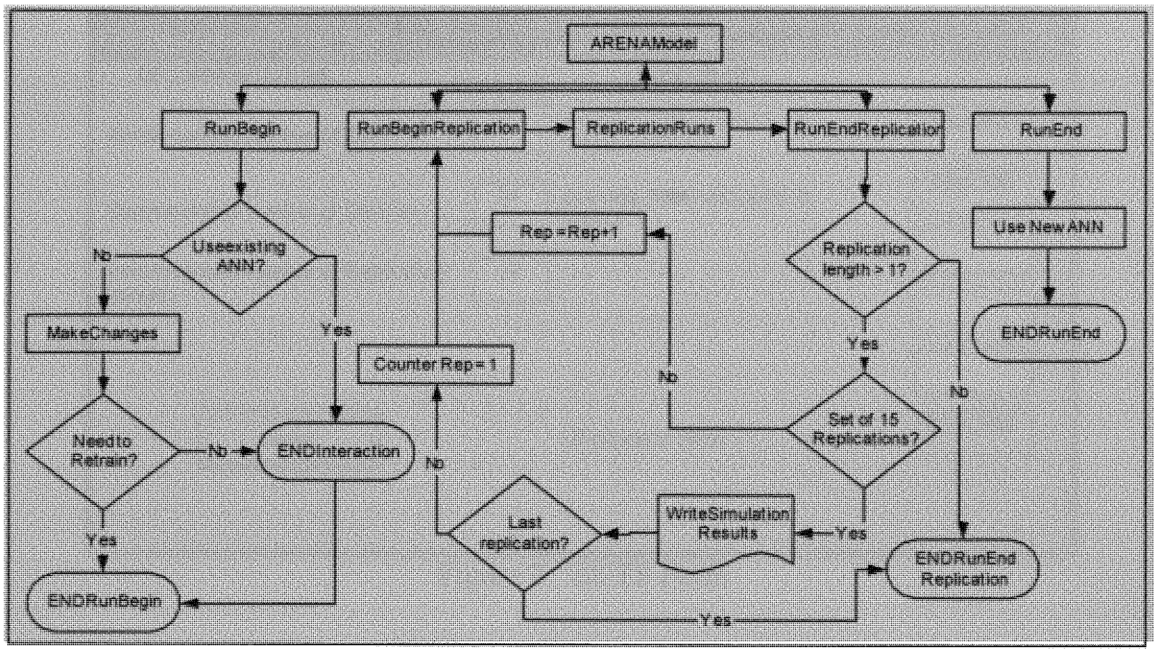
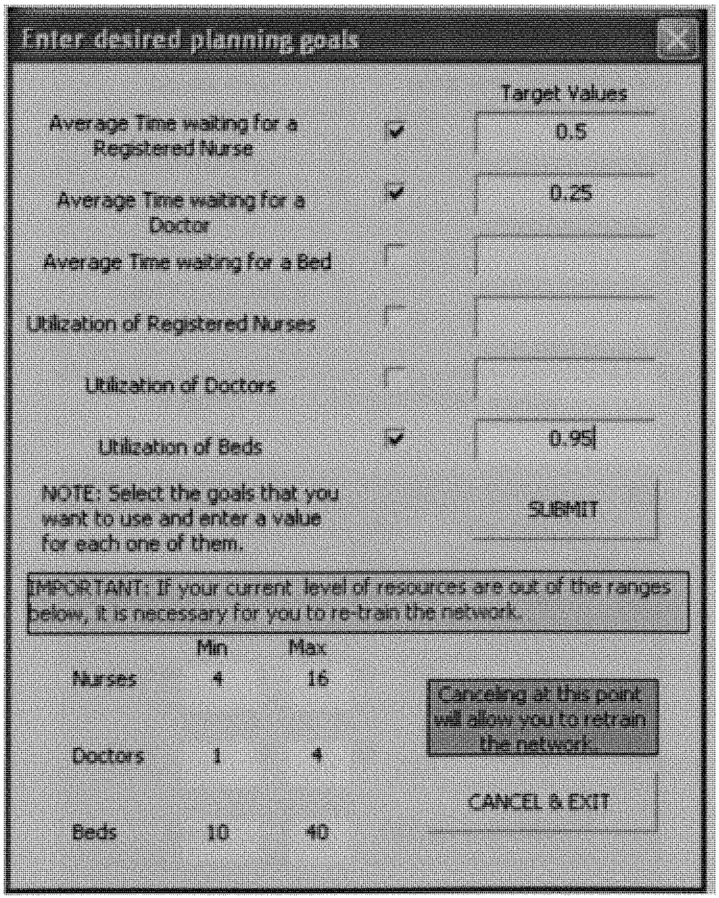


Figure 7: Events Programmed for Non-Expert User


Enter desired planning goals

		Target Values
Average Time waiting for a Registered Nurse	✓	0.5
Average Time waiting for a Doctor	✓	0.25
Average Time waiting for a Bed		
Utilization of Registered Nurses		
Utilization of Doctors		
Utilization of Beds	✓	0.95

NOTE: Select the goals that you want to use and enter a value for each one of them.

SUBMIT

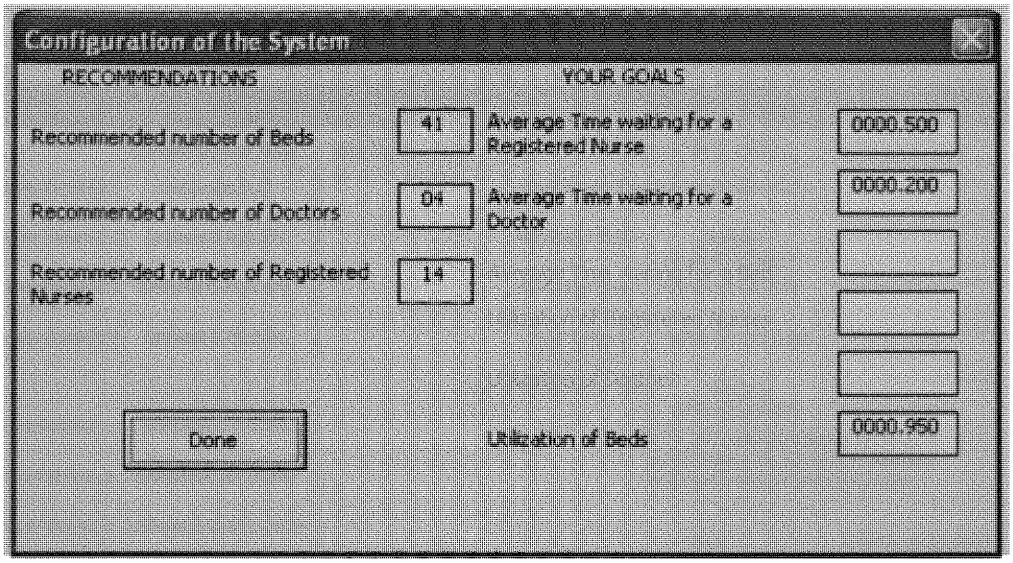
IMPORTANT: If your current level of resources are out of the ranges below, it is necessary for you to re-train the network.

	Min	Max
Nurses	4	16
Doctors	1	4
Beds	10	40

Canceling at this point will allow you to retrain the network.

CANCEL & EXIT

Figure 8: ENTRY Form


Configuration of the System

RECOMMENDATIONS		YOUR GOALS	
Recommended number of Beds	41	Average Time waiting for a Registered Nurse	0000.500
Recommended number of Doctors	04	Average Time waiting for a Doctor	0000.200
Recommended number of Registered Nurses	14		
		Utilization of Beds	0000.950

Figure 9: OUTS Form

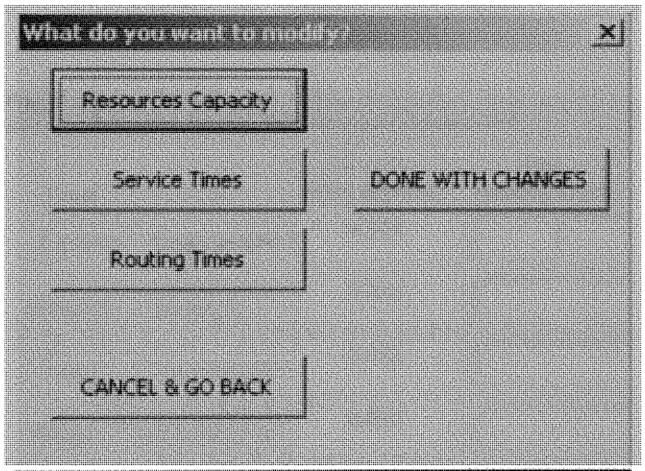
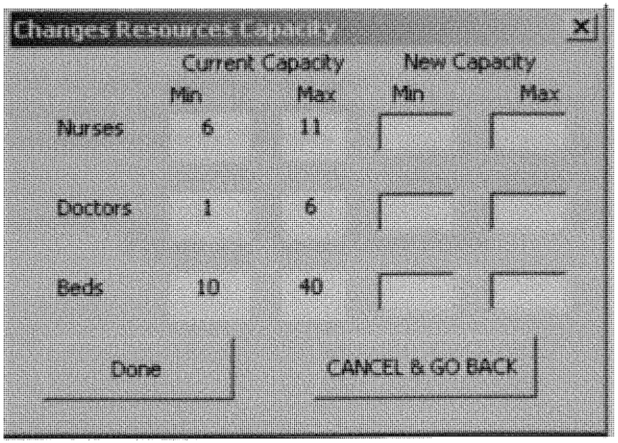
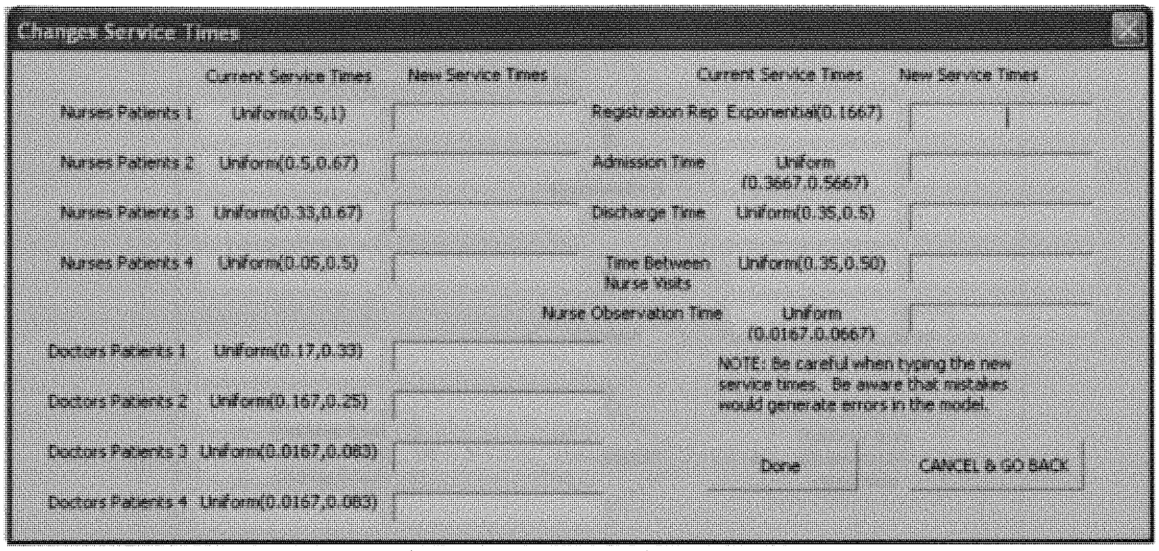


Figure 10: Changesform Form


Changes Resources Capacity

	Current Capacity		New Capacity	
	Min	Max	Min	Max
Nurses	6	11		
Doctors	1	6		
Beds	10	40		

Figure 11: ResCapacity form

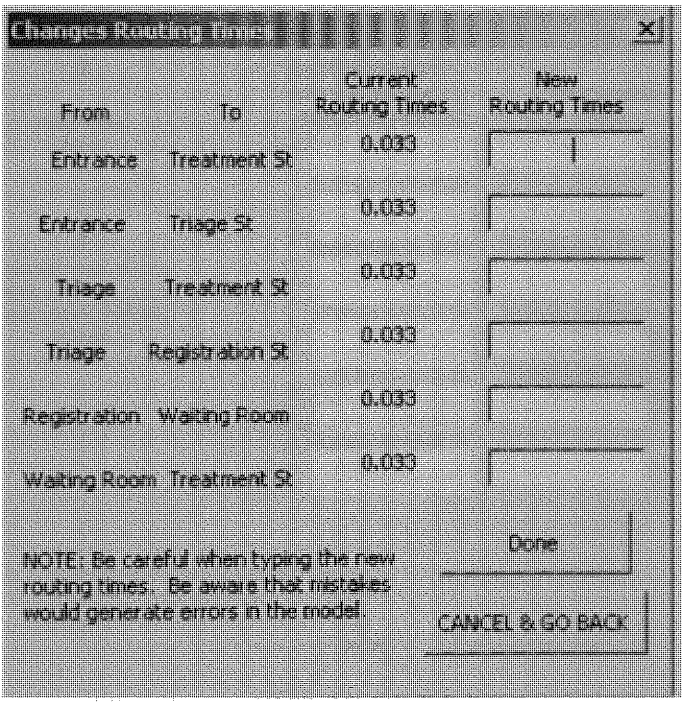


Changes Service Times

	Current Service Times	New Service Times		Current Service Times	New Service Times
Nurses Patients 1	Uniform(0.5,1)		Registration Rep	Exponential(0.1667)	
Nurses Patients 2	Uniform(0.5,0.67)		Admission Time	Uniform (0.3667,0.5667)	
Nurses Patients 3	Uniform(0.33,0.67)		Discharge Time	Uniform(0.35,0.5)	
Nurses Patients 4	Uniform(0.05,0.5)		Time Between Nurse Visits	Uniform(0.35,0.50)	
			Nurse Observation Time	Uniform (0.0167,0.0667)	
Doctors Patients 1	Uniform(0.17,0.33)				
Doctors Patients 2	Uniform(0.167,0.25)				
Doctors Patients 3	Uniform(0.0167,0.083)				
Doctors Patients 4	Uniform(0.0167,0.083)				

NOTE: Be careful when typing the new service times. Be aware that mistakes would generate errors in the model.

Figure 12: ResServiceT Form



Changes Routing Times

From	To	Current Routing Times	New Routing Times
Entrance	Treatment St	0.033	
Entrance	Triage St	0.033	
Triage	Treatment St	0.033	
Triage	Registration St	0.033	
Registration	Waiting Room	0.033	
Waiting Room	Treatment St	0.033	

NOTE: Be careful when typing the new routing times. Be aware that mistakes would generate errors in the model.

Figure 13: RoutTimes Form

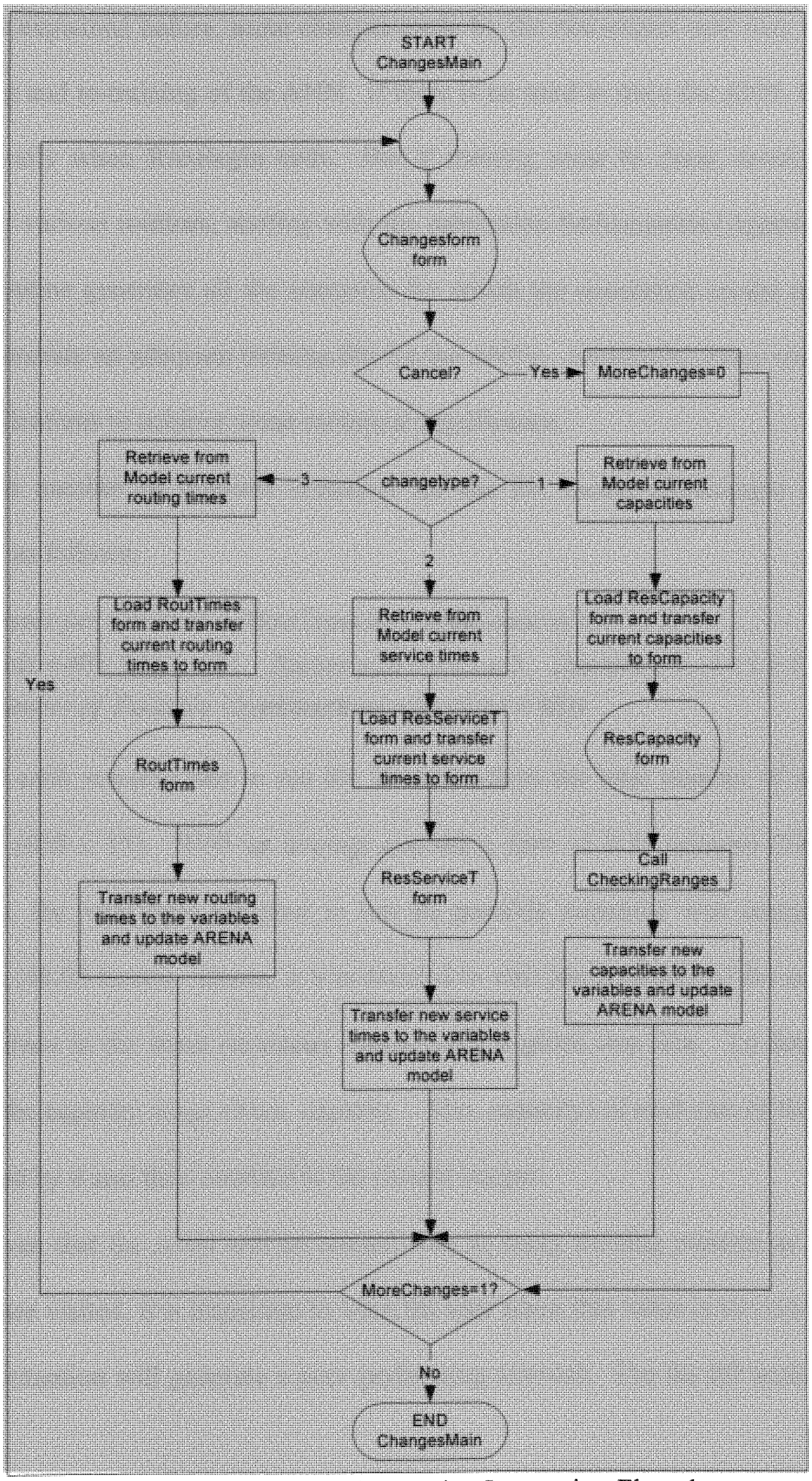

START ChangesMain
Changesform form
Cancel?
Yes
MoreChanges=0
changetype?
3
1
2
Retrieve from Model current routing times
Retrieve from Model current capacities
Retrieve from Model current service times
Load RoutTimes form and transfer current routing times to form
Load ResServiceT form and transfer current service times to form
Load ResCapacity form and transfer current capacities to form
RoutTimes form
ResServiceT form
ResCapacity form
Call CheckingRanges
Transfer new routing times to the variables and update ARENA model
Transfer new service times to the variables and update ARENA model
Transfer new capacities to the variables and update ARENA model
Yes
MoreChanges=1?
No
END ChangesMain

Figure 14: ChangesMain Routine Interaction Flowchart

As explained before, some changes made by the non-expert user may require the re-building and re-training of the ANN. If there is no need to train the ANN, the process ends basically when RunBegin ends. But if training must be done, several ARENA events call various routines, starting with the subroutine SetScenarios (Appendix D.2.2). This subroutine generates all the scenarios for which the simulation model must run in order to generate an adequate training file for the ANNs. Arbitrarily, every scenario runs for 15 replications to achieve some statistical significance.

The number of possible scenarios for the ER model (Sevimoglu 2002) was calculated as follows:

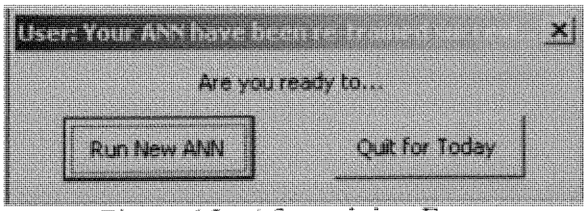
$$Levels\ per\ resource = MaxCapacity - MinCapacity + 1$$
$$Beds = 40 - 10 + 1 = 31 \quad Nurses = 10 - 6 + 1 = 5 \quad Doctors = 4 - 1 + 1 = 4$$
$$Number\ of\ Possible\ Combinations = 31 x 5 x 4 = 620$$

As each scenario had to run for 15 replications the total number of replications was 9300 (620 x 15). With an average execution time of 0.31 minutes (Table 4) per replication, the total time required to run all possible scenarios was computationally inefficient (48.05 hours). Hence, it was decided to visit just a sample of the scenarios by picking values in the input parameter range, using a given step size. To determine the step size, the capacity range (Max. capacity – Min. capacity) of each one of the resources was divided by 4 and then rounded to the closest integer.

Given that the ANN will be trained with as many training patterns as number of scenarios are visited, it is important to warn the user when this value is too low (i.e. < 40) since this situation will generate a poor performing network. For the BP architecture, as the one used in this prototype, the resulting model is better if more training patterns are used.

At this point the event RunBegin ends. The next ARENA events of importance for this user are: RunBeginReplication, RunEndReplication, and RunEnd. The RunEndReplication event, is programmed to calculate the average of the measures of performance for every 15 replications, and stored them in an ASCII file to feed the ANN (Appendix D.3.3). The CheckReplications routine varies the level of the resources every 15 replications and records the results in the ASCII file, 100% transparently to the user (Figure 16).

Once the training data have been generated, the training occurs using the subroutine NN2Training (Appendix D.4.1). This subroutine invokes all the actions required for the training task: to import the training file, to extract the testing set, to define the variables, to train the networks, and to generate the ".def" and the ".vb" file for each one of the networks. Upon completing the training of the macro ANN, the user is informed and given the option to interact with the newly trained networks, in a similar way as it was explained at the beginning of this section (Figure 15).



Are you ready to...

Figure 15: Aftertraining Form

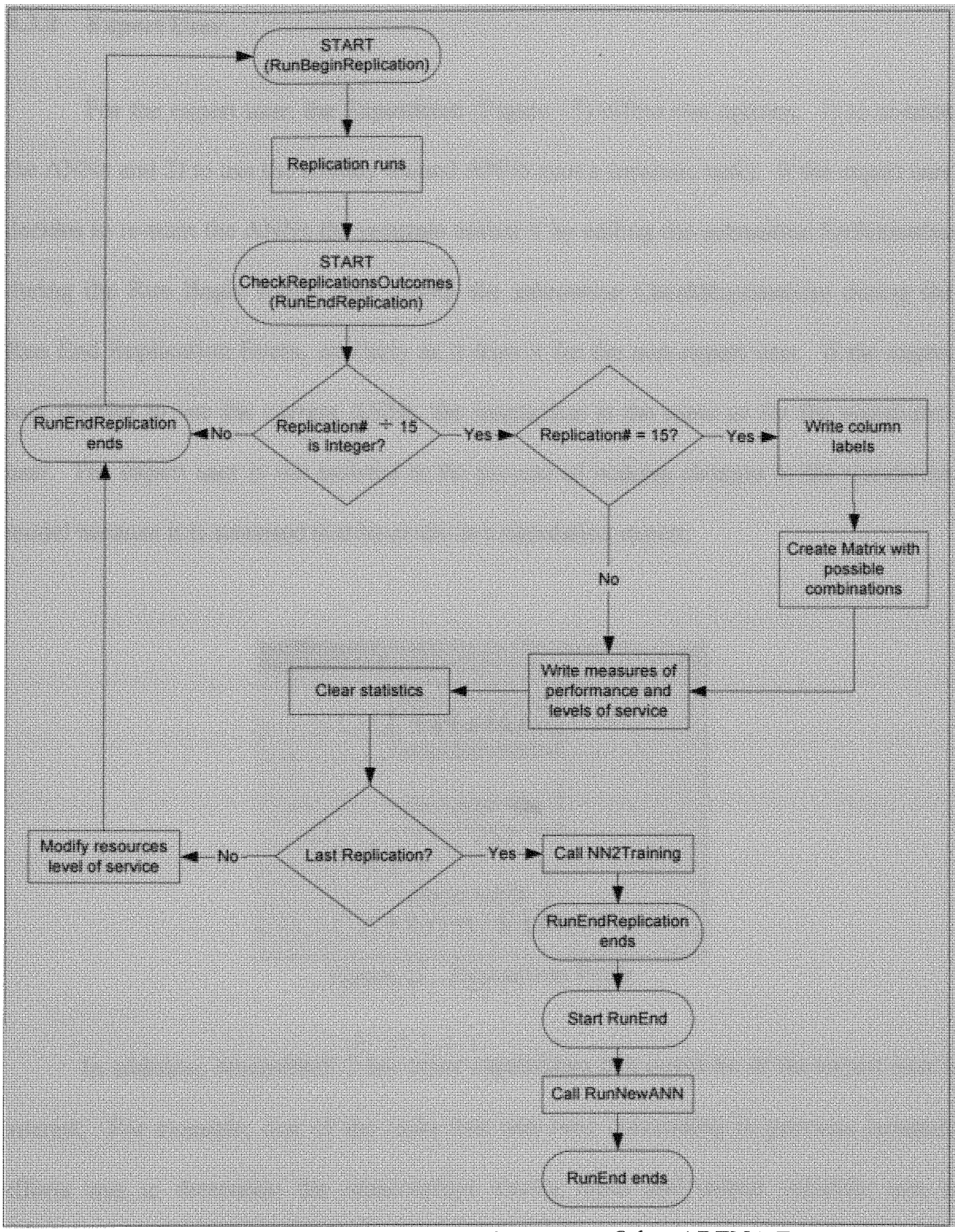

START (RunBeginReplication)
Replication runs
START CheckReplicationsOutcomes (RunEndReplication)
RunEndReplication ends
No
Replication# ÷ 15 is integer?
Yes
Replication# = 15?
Yes
Write column labels
Create Matrix with possible combinations
No
Clear statistics
Write measures of performance and levels of service
Modify resources level of service
No
Last Replication?
Yes
Call NN2Training
RunEndReplication ends
Start RunEnd
Call RunNewANN
RunEnd ends

Figure 16: Flowchart of the Interaction among Other ARENA Events

4.3.2 Expert User

For the expert user, the Expertform (Figure 17) offers two options: 1) to re-train the ANNs and 2) to use the already trained ANNs (not a common task). If the expert user decides to re-train the ANNs the program assists it by calling the subroutine SetScenarios during the Run Begin Event, and calling the subroutine CheckReplications during the Run End Replication Event, similarly as it does it for the non-expert user. If the expert user decides to use the current ANNs, the interaction is the same as for the non-expert user. The expert user is not assisted in the process of making changes to the simulation model because it is assumed that he or she is a simulation expert.

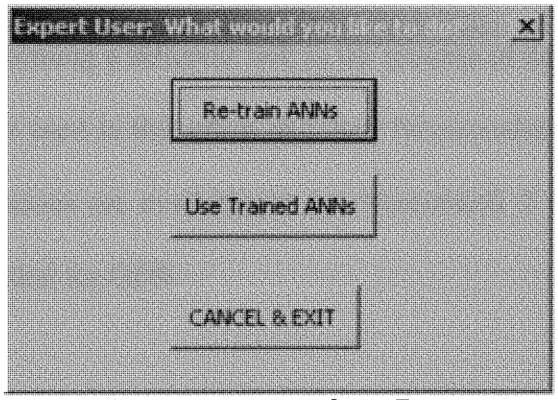


Figure 17: Expertform Form

In addition, NeuroShell 2 has other capabilities that could be of the expert user's interest. For example, one of the tasks that was time consuming in previous research efforts was to determine the relationships among input and output variables into a mathematical formula. In the past, Linear Regression was used for this purpose; however, it has been suspected that the relations among the variables may not necessarily be linear. By using the GMDH architecture instead of the BP one, it is possible to find mathematical equations that define each output variable (one at the time) as a function of the input variables, with indicators as to which variables are the most and least significant. Within this architecture different levels of non-linearity can be used. It is important to highlight that NeuroShell 2 does not recommend using these equations instead of the results obtained directly from firing the ANN because the loss of accuracy. Yet, this option can be a good approximation for situations where the NN software is not available to the user.

4.4 NEUROSHELL 2 APPLICATIONS USED

The NeuroShell 2 package is designed as a group of different applications; each one is in charge of performing a different task in the process of designing and building an ANN. The applications used in this effort and their functionalities are listed in Table 3. Although other applications are available in the NeuroShell 2 software package, only those in Table 3 are needed for GDS.

Table 3: NeuroShell 2 Applications

Application	Functionality
Impascii.exe	This application imports a data file in ASCII format.
Netinput.exe	It handles the definition of the independent and dependent variables (Inputs and Outputs).
Testset.exe	This application extracts the test set from the total data sample.
Begtrain.exe	This application is in charge of the learning process (training) for the beginners' module and it uses BP network architectures
Dllprime.exe	This application is in charge of generating the ".def" file, which is used to run the trained ANN from a different application.
Srcgen.exe	It generates the VB code to be included in the VBA editor of ARENA 5.0.

CHAPTER 5

EXPERIMENTATION AND ANALYSIS

This chapter describes the experiments performed with the prototype and their results. Similarly, it presents testing done with variations of this prototype. Other integration attempts are also discussed as well as the experiment performed with a terminating system for comparative purposes.

The experimentation presented in this Chapter is intended to evaluate the adequacy of NN for GDS by answering the following questions:

1) Can Neural Networks handle better the multivariable search space in the context of Goal Driven Simulation?
2) How difficult is it to train a Neural Network?
3) How difficult is it to implement Neural Networks-Base Goal Driven Simulation? Is it more efficient than previous approaches to Goal Driven Simulation?

5.1 Handling of Multivariable Search Space

Implementing the ANN technology does not have any restrictions regarding the number of variables in its input and output vectors. The experimental prototype presented in Chapter 4 was built using 6 input variables and 3 output variables; however, this number can be increased or decreased with minor changes in the VBA subroutines supporting the prototype. For instance, a second prototype was designed for the terminating system (Foundry) used by Jones (1999). This prototype was built using the original prototype as a template and the time it consumed was 3 hours.

The description of the system modeled with the new prototype is given in Appendix C. The prototype of this system has 4 output variables (Number of lathes, Number of Cover Grinders, Number of Ring Unloaders, and Number of Forklifts) and 8 input variables (Average Time in Queue and Utilization for the stations represented by the output variables). This increment in the number of inputs and outputs did not have any effect on the principles of the integration framework and most of the original prototype was re-used. In fact, most of the time invested in designing this new prototype was consumed on trivial changes such as user forms and new labeling of the variables, while the changes to the main subroutines in charge of training and building the NN were completed in minutes. The VBA Subroutines used in the new prototype are included for reference in Appendix F. It is evident that NN technology does not have major impediments to handle the multivariable search space. Moreover, because of its ability of learning by example, the ANN can explore a bigger selection of scenarios without the generation rules and with no further analysis from the user.

In addition, since the designing and training process is simple, different version of the same network can be built in order to respond to different input scenarios, this feature adds even more flexibility to the integration approach. The only constraint on increasing the number of variables is time. Increasing the number of input variables used in the simulation model increases the number of possible combinations (scenarios) and, therefore, increases the number of replications. For example, the Foundry system with 4 output variables could have a total number of combinations ranging from 16 (2^4) to 1296 (6^4), which may not be computationally efficient due to time constraints.

According to Sevimoglu (2002), adding a variable requires an extensive Response Surface Methodology (RSM) study of the impact of the variable on the outputs to develop decision rules; then, these rules must be manually coded in VB or similar high level programming language. All of these activities would take days, whereas adding a variable when using ANN is a matter of hours. Therefore, despite the lack of comparative data, it is possible to conclude, by common sense, that the use of ANN technology for GDS facilitates the handling of a multivariable search space.

5.2 TRAINING NN FOR GDS

To determine how difficult it was to train a NN, in the context of GDS, it was necessary: 1) to measure the time required to re-train and re-build a macro ANN, and 2) to assess complexity of automating the training task. To measure the time for training, the process was divided in two parts: a) the production of the training data, which consisted of running the simulation model for a set number of replications, and b) the process of importing the data into NeuroShell 2, defining the variables, extracting the testing set, training the networks and finally generating the ".def" and ".vb" files. A total of 10 trials were ran using a desktop computer with a Pentium 4 processor running at 2.4 GHz and with 256 MB of RAM memory (Table 4).

Table 4: Training Times for the Experimental Prototype

Trial	**Combinations**	**Number of Replications**	**Duration Part I (minutes)**	**Duration Part II (minutes)**
1	100	1500	468	*
2	125	1875	639	*
3	120	1800	436	*
4	72	1080	410	225
5	100	1500	425	355
6	100	1500	469	191
7	100	1500	569	169
8	125	1875	441	191
9	100	1500	642	287
10	130	1950	380	181
Average	107.2	1608	487.9	228.4
	Minutes/Replication		0.3108	

*Lost data due to operating system problems.

From the results (Table 4), 95% Confidence Intervals (CI) were built for the following variables (Table 5): average time per replication, average number of replications, and average training time for each one of the two parts. Details on the construction of these CI are included in Appendix G.1.

Table 5: 95% CI for Training ANN (minutes)

Variable	**Lower Bound**	**Average**	**Upper Bound**
Time per Replication	0.2582	0.3108	0.3634
Number of Replications	1418.21	1608	1797.8
Part I: Generation Training File	420.26	487.9	555.54
Part II: Actual Training ANN	165.2	228.4	291.66

From the 95% CI for the average time spent per replication, it is possible to observe the small variation that this variable presents: 95% of the observations are within a range of 6 seconds. This was an expected outcome since all the runs are executed using ARENA, without user intervention, and by using the same model every time. On average, a replication takes 0.3108 minutes (19 seconds).

For the number of replications, the variability is high with a range of almost 400 replications. This was also an expected outcome given that the number of combinations for one system's model can go from 8 combinations (2 possible values for each resource, 2^3) to 216 combinations (6 possible values for each resource, 6^3). Since the number of replications depends directly on the number of combinations, the number of replications could go from 120 (8x15) replications to 3240 (216x15) replications.

The training time for part I depends on two variables, the average time per replication and the number of replications. The variation on the number of replications is considerably high; therefore, the variation for the average training time part I is also high. On average, it takes 487.9 minutes to run all replications to re-train the macro ANN.

Finally, the CI for the average training time part II shows a variation of more than an hour. This was not an expected outcome since NeuroShell 2 always trained the same number of networks (63) and all the variables of the training process are kept constant. The only possible explanation could be the variation in the training file size. The size of this file depends on the total number of possible combinations of the system. Other presumable reason for this variation could be the behavior of the *sendkeys* statement that is managing the execution of the program during the second part of the re-building and re-training process. On average, it takes 228.4 minutes to execute part II of the training process.

Based on these results, 97.5% of the training will take under 14 hours. Although it seems a long time, two issues must be kept in mind: a) training will occur only on special occasions, and b) one of these special occasions is adding variables, which under previous approaches could take days not hours.

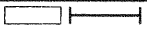
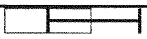
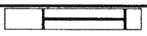
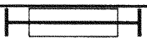
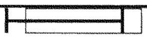
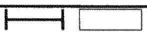
Further, this effort has demonstrated that under the proposed framework, training and using the ANN is a relatively easy task for the end user because under the framework the entire process is 100% seamless. It has been demonstrated that NeuroShell 2 can be fully controlled from the ARENA environment. The programming capabilities to control NeuroShell 2 are standard capabilities to any VB or VBA implementation (Shell, sendkeys, etc.); hence, this feature of the framework is not ARENA dependant. Furthermore, even though we have taken advantage of the various events that an ARENA model experiences, most simulation packages have equivalent features (Start run, End replication, etc.). The subroutines herein developed are robust and offer a wide range of possibilities to the user; hence, there is no need to develop new algorithms or to program new subroutines when the system is modified. Therefore, we can confidently say that based on experimental results as well as in knowledge of simulation packages technology, training an ANN for GDS is relatively easy task.

5.3 LEVEL OF DIFFICULTY TO IMPLEMENT NN-BASED GDS

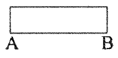
In order to see if NN-Based GDS is more efficient than other approaches, the effort done by Sevimoglu (2002) was used as point of comparison. In her research to develop a multivariable heuristic for GDS, she had to determine the effect on the output parameters per changes in the input parameters. Due to time constraints and the high level of work involved in this task, she worked with only two goals simultaneously.

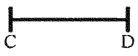
Since there are 6 possible goals to select from (Average Queue Time Nurses, Doctor, and Beds, and Utilization of Nurses, Doctors, and Beds), it was necessary to determine all the possible ways to select 2 goals out of the 6, $C_2^6=15$. For each one of the 2 variables, 6 different scenarios may occur (Table 6, Sevimoglu 2002).

Table 6: Possible Scenarios for each Goal

Scenario	Condition
1. 	B < C
2. 	C < B and B < D
3. 	A < C and B > D
4. 	A > C and D > B
5. 	A > C and D < B
6. 	C < B

Where:



: Model (Model 95% CI)


: Goal (User Goal Range)

Using the multiplication rule, if each one of the variables has 6 scenarios and there are 15 possible pairs, the total number of rules she had to derive was 540 (6x6x15). In order to derive these rules, she had to run the simulation model for each one of the 540 configurations. Estimating that she ran each configuration for 15 replications (as in this effort), the total number of replications was 8100. With an average time per replication of 0.31 minutes (as obtained from the experiment in Section 5.2), the total time for this task was 42 hours just to generate the data for the RSM. This value already exceeds the worst case scenario of 14 hours under the NN-Based approach.

In order to extend her results to three variables, the possible number of ways to select 3 goals out of 6 is equal to $C_3^6=20$. With three variables the total number of rules to be derived increase to 4,320 (6x6x6x20), which would require 64,800 replications or

334.8 hours (14 days). These quantities do not include the amount of time needed to conduct the analysis (RSM and rule derivation), which anecdotally, we know from Sevimoglu that it takes a couple of days.

With these results is easy to realize that NN-Based GDS is more efficient than several of the previous approaches since reduces the amount of time and effort required to embed new knowledge into the system. Also, it is flexible to handle the multivariable search space and robust to adapt to changes in the modeled system without requiring user interaction.

5.4 Other Lessons Learned

5.4.1 The Issue of Infeasibility

The implementation of NN-Based GDS was considerably simple from the integration framework point of view. However, an issue remains unsolved: *How to determine if a goal set is infeasible?*

When ANN were proposed for GDS, we had hoped to also be able to resolve the issue of infeasibility of the goal set. We expected to have a practical and efficient experimental tool to explore the neighborhood outside the allowable range of the input parameters, and we do have such tool. In fact, NeuroShell 2 provides the necessary equations relating I/O (Table 8).

NeuroShell 2 generates ANN that always gives an answer even when one does not exist. NeuroShell 2 rounds its answers that fall outside the training range to either the lower or the upper bound; therefore, the end user is not sure of the validity of the response given by the ANN when its answer coincides with the boundaries of the parameters. To overcome this ambiguity, we attempted to use the VB code generated by NeuroShell 2 (Appendix E), and insert it into the VBA of the experimental prototype. However, NeuroShell 2 generates one of these files for each one of the 63 networks, which means that ARENA must select the appropriate file, at run-time, automatically. But, the automation of the process is not possible with VBA for ARENA because it lacks the #insert or #include statement, and it must compile all code before running.

An additional attempt to detect infeasibility was done by training the networks for a range of resources' capacities wider than the one specified by the user. In this way, when NeuroShell 2 rounds its answer to the upper or lower bound, the framework will be aware of the infeasibility. However, this approach is still not capable of giving the user the actual answer as it still gives the rounded approximation.

Training the ANN for a range two units wider than the original range *may* increase the time consumed due to the number of combinations increasing, but it may not, as shown in the example for the ER system where the increment on time consumption is zero (Table 7). The maximum increment in the number of combinations that any variable can present is two additional combinations. If the number of combinations for all variables increases by two in a system of three output variables, the maximum increment in the total number of combination will be:

$$\Delta comb = 2(comb1 * comb2) + 2(comb2 * comb3) + 2(comb1 * comb3) + 4(comb1) + 4(comb2) + 4(comb3) + 8$$

Where,

$\Delta comb$ = Increment in the total number of combinations

$combi$ = Number of combinations variable i

It is important to notice that even though this increment looks considerable high, an increment of two in the number of combinations occurs only in the presence of very narrow ranges; therefore, the values of *comb1*, *comb2* and *comb3* will be small.

Table 7: Number of Combinations Original and Extended Ranges

Variable	Original			Extended		
	Range	**Scenarios**	**Comb**	**Range**	**Scenarios**	**Comb**
Number of Nurses	[6-10]	6 7 8 9 10	5	[5-11]	5 7 9 11	4
Number of Doctors	[1-4]	1 2 3 4	4	[1-5]	1 2 3 4 5	5
Number of Beds	[10-40]	10 18 26 34 40	5	[9-41]	9 17 25 33 41	5
Total Combinations	*100*			*100*		

5.4.2 GMDH Architecture

Additional experimentation was done using the GMDH architecture. The goal was to evaluate the capability of ANN to obtain mathematical equations of the modeled system and to establish the accuracy of these equations. These mathematical equations relate the input and output variables. The GMDH architecture does not require the extraction of a testing set. The learning process stops automatically when NeuroShell 2 determines that no further improvement is possible. To obtain the equations for each one of the 3 outputs variables, each one of them was selected independently as unique output while all the 6 inputs were used everytime to calculate the best formula. Selecting all the inputs enables the GMDH architecture to decide which ones to use (not all inputs have to appear in a particular formula); yet, if inputs are omitted (define as unused) the GMDH architecture cannot use them regardless. A major drawback for this experiment was the infeasibility to automate the generation of the equations, as the application within NeuroShell 2 responsible for this task, **Design.exe,** cannot be controlled using the VB statement *sendkeys*. Nonetheless, the experiment was executed and its results were analyzed.

Table 8 gives the equations obtained for the ER system. The training ranges for this example were: Nurses (6-10), Doctors (1-4), and Beds (10-40). In reading these equations, let

X1 = Wq_n = Average Queue Time Nurses

X2 = ρ_n = Utilization Nurses

X3 = Wq_d = Average Queue Time Nurses

X4 = ρ_d = Utilization Nurses

X5 = Wq_b = Average Queue Time Nurses

X6 = ρ_b = Utilization Nurses

A quick look at these equations clearly shows the complex polynomial nature of the relationship between inputs and outputs of an ER system. This fact was suspected by Correa (1999), Lee (2002), and Sevimoglu (2002) but they were not able to irrefutable make the statement. With the help of NeuroShell 2, using GMDH, it is now easy to ascertain that any linear relationship of I/O for an ER system is inappropriate.

Table 8: Outputs' Equations Generated by GMDH Architecture

Output	Equations
A	$-0.18*\left(\frac{2*(X3-0.07)}{0.01}-1\right)-0.49*\left(\frac{2*(X4-0.53)}{0.01}-1\right)-0.42+0.72*\left(\frac{2*(X5-24.54)}{288.03}-1\right)+\frac{2*(X6-0.99)}{0.01}-1$
B	$0.89*\left(\frac{2*(X4-0.53)}{0.01}-1\right)^2-0.98*\left(\frac{2*(X5-24.54)}{288.03}-1\right)^2-0.62*\left(\frac{2*(X4-0.07)}{0.01}-1\right)^3$
C	$0.65*\left(\frac{2*(X4-0.53)}{0.01}-1\right)^3-2.8*\left(\frac{2*(X5-24.54)}{28803}-1\right)^3+0.45*\left(\frac{2*(X3-0.07)}{0.01}-1\right)*\left(\frac{2*(X5-24.54)}{28803}-1\right)$
D	$0.78*\left(\frac{2*(X4-0.53)}{0.01}-1\right)*\left(\frac{2*(X5-24.54)}{288.03}-1\right)$
E	$2.6*\left(\frac{2*(X3-0.07)}{0.01}-1\right)*\left(\frac{2*(X4-0.53)}{0.01}-1\right)*\left(\frac{2*(X5-24.54)}{288.03}-1\right)-0.67*\left(\frac{2*(X6-0.99)}{0.01}-1\right)^3$
F	$0.87*\left(\frac{2*(X3-0.07)}{0.01}-1\right)*\left(\frac{2*(X6-0.99)}{0.01}-1\right)-0.58*\left(\frac{2*(X2-0.76)}{0.01}-1\right)*\left(\frac{2*(X5-24.54)}{288.03}-1\right)$
G	$-0.2-\left(\frac{2*(X3-0.07)}{0.01}-1\right)-0.25*\left(\frac{2*(X6-0.99)}{0.01}-1\right)+0.4*\left(\frac{2*(X1-0.06)}{0.01}-1\right)$
H	$0.74*\left(\frac{2*(X2-0.76)}{0.01}-1\right)+0.38*\left(\frac{2*(X4-0.53)}{0.01}-1\right)+0.9*\left(\frac{2*(X2-0.76)}{0.01}-1\right)^2$
I	$-0.84*\left(\frac{2*(X4-0.53)}{0.01}-1\right)^2-1.1*\left(\frac{2*(X1-0.06)}{0.01}-1\right)^3-0.77*\left(\frac{2*(X1-0.06)}{0.01}-1\right)*\left(\frac{2*(X2-0.76)}{0.01}-1\right)$
J	$-0.67*\left(\frac{2*(X1-0.06)}{0.01}-1\right)*\left(\frac{2*(X4-0.53)}{0.01}-1\right)+1.8*\left(\frac{2*(X2-0.76)}{0.01}-1\right)*\left(\frac{2*(X4-0.53)}{0.01}-1\right)$
K	$-2.2*\left(\frac{2*(X1-0.06)}{0.01}-1\right)*\left(\frac{2*(X2-0.76)}{0.01}-1\right)*\left(\frac{2*(X4-0.53)}{0.01}-1\right)+0.49*\left(\frac{2*(X6-0.99)}{0.01}-1\right)^2$
L	$1.3*\left(\frac{2*(X3-0.07)}{0.01}-1\right)*\left(\frac{2*(X6-0.99)}{0.01}-1\right)$
M	$0.48*\left(\frac{2*(X3-0.07)}{0.01}-1\right)+0.2+0.41*\left(\frac{2*(X4-0.53)}{0.01}-1\right)-0.73*\left(\frac{2*(X5-24.54)}{288.03}-1\right)$
N	$-1.9*\left(\frac{2*(X6-0.99)}{0.01}-1\right)+0.57*\left(\frac{2*(X6-0.99)}{0.01}-1\right)^2+1.2*\left(\frac{2*(X5-24.54)}{288.03}-1\right)^3$
O	$1.5*\left(\frac{2*(X6-0.99)}{0.01}-1\right)^3-1.3*\left(\frac{2*(X3-0.07)}{0.01}-1\right)*\left(\frac{2*(X6-0.99)}{0.01}-1\right)-0.82*\left(\frac{2*(X1-0.06)}{0.01}-1\right)^2$
P	$-0.32*\left(\frac{2*(X6-0.99)}{0.01}-1\right)^2-0.64*\left(\frac{2*(X6-0.99)}{0.01}-1\right)^3+1.3*\left(\frac{2*(X1-0.06)}{0.01}-1\right)*\left(\frac{2*(X6-0.99)}{0.01}-1\right)$
Q	$0.52*\left(\frac{2*(X3-0.07)}{0.01}-1\right)^3-0.67*\left(\frac{2*(X1-0.06)}{0.01}-1\right)*\left(\frac{2*(X3-0.07)}{0.01}-1\right)-0.3*\left(\frac{2*(X2-0.76)}{0.01}-1\right)^2$
Number Beds	$30*\left[\frac{(A+B+C+D+E+F)+1}{2}\right]+10$
Number Doctors	$3*\left[\frac{(G+H+I+J+K+L)+1}{2}\right]+1$
Number Nurses	$4*\left[\frac{(M+N+O+P+Q)+1}{2}\right]+6$

From the last three rows of Table 8, it is possible to observe that NeuroShell 2, under GMDH, establishes a probability factor to each combination of inputs, and then it uses the Uniform Distribution to determine the output value; therefore, these 3 equations can be generically represented as follows:

Let,

φ_i = Output variable i

i = (1,2,3) = (number of beds, number of doctors, number of nurses)

a_i = Lower bound of permissible range for output i

b_i = Upper bound of permissible range for output i

π_i = Probability factor for output i

$$\pi_1 = \frac{A+B+C+D+E+F+1}{2} \qquad \pi_2 = \frac{G+H+I+K+L+1}{2}$$
$$\pi_3 = \frac{M+N+O+P+Q+1}{2}$$

Then $\varphi_i = a_i + \pi_i(b_i - a_i)$, which is the inverse transformation relation to generate a uniformly distributed random variable.

A hypothesis test was designed to assess the accuracy of these equations:

$$H_0 : \delta = 0$$
$$H_1 : \delta \neq 0$$

Where $\delta = \mu_{NN} - \mu_{GMDH}$

μ_{GMDH} = Output given by the GMDH equations

μ_{NN} = Output given by the trained NN

Since the ANN are sensitive to the initial weights of their connections, five versions of the same ANN were created using different random seeds to generate the initial weights. Then, their performance was averaged and compared against the GMDH equations.

The results of the *t*-tests are summarized in Figure 18. For the variable Number of Beds, the 95% CI on the difference (-20.6330, -6.4337) does not contain zero;

therefore, we reject the null hypothesis (t=3.899>$t_{0.025,29}$=2.045). Since the CI falls below zero, the GMDH equations give higher response values than the trained ANN. A similar situation occurs for the variable Number of Nurses (95% CI→ (-1.8310, -0.5690) and t=3.890>$t_{0.025,29}$=2.045). For the variable Number of Doctors, the 95% CI on the difference (0.0427, 0.9573) does not contain zero; hence, we reject the null hypothesis (t=2.236>$t_{0.025,29}$=2.045). But for this variable, the 95% CI falls above zero, which means that the GMDH equations give lower response values than the trained ANN.

In summary, there *is* enough statistical evidence to conclude that the results obtained with a trained ANN and with the GMDH equations are different. Deciding which one is better can only be done if each one of these two alternatives is tested against the "real" output (that from the simulation model), which is done in the next section.

Paired Samples Test

		Paired Differences							
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper	t	df	Sig. (2-tailed)
Pair 1	BEDSNN - BEDSGMDH	-13.5333	19.01312	3.47131	-20.6330	-6.4337	-3.899	29	.001
Pair 2	DOCSNN - DOCSGMDH	.5000	1.22474	.22361	.0427	.9573	2.236	29	.033
Pair 3	NURNN - NURGMDH	-1.2000	1.68973	.30850	-1.8310	-.5690	-3.890	29	.001

Figure 18: t-tests for the Difference of Means

Table 9: Average Difference for Both Approaches

Trial	Number of Beds			Number of Doctors			Number of Nurses		
	μ_{NN}	μ_{GMDH}	d_i	μ_{NN}	μ_{GMDH}	d_i	μ_{NN}	μ_{GMDH}	d_i
1	25	34	-9	2	2	0	8	9	-1
2	26	20	6	3	2	1	8	8	0
3	27	40	-13	3	1	2	7	9	-2
4	23	33	-10	2	1	1	9	6	3
5	23	41	-18	2	3	-1	9	8	1
6	27	35	-8	3	1	2	7	7	0
7	24	35	-11	2	4	-2	8	8	0
8	27	10	17	2	2	0	8	9	-1
9	25	28	-3	3	1	2	8	9	-1
10	24	40	-16	2	1	1	8	11	-3
11	25	36	-11	2	2	0	8	7	1
12	23	23	0	2	0	2	8	10	-2
13	24	57	-33	2	1	1	8	11	-3
14	27	15	12	2	2	0	8	10	-2
15	24	94	-70	3	1	2	8	9	-1
16	24	25	-1	2	3	-1	8	9	-1
17	25	51	-26	2	2	0	8	11	-3
18	23	37	-14	2	1	1	8	10	-2
19	24	83	-59	2	1	1	8	12	-4
20	24	49	-25	2	0	2	8	9	-1
21	27	70	-43	2	0	2	8	10	-2
22	24	20	4	2	2	0	8	10	-2
23	23	18	5	2	2	0	9	12	-3
24	24	44	-20	2	0	2	8	12	-4
25	24	31	-7	2	1	1	8	11	-3
26	24	36	-12	2	2	0	8	9	-1
27	26	52	-26	3	3	0	8	6	2
28	24	24	0	2	3	-1	8	9	-1
29	25	36	-11	2	4	-2	8	9	-1
30	24	28	-4	3	4	-1	8	7	1
Average			*-13.5069*			*0.699231*			*-0.65138*
STD			*19.01393*			*1.096072*			*1.606528*

5.4.3 Accuracy of the NN-Based GDS

To assess the accuracy of the trained ANN in relation to the responses from the simulation model, a hypothesis test is required. Given that in Section 5.4.2, it was concluded that the results obtained with a trained ANN differ from the ones obtained with the GMDH equations, it was decided to perform hypothesis tests for both options, with the hope of showing if one of the approaches is better than the other.

$$H_0 : \delta = 0$$
$$H_1 : \delta \neq 0$$

Where $\delta = \mu_{NN} - \mu_S$ *or* $\delta = \mu_{GMDH} - \mu_S$

μ_S = Response given by the simulation model

μ_{GMDH} = Response given by the GMDH equations

μ_{NN} = Response given by the trained NN

Independent hypothesis tests were performed for each one of the output variables. The results of the t-tests are summarized in Figure 19.

Paired Samples Test

		Paired Differences							
					95% Confidence Interval of the Difference				
		Mean	Std. Deviation	Std. Error Mean	Lower	Upper	t	df	Sig. (2-tailed)
Pair 1	BEDSNN - BSIM	.3000	11.01144	2.01040	-3.8117	4.4117	.149	29	.882
Pair 2	BEDSGMDH - BSIM	13.8333	22.32686	4.07631	5.4963	22.1703	3.394	29	.002
Pair 3	DOCSNN - DSIM	-.0667	1.08066	.19730	-.4702	.3369	-.338	29	.738
Pair 4	DOCSGMDH - DSIM	-.5667	1.67504	.30582	-1.1921	.0588	-1.853	29	.074
Pair 5	NURNN - NSIM	.3333	1.49328	.27263	-.2243	.8909	1.223	29	.231
Pair 6	NURGMDH - NSIM	1.5333	1.83328	.33471	.8488	2.2179	4.581	29	.000

Figure 19: *t*-test for Accuracy

For the three variables (Number of Beds, Number of Nurses, and Number of Doctors), the 95% CI on the difference with the trained ANN contain zero; therefore, we fail to reject the null hypothesis on the three cases, which means that the ANN is given statistically the same responses as the simulation model alone. On the other hand, the 95% CIs on the difference with the GMDH equations do not contain zero in the case of

Number of Beds and Number of Nurses; hence, we reject the null hypothesis. The latter means that the GMDH equations are not matching the results of the simulation alone. Further, since these CI fall above zero, the GMDH equations give higher values than the "real" ones. The only case in which the GMDH equations give correct answers is for the variable Number of Doctors.

Based on the results of the hypothesis tests for each one of the output variables, it is concluded that the accuracy level observed with the trained ANN is superior than the one observed with the GMDH equations. It is worth noting that the power of the test for the GMDH equations is stronger than the one for the trained ANN.

5.4.4 Importance of Factors

An interesting result that NeuroShell 2 indicated is that the variable Number of Nurses affects all the measures of performance. In her work, Correa (1999) had also concluded that. The findings done by NeuroShell 2 provide additional strength to Correa's statement. NeuroShell 2 indicates the significance of the variables via contribution factors graphs. The definition of *Contribution factor* was presented in Section 3.2.

CHAPTER 6

RESEARCH EFFORT SUMMARY

6.1 Accomplishment of Proposed Objectives

The goal of this investigation was to develop a methodology to integrate ANN technologies to simulation models to achieve a robust and flexible GDS framework. Accordingly, a framework for the integration between ANN and GDS has been designed. An experimental prototype has been built to model its concepts and to explore its worthiness by experimentation. As a result, this effort presents the following integration methodology:

1) Build and validate the simulation model.
2) Establish ranges of controllable input parameters and measures of performance.
3) Establish combinations of input parameters to produce simulation data to train ANN. Use variable step size so that the total number of levels for each input parameter does not exceed 6.
4) Run the simulation for each combination, 15 replications each. Produce a data file (ASCII, Excel, etc.)
5) Feed the resulting file to ANN software and train the networks, the number of networks will be equal to (2inputs – 1). Use BackPropagation architecture to design the ANN and train it for 45 seconds. Set the LR, momentum, number

of hidden neurons, and pattern selection, according with your knowledge of the system, if not familiar with ANN set to the default values for a complex system.

6) ANN-Based GDS is ready to use.

It had been proposed (Section 1.2):

1) *Investigate Neural Networks in the context of simulation.* NeuroShell 2 was self-taught, and stand-alone experimentation with it demonstrated that it offers many advantages for GDS. Among these are:
   a) Once the ANN is trained, the simulation model does not need to be run again to answer user's queries.
   b) The ANN can be totally embedded in another application.
   c) The ANN knowledge can be implemented without using NeuroShell 2 as it generates a set of equations that can be implemented in any programming language. However, the accuracy achieved with the equations is less than the one achieved with the trained ANN.
2) *Develop experimental models.* One of the models (ER) was used to test the integration routines. The other model was used to test the portability of these routines from one model to another.
3) *Design experiments.* Once the framework was designed, different experiments were performed to determine the level of difficulty of implementing this approach as well as the efficiency of training an ANN for GDS. The results from these experiments were then compared with the

results obtained by previous approaches to improve GDS. Conclusions of these experiments are summarized in the next section.

6.2 Summary of Results

This research has proven that is possible to integrate ANN and GDS, and that it is more efficient than previous heuristic approaches.

This research started with two questions (Section 1.1):

1) What would happen if one or more structural characteristics of the model change?
2) Would the knowledge acquired by the Neural Network allow an early detection of infeasibilities?

This investigation has answered these questions as follows:

What would happen if one or more structural characteristics of the model change? In previous approaches, it was required to make major modifications to the already built system, i.e. generation of new decision rules. Using NN-Base GDS, the knowledge of the NN must be modified according to the changes made to the system. The learning process (transfer of knowledge) takes place automatically, and it does not require any expertise from the modeler.

Would the knowledge acquired by the Neural Network allow an early detection of infeasibilities? ANN rounds its outputs to the lower or upper bound of the training range; hence, detection of infeasibility is not straightforward task using NeuroShell 2. We proposed two untested approaches that may take advantage of ANN:

a) To generate the VB code corresponding to the trained ANN, and manually make the required modifications to avoid rounding.

b) To train the ANN for a wider range than the originally training range. This approach will detect the infeasibility, but it will not give an exact response.

Summarizing, the following conclusions and deliverables have been obtained from this effort:

1) It is possible to teach the ANN the existing relations between the inputs and outputs variables of a simulation model.

2) A working experimental prototype has been built using the designed integration framework. The prototype enables the user to select any combination of input variables or all of them. Also, it advices the user when no re-training is required, which represents savings on time.

3) A great reduction in designing time, and effort level from the modeler standpoint was achieved; the time consumed was reduced from 42 hours of simulation time and several days of RSM analysis (derivation of rules) to at most 14 hours of simulation and NeuroShell 2 running time with no user interaction. This research can be extended to larger number of variables with minor changes to the existing prototype.

4) The designed framework can be applied to terminating or non-terminating systems since the I/O pairs are collected at the end of the simulation run. A prototype for Jones' (1999) terminating system was built as test.

5) Regarding the variables of the ER system, it was found that all measures of performances are affected by the variable Number of Nurses. This

result confirms the conclusions reached by Correa (1999) and Sevimoglu (2002) during their research.

6) Using the GMDH approach it was determined that the relations between the input and output variables of the ER system are definitely not linear. This confirms the speculations made by Correa (1999), Lee (2002) and Sevimoglu (2002).

6.2 RESEARCH SIGNIFICANCE

This study marks a breakthrough in the field of GDS by integrating ANN technology. Previous approaches for GDS were based on statistical and forecasting techniques with good results, but the ANN approach appeared to be more promising and it has been proven so.

As ANN is a self-learning system, the relationship between inputs and outputs are established with lesser effort than previous approaches. Sevimoglu (2002) develop a heuristic to establish the relationship between inputs and outputs by using the RSM. While this approach is accurate, it is also time consuming. The time consumed developing a multivariable heuristic for two variables was estimated as 42 hours of simulation time, plus several days conducting the analysis of the RSM to derive the decision rules. Hence, a trade-off must be achieved between time consumption and the accuracy of the results for the GDS methodology. ANN can help achieving this trade-off. Once the prototype has been built, it is estimated that the time required to generate the simulation data and re-train the ANN is 14 hours (847.2 minutes) in the worst-case scenario. It is important to remark that this estimated time corresponds to a model with 3 variables.

Previous research efforts in this area have shown that a change in the modeling conditions requires re-doing a big portion of the model, which implies more time and effort. In this research, it was determined the amount of time required to make the same nature of changes in the ANN-Based GDS model was around 3 hours. These results compared with the ones from previous approaches show considerable reductions.

However, there is one question that remains unanswered, *the automated detection of infeasibilities.*

Although it was possible to automate the detection of infeasibilities, it was not possible to automate the process of finding the values for which the goals would be feasible. Additional programming or the application of other ANN principles can be studied to answer this.

REFERENCES

Alsugair, A. M. and D. Y. Chang, 1994, "A Goal Driven Approach to Discrete Event Simulation," *Proceedings of the 1st Congress on Computing in Civil Engineering*, 515-522.

Centeno, Martha A. and Charlie R. Standridge, 1992, "Databases and Artificial Intelligence: Enabling Technologies for Simulation Modeling," *Proceedings of the 1992 Winter Simulation Conference*, J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson- (eds.), 181-188.

Cho, T. H., 2002, "Embedding Intelligent Planning Capability to DEVS Models by Goal Regression Method," *Simulation*, 78(12), December 2002, 716-730.

Correa, Daisy, 1999, "A Study of Response Surface in Simulation of Emergency Room Systems", *Master's Thesis*, Department of Industrial and Systems Engineering, Florida International University, Miami, FL 33199.

Erraguntla, M., P. C. Benjamin, and Rick J. Mayer, 1994, "An Architecture of a Knowledge-Based Simulation Engine," *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila (eds), 673-680.

Glover, F., J. P. Kelly, and M. Laguna, 1999, "New Advances for Wedding Optimization and Simulation," *Proceedings of the 1999 Winter Simulation Conference*, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans (eds), 255-260.

Hamilton, J. L., E. Micheli-Tzanakou, and R. M. Lehman, 2000, "Neural Networks Trained with Simulation Data for Outcome Prediction in Pallidotomy for Parkinson's Disease," *Proceedings of the 22nd Annual EMBS International Conference*, July 23-28, 1-4.

Isaacson, E., 1998, "SIMNET: A Neural Network Model for Post-Tonal Segmentation," *Society for Music Theory,* Indiana University.

Jones, Jacqueline M., 1999, "On-line Goal Driven Simulation for Terminating Systems," *Master's Thesis*, Department of Industrial and Systems Engineering, Florida International University, Miami, FL 33199.

Kelton, David W., Randall P. Sadowski, and Deborah A. Sadowski, 1998, *Simulation with Arena*, Boston, MA, McGraw-Hill Companies, Inc.

Kilgore, Rick A., 2001, "Open-Source SML and SILK for Java-Based, Object-Oriented Simulation," *Proceedings of the 2001 Winter Simulation Conference,* B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer (eds), 262-268.

Krishnaswamy, C. R., E. W. Gilbert, and M. M. Pashley, 2000, "Neural Network Applications in Finance: A Practical Introduction," *Financial Practice and Education*, 75-84.

Laguna, M. and R. Marti, 2002, "Neural Networks Prediction in a System for Optimizing Simulations," *IIE Transactions,* Vol 34, 273-282.

Lee, Marsha A., 2002, "Comparative Study of the Response Surfaces of two types of Systems," *Master's Thesis*, Department of Industrial and Systems Engineering, Florida International University, Miami, FL 33199.

Panayiotou, C. G., C. G. Cassandras, and W. Gong, 2000, "Model Abstraction for Discrete Event Systems Using Neural Networks and Sensitivity Information," *Proceedings of the 2000 Winter Simulation Conference*, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick (eds), 335-341.

Peñaloza, Gabriela C., 1999, "A Forecasting Based Heuristic for On-line Goal Driven Simulation," *Master's Thesis*, Department of Industrial and Systems Engineering, Florida International University, Miami, FL 33199.

Reed, Russell and Robert Marks, *Neural Smithing*, The MIT Press, 1998.

Reyes, M. Florencia, 1998, "A Heuristic For On-line Assesment of Simulation Output for Goal Driven Simulation," *Master's Thesis*, Department of Industrial and Systems Engineering, Florida International University, Miami, FL 33199.

Reyes, M. Florencia, Martha A. Centeno, Khokiat Kengskool, and Shih-Ming Lee, 1999, "Towards On-Line Goal Driven Simulation," *Proceedings of 25th International Conference on Computers and Industrial Engineering*, S. M. Waly (ed), 466-469.

Rich, Elaine, *Artificial Intelligence*, McGraw-Hill, Inc., 1991.

Russell, C. S., A. S. Elmaghraby, and J. H. Graham, 1992, "An Investigation of a Standard Simulation-Knowledge Interface," *Proceedings of the 1992 Winter Simulation Conference*, J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson (eds), 807-815.

Sanchez, P. J, 1994, "Simulation Statistical Software: An Introspective Appraisal," *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila (eds), 1311-1315.

Schewetman, H., 2000, "Optimizing Simulations with CSIM18/Optquest: Finding the Best Configuration," *Proceedings of the 2000 Winter Simulation Conference,* J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick (eds), 268-273.

Sevimoglu, Tuba, 2002, "A Heuristic For Multivariable Goal Driven Simulation," *Master's Thesis*, Department of Industrial and Systems Engineering, Florida International University, Miami, FL 33199.

Shannon, Robert E. and Subraminian Prakash, 1990, "Goal Directed Simulation Systems," *Proceedings of the 1990 International Industrial Engineering Conference*, 345-350.

Sharda, R. and R. Rampal, 2000, "Neural Networks and Management Science/Operations Research," *Department of Management College of Business Administration,* Oklahoma State University. http://catt.okstat.edu/itorms/guide/nnpaper.html.

Sparks, D. W. and P. G. Maghami, 1998, "Neural Networks for Rapid Design and Analysis," *American Institute of Aeronautics and Astronautics*, 672-680.

Umphress, D. A. and Udo W. Pooch, 1987, "A Goal-Oriented Approach to Simulation," *Simulation Series*, 19 (1), 44-49.

Ward Systems Group, Inc., www.wardsystems.com, March 12th, 2004.

ADDITIONAL REFERENCES

Adam, A. and X. He, 2003, "Backpropagation of Pseudoerrors: Neural Networks that are Adaptive to Heterogeneous Noise," *IEEE Transactions on Neural Networks,* 14 (2), 253-262.

Brown, A. and H. Yang, 2001, "Neural Networks for Multiobjective Adaptive Structural Control," *Journal of Structural Engineering,* February 2001, 203-210.

Chance, F., 1994, "Simulation Statistical Software: An Introspective Appraisal," *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila (eds), 1311-1315.

Klahr, P., W. S. Faught, and G. R. Martins, 1980, "Rule-oriented Simulation," *Proceeding of 1980 International Conference on Cybernetics and Society*, 350-354.

Molina, Luis A., Carlos Gandarillas, and Martha A. Centeno, 1996, "Goal Driven Simulation Intelligent Back Ends: A State of The Art Review," *Proceedings of the 1996 Winter Simulation Conference*, J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain (eds), 734-739.

Setyawati, B., R. C. Creese, and M. Jaraiedi, 2002, "Neural Networks for Univariate and Multivariate Time Series Forecasting," *Proceeding of the 2003 IIE Conference.*

Shannon, Robert E., 1987, "Models and Artificial Intelligence," *Proceedings of the 1987 Winter Simulation Conference*, A. Thesen, H. Grant, W. D. Kelton (eds), 16-24.

Yu, Wen and Xiaoou Li, 2002, "Adaptive Control with Multiple Neural Networks," *Proceedings of the American Control Conference*, 1543-1548.

APPENDICES

APPENDIX A

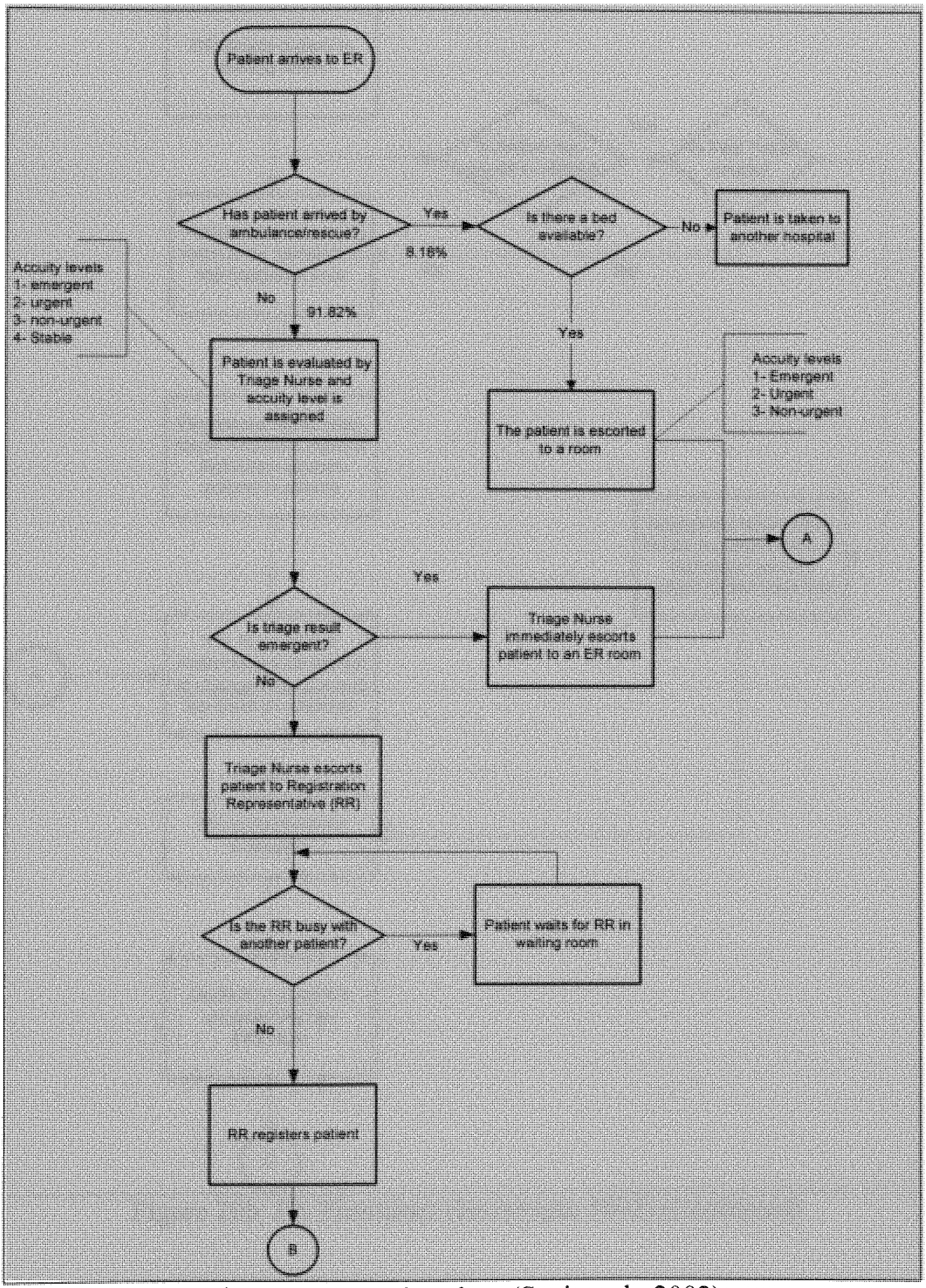
A.1 ANN Software

Neural Studio: Software created by the CATE laboratory at Florida International University. It is used in the Applied Neural Networks class offered by the Electrical and Computer Engineering Department. Although a great tool, it requires some designing knowledge in the area of ANN in order to create a working network. Before any experimentation, the user has to define several parameters such as the number of neurons per layer, number of connections between neurons, number of layers, and the weights between connections. For a first-time user defining these parameters is not easy, and the accuracy of the results obtained with the network depends greatly on the designing skills of the modeler. The package offers the possibility to import training patterns from an ASCII file. The user interface of the main page is too cluttered for the novice.

Neuro Solutions: Software by NeuroDimension-Inc. it possesses a very powerful icon-based graphic user interface, which is very flexible. The software is able to generate C++ code and DLL files, and it allows some level of control from Visual Basic, which is ideal ARENA 5.0. Neuro Solutions includes a Neural Wizard to assist the designing process of the networks; but, it requires a substantial amount of theoretical knowledge on ANN. The type of network must be selected from the starting point and it is not equipped with a beginners' section. For training it is able to import ASCII files, Excel files are not supported. During training the main page gets cluttered and it is difficult to follow the development of the network.

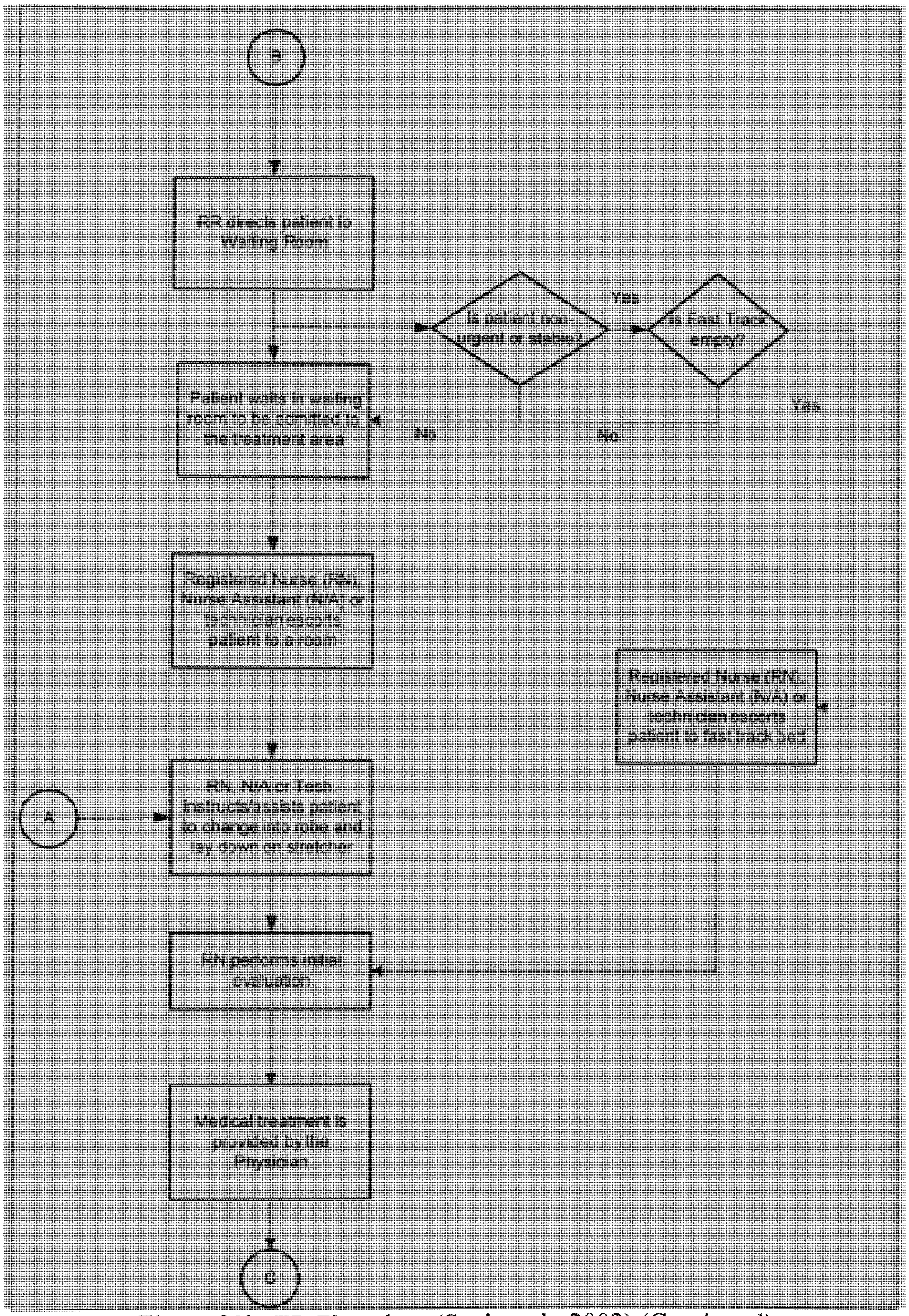
APPENDIX B

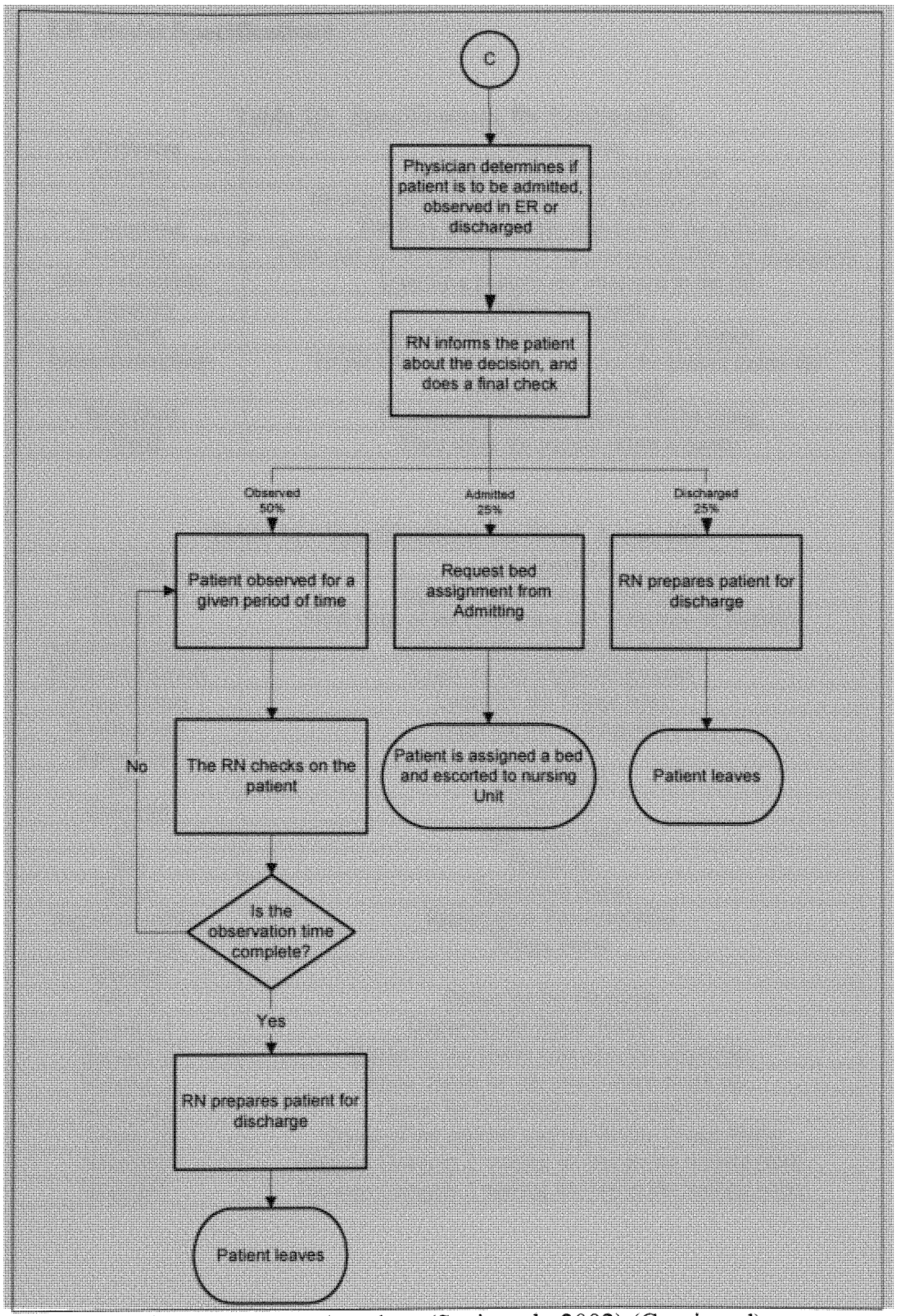
B.1 ER Test Bed



Patient arrives to ER
Has patient arrived by ambulance/rescue?
Yes
8.18%
Is there a bed available?
No
Patient is taken to another hospital
Acuity levels
1- emergent
2- urgent
3- non-urgent
4- Stable
No
91.82%
Yes
Patient is evaluated by Triage Nurse and accuity level is assigned
The patient is escorted to a room
Acuity levels
1- Emergent
2- Urgent
3- Non-urgent
A
Yes
Is triage result emergent?
Triage Nurse immediately escorts patient to an ER room
No
Triage Nurse escorts patient to Registration Representative (RR)
Is the RR busy with another patient?
Yes
Patient waits for RR in waiting room
No
RR registers patient
B

Figure 20: ER Flowchart (Sevimoglu 2002)


B
RR directs patient to Waiting Room
Is patient non-urgent or stable?
Yes
Is Fast Track empty?
Yes
Patient waits in waiting room to be admitted to the treatment area
No
No
Registered Nurse (RN), Nurse Assistant (N/A) or technician escorts patient to a room
Registered Nurse (RN), Nurse Assistant (N/A) or technician escorts patient to fast track bed
A
RN, N/A or Tech. instructs/assists patient to change into robe and lay down on stretcher
RN performs initial evaluation
Medical treatment is provided by the Physician
C
Figure 20b: ER Flowchart (Sevimoglu 2002) (Continued)


C
Physician determines if patient is to be admitted, observed in ER or discharged
RN informs the patient about the decision, and does a final check
Observed 50%
Admitted 25%
Discharged 25%
Patient observed for a given period of time
Request bed assignment from Admitting
RN prepares patient for discharge
No
The RN checks on the patient
Patient is assigned a bed and escorted to nursing Unit
Patient leaves
Is the observation time complete?
Yes
RN prepares patient for discharge
Patient leaves
Figure 20c: ER Flowchart (Sevimoglu 2002) (Continued)

B.2 ER Model Specifications

Table 10: Specifications for Entities-ER

Attributes	Explanation
Arriveby	By what means the patient arrives
Decision	Decision of the doctor: observed, admitted or discharged
Doctortime	The time the physician spends with the patient
Nursetime	Time that the RN spends with the patients (differs by pcategory)
observationtime	Total time patients are observed in the ER
Pcategory	Patient's category
Timein	Time that the patient arrives to the system
tnowenteringloop	To put a timestamp to the time that the patient enters the Observation area logic
Whichbed	To assign the bed the patient will be using
TOAQ	Time when the patient enters a queue

Table 11: Specifications for Work Areas-ER

Stations	Explanation
MyEntrance	Logic for the entrance to the system
MyExit	Final stage of the simulation and collection of statistics
registrationstation	Logic for the registration process
TreatmentStation	Logic for the treatment process
TriageStation	Logic for the Triage Area
Waitingroom	Logic for the waiting area
Resources	
Bed	Regular bed
Extrabed	Extra bed
FT	Fast Track Bed
MD	Physician
Nurse	Registered Nurse
RegistrationRep	Registration Representative
Triagenurse	Triage Nurse
Queues	
MD_Q	Queue for the Physician
NurseQ_1	Queue for the Initial Nurse visit
NurseQ_2	Queue for the Nurse visit after the physician.
NurseQ_3	Nurse Queue for the observation check up
RegRepQ	Queue for Registration Representative
Triage_Q	Queue of the patients waiting for the Triage Nurse
WaitQP2	Patients Category 2 waiting for bed in the waiting room
WaitQP3	Patients Category 3 waiting for bed in the waiting room

B.3 ER Experiment Frame Text Version

```
ATTRIBUTES:
1,doctortime,0:
2,nursetime,0:
3,pcategory,0:
4,arriveby,0:
5,whichbed,0:
6,observationtime,0:
7,decision,0:
8,tnowenteringloop,0:
9,TOAQ;

STATIONS:
1,registrationstation:
2,waitingroom:
3,TreatmentStation:
4,MyEntrance:
5,MyExit:
6,TriageStation;

SETS:
1,BedsP1,Bed,extrabed:
2,BedsP2,Bed,extrabed:
3,BedsP3,FT,Bed,extrabed:
4,BedsP4,FT,Bed;
```

```
NICKNAMES:
observe,1:
emergent,1:
ambulance,1:
urgent,2:
owncar,2:
discharge,3:
stable,4:
nonurgent,3:
admit,2;

TALLIES:
1,TWaitQP2:
2,TWaitQP3:
3,TWaitQP4:
4,Time ER 1:
5,Time ER 2:
6,Time ER 3:
7,Time ER 4:
8,TNurseQ_1:
9,TNurseQ_2:
10,TNurseQ_3:
11,TMD_Q;
```

```
RESOURCES:
1,Nurse,Capacity(6),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
2,MD,Capacity(1),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
3,bed,Capacity(10),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
4,extrabed,Capacity(1),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
5,RegistrationRep,Capacity(2),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
6,FT,Capacity(1),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
7,triagenurse,Capacity(2),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,);

QUEUES:
1,WaitQP2,FirstInFirstOut,,AUTOSTATS(Yes,,):
2,WaitQP3,FirstInFirstOut,,AUTOSTATS(Yes,,):
3,WaitQP4,FirstInFirstOut,,AUTOSTATS(Yes,,):
4,nurseQ_1,LowValueFirst(pcategory),,AUTOSTATS(Yes,,):
5,NurseQ_2,LowValueFirst(pcategory),,AUTOSTATS(Yes,,):
6,NurseQ_3,LowValueFirst(pcategory),,AUTOSTATS(Yes,,):
7,triage_Q,LowValueFirst(pcategory),,AUTOSTATS(Yes,,):
8,MD_Q,LowValueFirst(pcategory),,AUTOSTATS(Yes,,);

DSTATS:
1,nq(waitQP4),Number of patients 4 in waiting room:
2,NR(1),Nurse Busy:
3,NR(2),MD Busy:
4,NR(3),Bed Busy:
5,MR(MD),MD Available:
6,MR(Nurse),Nurse Available:
7,mr(bed),Beds Available:
8,NQ(waitQP3),Number of patients 3 in waiting room:
9,NQ(waitQP2),Number of patients 2 in waiting room:
10,nq(MD_Q),Number of patients in doctor q:
11,nq(nurseQ_1),Number in nurse Q_1:
12,nq(nurseQ_2),Number in nurse Q_2:
13,nq(nurseQ_3),Number in nurse Q_3:
14,nr(1)/mr(1),util of nurse:
15,nr(2)/mr(2),md util:
16,nr(3)/mr(3),bed util;
```

```
COUNTERS:
1,arrivebyambulance,,Replicate:
2,arrivebyowncar,,Replicate:
3,Patients No Beds,,Replicate:
4,Patients Leaving ER,,Replicate:
5,People entering Waiting Room,,Replicate:
6,p2_goingforbed,,Replicate:
7,p3_goingforbed,,Replicate:
8,p4_goingforbed,,Replicate;

EXPRESSIONS:
1,doctor1,uniform(0.17,0.33):
2,doctor2,uniform(0.167,0.25):
3,doctor3,uniform(0.0167,0.083):
4,doctor4,uniform(0.0167,0.083):
5,nurse1,uniform(0.5,1):
6,nurse2,uniform(0.5,0.67):
7,nurse3,uniform(0.33,0.67):
8,nurse4,uniform(0.05,0.5):
9,PatientTypeAmbulance,discrete(.435,1,.82,2,1,3):
10,PatientTypeOwnCar,Discrete(.026,1,.471,2,.967,3,1,4):
11,Observation1,max(0,(erla(1.02,4)-doctortime-nursetime)):
12,Observation2,Max(0,(max(norm(3.84,2.05),1.79)-doctortime-nursetime)):
13,Observation3,Max(0,(gamm(1.4,1.5)-nursetime-doctortime)):
14,Observation4,MAx(0,(max(norm(0.552,0.295),.257)-doctortime-nursetime)):
15,admitted,uniform(0.35,0.5)+uniform(1/60,4/60):
16,discharged,uniform(0.35,0.5);

VARIABLES:
1,MaxNur,CLEAR(System),CATEGORY("None-None"),9:
2,MaxDoc,CLEAR(System),CATEGORY("None-None"),4:
3,MaxBeds,CLEAR(System),CATEGORY("None-None"),40:
4,Combinaciones,CLEAR(System),CATEGORY("None-None"),120:
5,RecordW,CLEAR(System),CATEGORY("None-None"),1:
6,MinNur,CLEAR(System),CATEGORY("None-None"),6:
7,MinDoc,CLEAR(System),CATEGORY("None-None"),1:
8,MinBeds,CLEAR(System),CATEGORY("None-None"),10;

REPLICATE,
1500,0.0,5300,Yes,Yes,1000,,,24.0,Hours,No,No;
```

B.4 ER Model Frame Text Version

```
83$           CREATE,        1,0:expo(1/6):NEXT(84$);
84$           ASSIGN:        arriveby=discrete(0.0818,1,1.0,2);
86$           COUNT:         ArriveBy,1;
85$           ASSIGN:        pcategory=ed(ArriveBy + 8);
87$           ROUTE:         0.0,MyEntrance;

5$            STATION,       MyEntrance:MARK(Timein);
3$            BRANCH,        1,10:
                             If,ArriveBy==1,0$,Yes:
                             Else,2$,Yes;
0$            BRANCH,        1,10:
                             If,(nr(3)<mr(3)).or.(nr(4)<mr(4)),79$,Yes:
                             If,(nr(6)<mr(6)),82$,Yes:
                             Else,6$,Yes;
79$           BRANCH,        1,10:
                             If,pcategory==1,7$,Yes:
                             If,pcategory==2,80$,Yes:
                             Else,81$,Yes;
7$            SEIZE,         1,Other:
```

```
                                  Select(BedsP1,POR,WhichBed),1:NEXT(1$);
1$              ROUTE:            0.033,TreatmentStation;
80$             SEIZE,            1,Other:
                                  Select(BedsP2,POR,WhichBed),1:NEXT(1$);
81$             SEIZE,            1,Other:
                                  Select(BedsP3,POR,WhichBed),1:NEXT(1$);
82$             BRANCH,           1,10:
                                  If,pcategory==1,6$,Yes:
                                  If,pcategory==2,6$,Yes:
                                  Else,81$,Yes;
6$              COUNT:            Patients No Beds,1;
4$              DISPOSE:          No;
2$              ROUTE:            0.033,TriageStation;

12$             STATION,          TriageStation;
9$              QUEUE,            triage_Q;
10$             SEIZE,            1,Other:
                                  triagenurse,1:NEXT(13$);
13$             DELAY:            Uniform(0.0833,0.1667),,Other:NEXT(11$);
11$             RELEASE:          triagenurse,1;
14$             BRANCH,           1,10:
                                  If,pcategory==1,15$,Yes:
                                  Else,8$,Yes;
15$             BRANCH,           1,10:
                                  If,(nr(3)<mr(3)).or.(nr(4)<mr(4)),19$,Yes:
                                  Else,18$,Yes;
19$             SEIZE,            1,Other:
                                  Select(BedsP1,POR,WhichBed),1:NEXT(16$);
16$             ROUTE:            0.033,TreatmentStation;
18$             COUNT:            Patients No Beds,1;
17$             DISPOSE:          No;
8$              ROUTE:            0.033,registrationstation;

21$             STATION,          registrationstation;
22$             SEIZE,            1,Other:
                                  RegistrationRep,1:NEXT(23$);
23$             DELAY:            Expo(0.1667),,Other:NEXT(24$);
24$             RELEASE:          RegistrationRep,1;
20$             ROUTE:            0.033,waitingroom;
25$             STATION,          waitingroom;
26$             COUNT:            People entering Waiting Room,1;
34$             BRANCH,           1,10:
                                  If,pcategory==4,28$,Yes:
                                  If,pcategory==3,30$,Yes:
                                  Else,27$,Yes;
28$             QUEUE,            WaitQP4:MARK(TOAQ);
38$             SEIZE,            1,Other:
                                  Select(BedsP4,POR,WhichBed),1:NEXT(35$);
35$             TALLY:            TWaitQP4,interval(TOAQ),1;
31$             COUNT:            p4_goingforbed,1;
29$             ROUTE:            0.033,TreatmentStation;
30$             QUEUE,            WaitQP3:MARK(TOAQ);
39$             SEIZE,            1,Other:
                                  Select(BedsP3,POR,WhichBed),1:NEXT(36$);
36$             TALLY:            TWaitQP3,interval(TimeWR),1;
32$             COUNT:            p3_goingforbed,1:NEXT(29$);
27$             QUEUE,            WaitQP2:MARK(TOAQ);
40$             SEIZE,            1,Other:
                                  Select(BedsP2,POR,WhichBed),1:NEXT(37$);
37$             TALLY:            TWaitQP2,interval(TOAQ),1;

33$             COUNT:            p2_goingforbed,1:NEXT(29$);
46$             STATION,          TreatmentStation;
47$             ASSIGN:           nursetime=ed(pcategory+4):
                                  doctortime=ed(pcategory);
49$             QUEUE,            nurseQ_1:MARK(TOAQ);
48$             SEIZE,            1,Other:
                                  Nurse,1:NEXT(61$);
61$             TALLY:            TNurseQ_1,interval(TOAQ),1;
```

```
62$            DELAY:         nursetime*1.5,,Other:NEXT(50$);
50$            RELEASE:       Nurse,1;
52$            QUEUE,         MD_Q:MARK(TOAQ);
51$            SEIZE,         1,Other:
                              MD,1:NEXT(67$);
67$            TALLY:         TMD_Q,interval(TOAQ),1;
63$            DELAY:         doctortime,,Other:NEXT(53$);
53$            RELEASE:       MD,1;
55$            QUEUE,         NurseQ_2:MARK(TOAQ);
54$            SEIZE,         1,Other:
                              Nurse,1:NEXT(68$);
68$            TALLY:         TNurseQ_2,interval(TOAQ),1;
64$            DELAY:         nursetime*0.25,,Other:NEXT(56$);
56$            RELEASE:       Nurse,1;
44$            ASSIGN:        decision=discrete(0.5,1,.75,2,1,3);
42$            BRANCH,        1,10:
                              If,decision==1,45$,Yes:
                              Else,43$,Yes;
45$            ASSIGN:        observationtime=ed(pcategory+10):MARK(tnowenteringloop);
66$            DELAY:         uniform(0.35,0.5),,Other:NEXT(58$);                 Time
between nurse checking
58$            QUEUE,         NurseQ_3:MARK(TOAQ);
57$            SEIZE,         1,Other:
                              Nurse,1:NEXT(69$);
69$            TALLY:         TNurseQ_3,interval(TOAQ),1;
65$            DELAY:         uniform(1/60,4/60),,Other:NEXT(59$);
59$            RELEASE:       Nurse,1;
41$            BRANCH,        1,10:
                              If,tnow>=(observationtime+tnowenteringloop),60$,Yes:
                              Else,66$,Yes;
60$            ROUTE:         0.0,MyExit;
43$            DELAY:         ed(decision+13),,Other:NEXT(60$);

73$            STATION,       MyExit;
75$            BRANCH,        1,10:
                              If,pcategory==4,70$,Yes:
                              If,pcategory==3,76$,Yes:
                              If,pcategory==2,77$,Yes:
                              Else,78$,Yes;
70$            RELEASE:       BedsP4(whichbed),1;
74$            TALLY:         pcategory+3,interval(timein),1;
72$            COUNT:         Patients Leaving ER,1;
71$            DISPOSE:       No;
76$            RELEASE:       BedsP3(whichbed),1:NEXT(74$);
77$            RELEASE:       BedsP2(whichbed),1:NEXT(74$);
78$            RELEASE:       BedsP1(whichbed),1:NEXT(74$);
```

APPENDIX C

C.1 Foundry Test Bed

The Foundry model used by Jones (1999) produces castings made of gray iron including, covers, rings, grates, hoods, frames, and others. The first stage is the molding process. Then, the castings are taken to a cooling yard for a rest period of 24 hours before been taken to the finishing department. In the finishing department all the casting are hanged from a conveyor, which takes them to the sand blast system for cleaning. Following, the castings are routed to different machines according to their types. The routes are as follows:

- Rings: The rings go to the lathe machines where they are machined in order to obtain a perfect fit with the covers.
- Covers: The covers first go to the grinding stations where they are polished. Then, they are transported to the lathe machines where they are machined in order to obtain a perfect fit with the rings.
- Frames, grates, hoods, and boxes: They are transported to the grinding stations to remove the excess of material.

All the castings on the finishing department must be transported to the final storage to wait for shipping. The transportation is done by power conveyors, gravity conveyors, and forklifts, and it can be done in batches or individually depending on the casting. A flow chart of this test bed is given in Appendix C.

Specific numeric data for this model can be found in the work of Jones (1999).

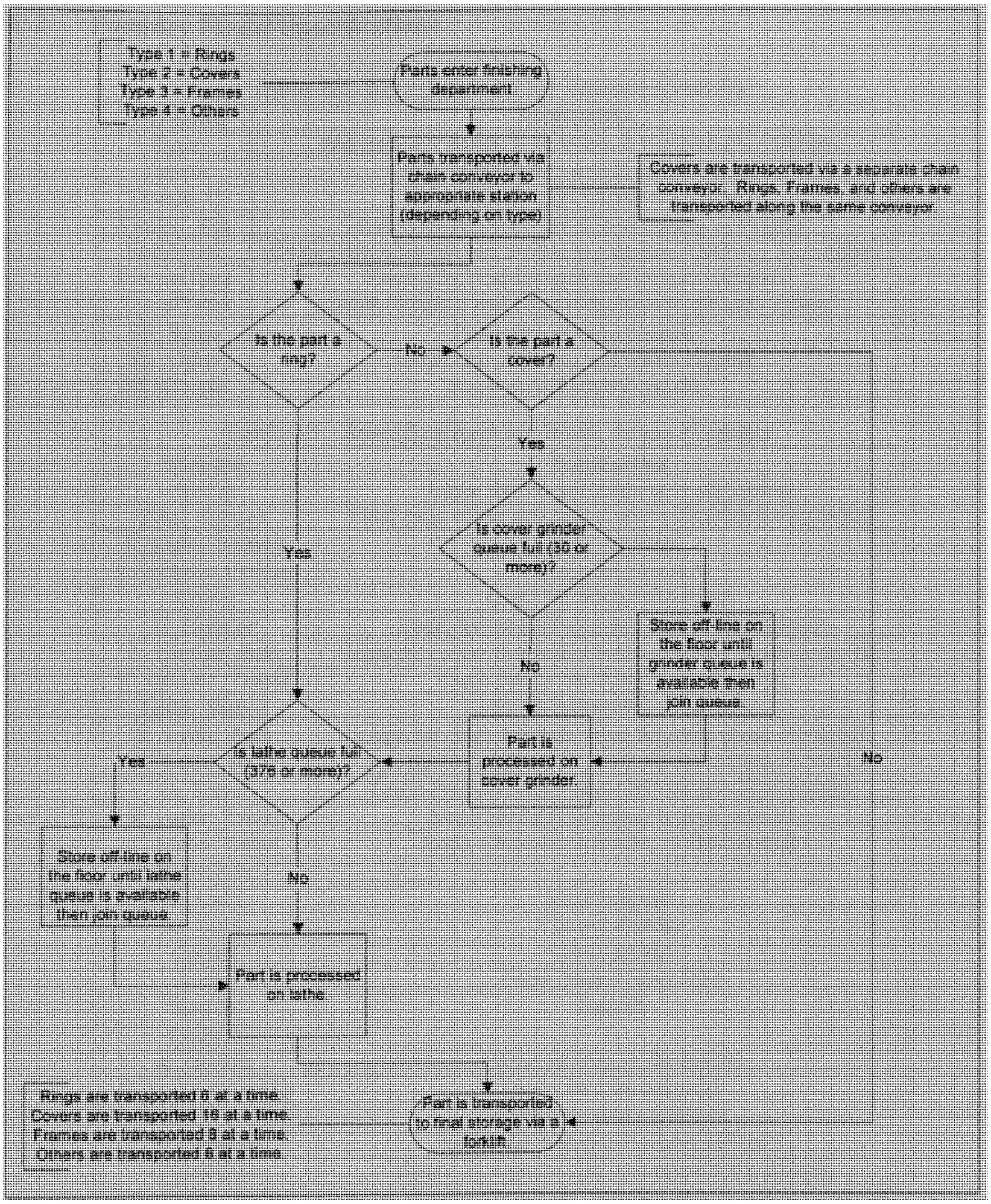

Type 1 = Rings
Type 2 = Covers
Type 3 = Frames
Type 4 = Others
Parts enter finishing department
Parts transported via chain conveyor to appropriate station (depending on type)
Covers are transported via a separate chain conveyor. Rings, Frames, and others are transported along the same conveyor.
Is the part a ring?
No
Is the part a cover?
Yes
Is cover grinder queue full (30 or more)?
Yes
Store off-line on the floor until grinder queue is available then join queue.
No
No
Part is processed on cover grinder.
Is lathe queue full (376 or more)?
Yes
Store off-line on the floor until lathe queue is available then join queue.
No
Part is processed on lathe.
Rings are transported 6 at a time.
Covers are transported 16 at a time.
Frames are transported 8 at a time.
Others are transported 8 at a time.
Part is transported to final storage via a forklift.

Figure 21: Foundry Flowchart (Lee 2002)

C.2 Foundry Model Specifications

Table 12: Specifications for Entities-Foundry

Attributes	Explanation
Type	Part type. 1 = Rings, 2 = Covers, 3 = Frames, 4 = Others
TimeIn	Clock time the part arrives to the system.
TimeIn_Q	Clock time the part arrives to a queue.
TimeIn_Res	Clock time the part is capable of seizing a resource.
EndTravel	Clock time when the part arrives at a station. TNOW + TravelTime
ProcessTime	Processing time for each part at each resource.
TravelTime	Length of time a part takes to traverse to the next station.
BS_Rings	Batch size of rings created.
BS_Other	Batch size of frames and others created.
BS_Covers	Batch size of covers created.

Table 13: Specifications for Work Areas-Foundry

Stations	Explanation
Arrive	Logic for the real entrance to the system
Final_Storage	Final stage of the simulation where statistics are collected
R_Lathe_Unload	Logic for rings
C_Grinder_Unload	Logic for covers
Others	Logic for frames and others
C_Grinder	Logic for covers being processed at the grinder
C_Grinder_Balk	Logic for covers that balk when the grinder queue is too full
C_Grinder_Travel	Logic for covers traveling to the lathe
Lathe_Station	Logic for processing at the lathe
R_Lathe_Balk	Logic for rings that balk when the lathe queue is too full
C_Lathe_Balk	Logic for covers that balk when the lathe queue is too full
O_Travel	Logic for frames and others traveling to final storage
R_Lathe_Travel	Logic for rings traveling to final storage
C_Lathe_Travel	Logic for covers traveling to final storage
Resources	
R_Unld	Ring unloader
C_Unld	Cover unloader
Cgrind	Cover grinder
Lathe	Cover lathe + Ring lathe
Forklift	Forklift
Queues	
A_Travelq	Queue arriving to the system
R_Lathe_Unloadq	Queue for ring unloader
C_Grinder_Unloadq	Queue for cover unloader
O_Unloadq	Queue for frames and others unloader
C_Grinderq	Queue for covers being processed at the grinder
C_Grinder_Balkq	Queue for covers that balk when the grinder queue is too full
C_Grinder_Travelq	Queue for covers traveling to the lathe
Latheq	Queue for lathe station
R_Lathe_Balkq	Queue for rings that balk when the lathe queue is too full
C_Lathe_Balkq	Queue for covers that balk when the lathe queue is too full
O_Travelq	Queue for frames and others traveling to final storage
R_Lathe_Travelq	Queue for rings traveling to final storage
C_Lathe_Travelq	Queue for covers traveling to final storage
Forkliftq	Queue for units waiting for a forklift

C.3 Foundry Experiment Frame Text Version

```
ATTRIBUTES:
1,Type,:
2,TimeIn:
3,TimeIn_Q:
4,TimeIn_Res:
5,EndTravel:
6,ProcessTime:
7,TravelTime;

STATIONS:
1,Arrive:
2,Others:
3,O_Travel:
4,R_Lathe_Unload:
5,Lathe_Station:
6,R_Lathe_Balk:
7,R_Lathe_Travel:
8,C_Grinder_Unload:
9,C_Grinder:
10,C_Grinder_Balk:
11,C_Grinder_Travel:
12,C_Lathe_Balk:
13,C_Lathe_Travel:
14,Final_Storage;
```

```
QUEUES:
1,A_Travelq,FirstInFirstOut,,AUTOSTATS(Yes,,):
2,O_Unloadq,FirstInFirstOut,,AUTOSTATS(Yes,,):
3,O_Travelq,FirstInFirstOut,,AUTOSTATS(Yes,,):
4,R_Lathe_Unloadq,FirstInFirstOut,,AUTOSTATS(Yes,,):
5,Latheq,LowValueFirst(Type),,AUTOSTATS(Yes,,):
6,R_Lathe_Balkq,FirstInFirstOut,,AUTOSTATS(Yes,,):
7,R_Lathe_Travelq,FirstInFirstOut,,AUTOSTATS(Yes,,):
8,C_Grinder_Unloadq,FirstInFirstOut,,AUTOSTATS(Yes,,):
9,C_Grinderq,FirstInFirstOut,,AUTOSTATS(Yes,,):
10,C_Grinder_Balkq,FirstInFirstOut,,AUTOSTATS(Yes,,):
11,C_Grinder_Travelq,FirstInFirstOut,,AUTOSTATS(Yes,,)
:
12,C_Lathe_Travelq,FirstInFirstOut,,AUTOSTATS(Yes,,):
13,C_Lathe_Balkq,FirstInFirstOut,,AUTOSTATS(Yes,,):
14,Forkliftq,FirstInFirstOut,SHARED,AUTOSTATS(Yes,,);

COUNTERS:
1,Num_of_Rings,,Replicate:
2,Num_of_Covers,,Replicate:
3,Num_of_Frames,,Replicate:
4,Num_of_Others,,Replicate:
5,Bottleneck_Rings,,Replicate:
6,Bottleneck_Covers_Lathe,,Replicate:
7,Bottleneck_Covers,,Replicate:
8,Periods,,Replicate;
```

```
RESOURCES:
1,CGrind,Capacity(3),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
2,Lathe,Capacity(12),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
3,Forklift,Capacity(7),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
4,R_Unld,Capacity(1),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,):
5,C_Unld,Capacity(1),,Stationary,COST(0.0,0.0,0.0),,AUTOSTATS(Yes,,);

SEQUENCES:
1,JobType1_Rings,Arrive,TravelTime=Normal(26.43,1.30)&R_Lathe_Unload,ProcessTime=0.03+Erla
ng(0.0255,6)&
Lathe_Station,ProcessTime=Normal(3.5,0.412)&R_Lathe_Travel&Final_Storage:
2,JobType2_Covers,Arrive,TravelTime=Normal(30.2,1.30)&C_Grinder_Unload,ProcessTime=0.03+
Erlang(0.0255,6)&
C_Grinder,ProcessTime=Normal(0.852,0.272)&C_Grinder_Travel&Lathe_Station,ProcessTime=1.15+
1.85*Beta(1.58,1.54)&
C_Lathe_Travel&Final_Storage:
3,JobType3N4_Frames_N_Others,Arrive,TravelTime=Normal(22.13,1.30)&Others,ProcessTime=0.03+
Erlang(0.0255,6)&
O_Travel&Final_Storage;Final_Storage:
3,JobType3N4_Frames_N_Others,Arrive,TravelTime=MX(0,Normal(0.33,1.30))+21.8&Others,
ProcessTime=0.03+Erlang(0.0255,6)&O_Travel&Final_Storage;

DSTATS:
1,NQ(Latheq):
2,NR(CGrind)/MR(CGrind),Util of CoverGrind:
3,NR(Lathe)/MR(Lathe),Util of Lathe:
4,NR(Forklift)/MR(Forklift),Util of Forklift:
5,NR(R_Unld)/MR(R_Unld),Util of RingUnloader:
6,NR(C_Unld)/MR(C_Unld),Util of CoverUnloader;

VARIABLES:
1,BS_Rings,CLEAR(System),CATEGORY("None-None"):
2,BS_Other,CLEAR(System),CATEGORY("None-None"):
3,BS_Covers,CLEAR(System),CATEGORY("None-None"):
4,MinRes1,CLEAR(System),CATEGORY("None-None"),4:
5,MinRes2,CLEAR(System),CATEGORY("None-None"),1:
6,MinRes3,CLEAR(System),CATEGORY("None-None"),3:
7,MinRes4,CLEAR(System),CATEGORY("None-None"),2:
```

```
8,MaxRes1,CLEAR(System),CATEGORY("None-None"),17:
9,MaxRes2,CLEAR(System),CATEGORY("None-None"),2:
10,MaxRes3,CLEAR(System),CATEGORY("None-None"),8:
11,MaxRes4,CLEAR(System),CATEGORY("None-None"),17:
12,Combination,CLEAR(System),CATEGORY("None-None"),360:
13,w,CLEAR(System),CATEGORY("None-None"),3;

SEEDS:
1,18765,No;

REPLICATE,
1,0.0,30000,Yes,Yes,1500,,,24.0,Hours,No,No;
```

C.4 Foundry Model Frame Text Version

```
0$            CREATE,       1,0.0001::NEXT(1$);
1$            ASSIGN:       BS_Rings=2:
                            BS_Covers=2:
                            BS_Other=0;
2$            SCAN:         NQ(Latheq).gt.225;
3$            DELAY:        Normal(5,.45),,Other:NEXT(4$);
4$            COUNT:        Periods,1;
5$            ASSIGN:       BS_Covers=0:
                            BS_Rings=0:
                            BS_Other=4;
6$            SCAN:         NQ(Latheq).lt.60:NEXT(1$);
7$            CREATE,       BS_Rings,0.001:Uniform(0.64,0.66,1):MARK(TimeIn):NEXT(8$);
8$            ASSIGN:       Type=1:
                            NS=1;
9$            ROUTE:        0.0,seq;
10$           CREATE,       BS_Covers,0.001:Uniform(0.64,0.66,1):MARK(TimeIn):NEXT(11$);
11$           ASSIGN:       Type=2:
                            NS=2;
12$           ROUTE:        0.0,seq;
13$           CREATE,       BS_Other,0.001:Uniform(0.64,0.66,1):MARK(TimeIn):NEXT(14$);
14$           ASSIGN:       Type=Discrete(0.5,3,1.0,4):
                            NS=3;
15$           ROUTE:        0.0,seq;

16$           STATION,      Arrive:MARK(TimeIn_Q);
17$           ASSIGN:       EndTravel=TNOW+TravelTime;
QArrive       QUEUE,        A_Travelq;
18$           SCAN:         TNOW.ge.EndTravel;
19$           TALLY:        Difference_arrivaltravel,TNOW-EndTravel,1;
20$           ASSIGN:       EndTravel=0;
21$           ROUTE:        0.0,SEQ;

22$           STATION,      R_Lathe_Unload:MARK(TimeIn_Q);
Q_R_Unload    QUEUE,        R_Lathe_Unloadq;
23$           SEIZE,        1,Other:
                            R_Unld,1:MARK(TimeIn_Res):NEXT(24$);
24$           DELAY:        ProcessTime,,Other:NEXT(25$);
25$           RELEASE:      R_Unld,1;
26$           ROUTE:        0.0,SEQ;

27$           STATION,      Lathe_Station:MARK(TimeIn_Q);
Lathe         QUEUE,        Latheq,376,Lathe_Balk;
28$           SEIZE,        1,Other:
                            Lathe,1:MARK(TimeIn_Res):NEXT(32$);
32$           TALLY:        Type+4,Interval(TimeIn_Q),1;
29$           DELAY:        ProcessTime,,Other:NEXT(30$);
```

```
30$           RELEASE:       Lathe,1;
31$           ROUTE:         0.0,SEQ;
Lathe_Balk    BRANCH,        1,10:
                             If,Type==1,33$,Yes:
                             Else,34$,Yes;

33$           STATION,       R_Lathe_Balk:MARK(TimeIn_Q);
35$           QUEUE,         R_Lathe_Balkq;
36$           SCAN:          NQ(Latheq).lt.376;
37$           COUNT:         Type+4,1:NEXT(Lathe);

34$           STATION,       C_Lathe_Balk;
Floors_Covers_Lathe QUEUE,   C_Lathe_Balkq;

38$           STATION,       C_Grinder_Unload:MARK(TimeIn_Q);
Q_C_G_Unload  QUEUE,         C_Grinder_Unloadq;
39$           SEIZE,         1,Other:
                             C_Unld,1:NEXT(40$);
40$           DELAY:         ProcessTime,,Other:NEXT(41$);
41$           RELEASE:       C_Unld,1;
42$           ROUTE:         0.0,SEQ;

GStat         STATION,       C_Grinder:MARK(TimeIn_Q);
Grind         QUEUE,         C_Grinderq,30,Floor_Covers;
43$           SEIZE,         1,Other:
                             CGrind,1:MARK(TimeIn_Res):NEXT(47$);
47$           TALLY:         TqueCGrinder_q,Interval(TimeIn_Q),1;
44$           DELAY:         ProcessTime,,Other:NEXT(45$);
45$           RELEASE:       CGrind,1;
46$           ROUTE:         0.0,SEQ;
Floor_Covers  QUEUE,         C_Grinder_Balkq;
49$           SCAN:          NQ(C_Grinderq).lt.30;
50$           COUNT:         Bottleneck_Covers,1:NEXT(Grind);

48$           STATION,       C_Grinder_Balk:MARK(TimeIn_Q):NEXT(Floor_Covers);

51$           STATION,       C_Grinder_Travel:MARK(TimeIn_Q);
53$           ASSIGN:
EndTravel=TNOW+TRIA(5.266,8.121,91.449)+TRIA(0.417,0.828,2.691)+0.33;
Q_C_G_Travel  QUEUE,         C_Grinder_Travelq;
52$           SCAN:          TNOW.ge.EndTravel;
54$           TALLY:         Difference_covergrindtravel,TNOW-EndTravel,1;
55$           ROUTE:         0.0,SEQ;

56$           STATION,       Others:MARK(TimeIn_Q);
Q_O_Unload    QUEUE,         O_Unloadq;
57$           SEIZE,         1,Other:
                             R_Unld,1:MARK(TimeIn_Res):NEXT(58$);
58$           DELAY:         ProcessTime,,Other:NEXT(59$);
59$           RELEASE:       R_Unld,1;
60$           ROUTE:         0.0,SEQ;

61$           STATION,       C_Lathe_Travel;
65$           GROUP,         ,Temporary:16,Last:NEXT(66$);
66$           ASSIGN:
EndTravel=TNOW+TRIA(6.896,10.110,14.541)+TRIA(0.167,0.707,2.06)+1.34;
62$           QUEUE,         Forkliftq;
63$           SEIZE,         1,Other:
                             Forklift,1:NEXT(Q_C_L_Travel);
Q_C_L_Travel  QUEUE,         C_Lathe_Travelq;
67$           SCAN:          TNOW.ge.EndTravel;
68$           TALLY:         Difference_coverlathetravel,TNOW-EndTravel,1;
64$           ROUTE:         0.0,seq;

69$           STATION,       O_Travel:MARK(TimeIn_Q);
73$           GROUP,         Type,Temporary:8,Last:NEXT(74$);
74$           ASSIGN:
EndTravel=TNOW+1.75+TRIA(0.930,2.049,115.34)+TRIA(0.167,23.023,46.279);
70$           QUEUE,         Forkliftq;
```

```
71$            SEIZE,         1,Other:
                              Forklift,1:NEXT(Q_O_Travel);
Q_O_Travel     QUEUE,         O_Travelq;
75$            SCAN:          TNOW.ge.EndTravel;
76$            TALLY:         Difference_othertravel,TNOW-EndTravel,1;
72$            ROUTE:         0.0,seq;

77$            STATION,       R_Lathe_Travel;
81$            GROUP,         ,Temporary:6,Last:NEXT(82$);
82$            ASSIGN:
EndTravel=TNOW+TRIA(1.749,3.099,5.960)+TRIA(0.083,1.077,3.116)+1.58;
78$            QUEUE,         Forkliftq;
79$            SEIZE,         1,Other:
                              Forklift,1:NEXT(Q_R_Travel);
Q_R_Travel     QUEUE,         R_Lathe_Travelq;
83$            SCAN:          TNOW.ge.EndTravel;
84$            TALLY:         Difference_ringtravel,TNOW-EndTravel,1;
80$            ROUTE:         0.0,seq;

85$            STATION,       Final_Storage;
86$            RELEASE:       Forklift,1;
87$            SPLIT::NEXT(88$);
88$            TALLY:         Type,TNOW-TimeIn,1;
89$            COUNT:         Type,1;
90$            DISPOSE:       No;
```

APPENDIX D

D.1 Definition of Global Variables

```
Option Explicit
'*******************************************************************************************
'Variables used to call and fire an already trained ANN
'inarray#(), outarray#(): Define the size of the input and output arrays
'inputs%, output%: Take their value from the network that has been called
'netnumber%: Variable to assign a number to the network
'netisopen%: Variable to determine if a network has been opened or not
'defpath%: Variable to define the path to the correspondent NN network

    Public inarray#(), outarray#()
    Public inputs%, output%
    Public netnumber%, netisopen%
    Public defpath%

'*******************************************************************************************
'The following functions are used to open, fire and close each one of the networks
'Declaring functions NS2-32.DLL

    Declare Function OpenNet% Lib "Ns2-32.dll" (ByVal defpath$, netnumber%, inputs%,
output%)
    Declare Function FireNet% Lib "Ns2-32.dll" (netnumber%, inputsarray#, outputsarray#)
    Declare Function CloseNet% Lib "Ns2-32.dll" (netnumber%)

'*******************************************************************************************
'Variables used to manage the training of the ANNs
'TotalComb: Stores number of possible combinations of resources' capacities
'w: Used to detect when to write the simulation outputs and change the level of services
'NeuralNetwork: Keeps track of the network selected by the user
'NNnumber: Takes the value of the network that must be fired by the user's inputs
'decision: Takes its value according with the interaction that a user has with the NN_GDS
'user: Defines which type of user is interacting with the prototype
'flag: Flag to indicate that the training file is ready
'changetype: Takes a value according with the change that the non-expert user selects
'experttask: Takes a value according with the interaction that the expert user selects
'unnecesary: Flag to indicate that the user has cancelled the process of make changes
'warning: Flag to warn the user when the number of training patterns is too low

    Public TotalComb, w, changetype As Integer
    Public NeuralNetwork As String
    Public decision, user, flag, NNnumber As Integer
    Public experttask, ranges, warning As Integer
    Public unnecessary As Integer
    Public Combo() As Variant

'Constants used to manage the changes requested by the user and to identify the of user

    Public Const ChangeRes As Integer = 1
    Public Const ChangeST As Integer = 2
    Public Const ChangeRT As Integer = 3
    Public Const Expert As Integer = 1
    Public Const Nonexpert As Integer = 2

'*******************************************************************************************
'Variables used to make changes in the simulation model
'i,j,k,l,o,p: Used for tagged elements
'MinRes1, MaxRes1, MinRes2, MaxRes2, MinRes3, MaxRes3:
'Store lower/upper bounds of the ranges for the resources (1:Nurses, 2:Doctors, 3:Beds)
'TheModel: Definition of the Arena model name
'ResourceModule: Definition of the element "resources" as an Arena Module
```

```
'VariableModule: Definition of the element "variables" as an Arena Module
'ExpressionsModule: Definition of the element "expressions" as an Arena Module
'Delay1Module, Delay2Module, Delay3Module: Definition of block "delay" as Arena Module
'Rout1Module, Rout2Module, Rout3Module, Rout4Module, Rout5Module, Rout6Module:
'Definition of the block "route" as an Arena Module
'ReplicaModule: Definition of the element "replicate" as an Arena Module
'OUT: Definition of the Siman model name

    Public i, j, k, l, o, p As Integer
    Public MinRes1, MaxRes1, MinRes2, MaxRes2, MinRes3, MaxRes3 As Integer
    Public TheModel As Arena.Model
    Public ResourceModule As Arena.Module
    Public VariableModule As Arena.Module
    Public ExpressionsModule As Arena.Module
    Public Delay1Module As Arena.Module
    Public Delay2Module As Arena.Module
    Public Delay3Module As Arena.Module
    Public Rout1Module As Arena.Module
    Public Rout2Module As Arena.Module
    Public Rout3Module As Arena.Module
    Public Rout4Module As Arena.Module
    Public Rout5Module As Arena.Module
    Public Rout6Module As Arena.Module
    Public ReplicaModule As Arena.Module
    Public OUT As SIMAN
```

D.2 VBA UIR Subroutines

D.2.1 Subroutine GDS_main

```
Sub GDS_main()

'*****************************************************************************************
'Definition of local variables
'MoreInteraction Variable that determines when the loop stops

    Dim StopProcess As Boolean
    Dim MoreInteraction As Integer
    Dim Hora

    flag = 0: MoreInteraction = 1
    unnecessary = 0: decision = 0: experttask = 0

    Do While MoreInteraction = 1
        Load UserType
        UserType.Show

        StopProcess = UserType.CancelButton.Cancel

        Unload UserType

'When the user cancels the program, the replication lenght changes to one (1)

        If StopProcess = True Then
            MoreInteraction = CancelRun
        Else
            If user = Nonexpert Then
                Load UserMenu
                UserMenu.Show

                StopProcess = UserMenu.CancelButton.Cancel

                Unload UserMenu

                If StopProcess = True Then MoreInteraction = CancelRun

'Make Changes
                If decision = 1 Then
                    Call ChangesMain(unnecessary)
                    If unnecessary = 0 Then
                        Call SetScenarios
                        MoreInteraction = 0
                    End If
                    If unnecessary = 1 Then MoreInteraction = CancelRun
                End If

'Use Current Neural Network
                If decision = 2 Then
                    Call NN2Querying(unnecessary)
                    Unload ENTRY
                End If
            Else
                Load Expertform
                Expertform.Show

                StopProcess = Expertform.CancelButton.Cancel

                Unload Expertform
                If StopProcess = True Then MoreInteraction = CancelRun

'Use Current Neural Network
                If experttask = 1 Then
                    Call NN2Querying(unnecessary)
                    Unload ENTRY
                End If

'Training new ANNs
                If experttask = 2 Then
                    Call SetScenarios
                    MoreInteraction = 0
                End If
            End If
        End If
    Loop
End Sub
```

D.2.2 Subroutine CancelRun

```
Public Function CancelRun() As Integer
    Set TheModel = ThisDocument.Model
    i = TheModel.Modules.Find(smFindTag, "Replicas")
    Set ReplicaModule = TheModel.Modules(i)
    ReplicaModule.Data("NumReps") = 1
    ReplicaModule.Data("Length") = 1
    ReplicaModule.UpdateShapes
    CancelRun = 0

End Function
```

D.3 VBA ACR Subroutines

D.3.1 Subroutine ChangesMain

```
Sub ChangesMain(unnecessary As Integer)

'Connecting to the model itself.  The replication time is returned to its normal value.
'When the user cancels the program, the replication lenght changes to one (1).

    Set TheModel = ThisDocument.Model
    Set OUT = TheModel.SIMAN

    i = TheModel.Modules.Find(smFindTag, "Replicas")
    Set ReplicaModule = TheModel.Modules(i)
    ReplicaModule.Data("Length") = 5300
    ReplicaModule.UpdateShapes

'Declaring local variables
    Dim MoreChanges As Integer
    Dim StopProcess As Boolean
    Dim STNP1, STNP2, STNP3, STNP4, STDP1, STDP2, STDP3, STDP4 As String
    Dim STRR, STAT, STDT, STOT1, STOT2 As String
    Dim RTETT, RTETTR, RTTTT, RTTTR, RTRTW, RTWTT As String

    MoreChanges = 1

Do While MoreChanges = 1
    changetype = -1

    Load Changesform
    Changesform.Show
    StopProcess = Changesform.CancelButton.Cancel

    Unload Changesform

    If changetype = 0 Then
        MoreChanges = 0: unnecessary = 0
    End If

    If StopProcess = True Then
        MoreChanges = 0: unnecessary = 1
    End If

    If changetype = ChangeRes Then
'To show current capacities on the ResCapacity form as reference for the user.

        i = TheModel.Modules.Find(smFindTag, "recursos")
        j = TheModel.Modules.Find(smFindTag, "variables")
        Set ResourceModule = TheModel.Modules(i)
        Set VariableModule = TheModel.Modules(j)

        Load ResCapacity

        MinRes1 = VariableModule.Data("value(1,6)")
        MaxRes1 = VariableModule.Data("value(1,1)")
        MinRes2 = VariableModule.Data("value(1,7)")
        MaxRes2 = VariableModule.Data("value(1,2)")
        MinRes3 = VariableModule.Data("value(1,8)")
        MaxRes3 = VariableModule.Data("value(1,3)")
        ResCapacity.MinCurCNurses.Caption = MinRes1 + 1
        ResCapacity.MaxCurCNurses.Caption = MaxRes1 - 1
        ResCapacity.MinCurCDoctors.Caption = MinRes2 + 1
        ResCapacity.MaxCurCDoctors.Caption = MaxRes2 - 1
        ResCapacity.MinCurCBeds.Caption = MinRes3 + 1
        ResCapacity.MaxCurCBeds.Caption = MaxRes3 - 1
        StopProcess = ResCapacity.CancelButton.Cancel
        ranges = 0

        ResCapacity.Show

        Call CheckingRanges

        If StopProcess = False And ranges = 0 Then

'The values collected from ResCapacity form should be transferred to the variables
'and then to the Arena Model
'To determine infeasibility the ranges given by the user are increase by one unit up
'and one unit down.

            MinRes1 = Val(ResCapacity.MinNewCNurses.Text) - 1
            MaxRes1 = Val(ResCapacity.MaxNewCNurses.Text) + 1
            MinRes2 = Val(ResCapacity.MinNewCDoctors.Text) - 1
            MaxRes2 = Val(ResCapacity.MaxNewCDoctors.Text) + 1
            MinRes3 = Val(ResCapacity.MinNewCBeds.Text) - 1
            MaxRes3 = Val(ResCapacity.MaxNewCBeds.Text) + 1

            If MinRes1 <= 0 Then MinRes1 = 1
            If MinRes2 <= 0 Then MinRes2 = 1
            If MinRes3 <= 0 Then MinRes3 = 1

'Inserting lower bound of resources capacities in RESOURCES element

            ResourceModule.Data("CapExp(1)") = MinRes1
            ResourceModule.Data("CapExp(2)") = MinRes2
            ResourceModule.Data("CapExp(3)") = MinRes3
            ResourceModule.UpdateShapes

'Inserting upper bound of resources capacities in VARIABLES element for future use
(events)

            VariableModule.Data("value(1,1)") = MaxRes1
            VariableModule.Data("value(1,2)") = MaxRes2
            VariableModule.Data("value(1,3)") = MaxRes3
            VariableModule.Data("value(1,6)") = MinRes1
            VariableModule.Data("value(1,7)") = MinRes2
            VariableModule.Data("value(1,8)") = MinRes3
            VariableModule.UpdateShapes

        End If

        Unload ResCapacity

    End If

    If changetype = ChangeST Then
'To show currente service times into ResServiceT form as reference for the user.

        i = TheModel.Modules.Find(smFindTag, "expresiones")
        j = TheModel.Modules.Find(smFindTag, "RRDelay")
        k = TheModel.Modules.Find(smFindTag, "OBSDELAY1")
        l = TheModel.Modules.Find(smFindTag, "OBSDELAY2")

        Set ExpressionsModule = TheModel.Modules(i)
        Set Delay1Module = TheModel.Modules(j)
        Set Delay2Module = TheModel.Modules(k)
        Set Delay3Module = TheModel.Modules(l)

        Load ResServiceT

        ResServiceT.CSTNP1.Caption = ExpressionsModule.Data("expression(1,5)")
        ResServiceT.CSTNP2.Caption = ExpressionsModule.Data("expression(1,6)")
        ResServiceT.CSTNP3.Caption = ExpressionsModule.Data("expression(1,7)")
        ResServiceT.CSTNP4.Caption = ExpressionsModule.Data("expression(1,8)")
        ResServiceT.CSTDP1.Caption = ExpressionsModule.Data("expression(1,1)")
        ResServiceT.CSTDP2.Caption = ExpressionsModule.Data("expression(1,2)")
        ResServiceT.CSTDP3.Caption = ExpressionsModule.Data("expression(1,3)")
        ResServiceT.CSTDP4.Caption = ExpressionsModule.Data("expression(1,4)")
        ResServiceT.CSTAT.Caption = ExpressionsModule.Data("expression(1,15)")
        ResServiceT.CSTDT.Caption = ExpressionsModule.Data("expression(1,16)")
        ResServiceT.CSTRR.Caption = Delay1Module.Data("Duration")
        ResServiceT.CSTOT1.Caption = Delay2Module.Data("Duration")
        ResServiceT.CSTOT2.Caption = Delay3Module.Data("Duration")
        StopProcess = ResServiceT.CancelButton.Cancel

        ResServiceT.Show

        If StopProcess = False Then

'The values collected from ResServiceT form should be transferred to the variables and
'then to the Arena Model

            STNP1 = ResServiceT.NSTNP1.Text
            STNP2 = ResServiceT.NSTNP2.Text
            STNP3 = ResServiceT.NSTNP3.Text
            STNP4 = ResServiceT.NSTNP4.Text
            STDP1 = ResServiceT.NSTDP1.Text
            STDP2 = ResServiceT.NSTDP2.Text
            STDP3 = ResServiceT.NSTDP3.Text
            STDP4 = ResServiceT.NSTDP4.Text
            STRR = ResServiceT.NSTRR.Text
            STAT = ResServiceT.NSTAT.Text
            STDT = ResServiceT.NSTDT.Text
            STOT1 = ResServiceT.NSTOT1.Text
            STOT2 = ResServiceT.NSTOT2.Text

'Inserting new service times in the EXPRESSIONS element

            ExpressionsModule.Data("expression(1,5)") = STNP1
            ExpressionsModule.Data("expression(1,6)") = STNP2
            ExpressionsModule.Data("expression(1,7)") = STNP3
            ExpressionsModule.Data("expression(1,8)") = STNP4
            ExpressionsModule.Data("expression(1,1)") = STDP1
            ExpressionsModule.Data("expression(1,2)") = STDP2
            ExpressionsModule.Data("expression(1,3)") = STDP3
            ExpressionsModule.Data("expression(1,4)") = STDP4
            ExpressionsModule.Data("expression(1,15)") = STAT
            ExpressionsModule.Data("expression(1,16)") = STDT
            ExpressionsModule.UpdateShapes

'Inserting other service times into the corresponding DELAY blocks

            Delay1Module.Data("Duration") = STRR
            Delay1Module.UpdateShapes

            Delay2Module.Data("Duration") = STOT1
            Delay2Module.UpdateShapes

            Delay3Module.Data("Duration") = STOT2
            Delay3Module.UpdateShapes

            Unload ResServiceT

        End If
    End If

    If changetype = ChangeRT Then
'To show the current routing times on the RoutTimes form as reference to the user.

        i = TheModel.Modules.Find(smFindTag, "ETT")
        j = TheModel.Modules.Find(smFindTag, "ETTR")
        k = TheModel.Modules.Find(smFindTag, "TRTT")
        l = TheModel.Modules.Find(smFindTag, "TRTR")
        o = TheModel.Modules.Find(smFindTag, "RTW")
        p = TheModel.Modules.Find(smFindTag, "WTT")

        Set Rout1Module = TheModel.Modules(i)
        Set Rout2Module = TheModel.Modules(j)
        Set Rout3Module = TheModel.Modules(k)
        Set Rout4Module = TheModel.Modules(l)
        Set Rout5Module = TheModel.Modules(o)
        Set Rout6Module = TheModel.Modules(p)

        Load RoutTimes

        RoutTimes.CRTETT.Caption = Rout1Module.Data("Duration")
        RoutTimes.CRTETTR.Caption = Rout2Module.Data("Duration")
        RoutTimes.CRTTTT.Caption = Rout3Module.Data("Duration")
        RoutTimes.CRTTTR.Caption = Rout4Module.Data("Duration")
        RoutTimes.CRTRTW.Caption = Rout5Module.Data("Duration")
        RoutTimes.CRTWTT.Caption = Rout6Module.Data("Duration")
        StopProcess = RoutTimes.CancelButton.Cancel

        RoutTimes.Show

        If StopProcess = False Then

            RTETT = RoutTimes.NRTETT.Text
            RTETTR = RoutTimes.NRTETTR.Text
            RTTTT = RoutTimes.NRTTTT.Text
            RTTTR = RoutTimes.NRTTTR.Text
            RTRTW = RoutTimes.NRTRTW.Text
            RTWTT = RoutTimes.NRTWTT.Text

 'Inserting new routing times into the ROUTE blocks in the model

            Rout1Module.Data("Duration") = RTETT
            Rout1Module.UpdateShapes

            Rout2Module.Data("Duration") = RTETTR
            Rout2Module.UpdateShapes

            Rout3Module.Data("Duration") = RTTTT
            Rout3Module.UpdateShapes

            Rout4Module.Data("Duration") = RTTTR
            Rout4Module.UpdateShapes

            Rout5Module.Data("Duration") = RTRTW
            Rout5Module.UpdateShapes

            Rout6Module.Data("Duration") = RTWTT
            Rout6Module.UpdateShapes

            Unload RoutTimes

        End If
    End If

Loop
End Sub
```

D.3.2 Subroutine SetScenarios

```
Sub SetScenarios()

'******************************************************************************************
'Once the user has inputted the new capacities and/or service times and/or routing times
'it is necessary to set the stage for re-training process by generating all possible
'combinations of the capacities' values
'Defining local variables
    Dim TotalRep As Integer
    Dim StepSize1, StepSize2, StepSize3 As Integer
    Dim NRep1, NRep2, NRep3 As Integer
    Dim indexloop1, indexloop2, indexloop3 As Integer
    Dim IndexLoopI, IndexLoopII, IndexLoopIII As Integer
    Dim IndexLoopA, IndexLoopB, IndexLoopC As Integer

    NRep1 = 0: NRep2 = 0: NRep3 = 0

'Connecting to the model to extract the value of the variables

    i = TheModel.Modules.Find(smFindTag, "recursos")
    j = TheModel.Modules.Find(smFindTag, "variables")

    Set ResourceModule = TheModel.Modules(i)
    Set VariableModule = TheModel.Modules(j)

    MinRes1 = Val(VariableModule.Data("value(1,6)"))
    MaxRes1 = Val(VariableModule.Data("value(1,1)"))
    MinRes2 = Val(VariableModule.Data("value(1,7)"))
    MaxRes2 = Val(VariableModule.Data("value(1,2)"))
    MinRes3 = Val(VariableModule.Data("value(1,8)"))
    MaxRes3 = Val(VariableModule.Data("value(1,3)"))

    indexloop1 = MinRes1: indexloop2 = MinRes2: indexloop3 = MinRes3

'The total number of replications must be determined.
'The Step Size is variable and it depends on the range of the capacities.
'For each resource the total number of possible iterations is calculated.

    If (MaxRes1 - MinRes1) <= 5 Then
        StepSize1 = 1
        NRep1 = (MaxRes1 - MinRes1) + 1
    Else
        StepSize1 = CLng((MaxRes1 - MinRes1) / 4)
        If (MaxRes1 - MinRes1) = 6 Then
            NRep1 = 4
        Else
            If MinRes1 + 4 * StepSize1 >= MaxRes1 Then
                NRep1 = 5
            Else
                NRep1 = 6
            End If
        End If
    End If
    If (MaxRes2 - MinRes2) <= 5 Then
        StepSize2 = 1
        NRep2 = (MaxRes2 - MinRes2) + 1
    Else
        StepSize2 = CLng((MaxRes2 - MinRes2) / 4)
        If (MaxRes2 - MinRes2) = 6 Then
            NRep2 = 4
        Else
            If MinRes2 + 4 * StepSize2 >= MaxRes2 Then
                NRep2 = 5
            Else
                NRep2 = 6
            End If
            End If
        End If
        If (MaxRes3 - MinRes3) <= 4 Then
            StepSize3 = 1
            NRep3 = (MaxRes3 - MinRes3) + 1
        Else
            StepSize3 = CLng((MaxRes3 - MinRes3) / 4)
            If (MaxRes3 - MinRes3) = 6 Then
                NRep3 = 4
            Else
                If MinRes3 + 4 * StepSize3 >= MaxRes3 Then
                    NRep3 = 5
                Else
                    NRep3 = 6
                End If
            End If
        End If

    'The total number of replications can be calculated

        TotalComb = NRep1 * NRep2 * NRep3
        TotalRep = TotalComb * 15

    'The following command will store the total number of combinations in an ARENA variable.

        j = TheModel.Modules.Find(smFindTag, "variables")

        Set VariableModule = TheModel.Modules(j)
        VariableModule.Data("value(1,4)") = TotalComb

    'The Replicate element must be found and modified.

        i = TheModel.Modules.Find(smFindTag, "Replicas")

        Set ReplicaModule = TheModel.Modules(i)
        ReplicaModule.Data("NumReps") = TotalRep
        ReplicaModule.UpdateShapes

    'Now the combinations must be storaged in a file for later use.
    'If the value of CLng((MaxRes - MinRes)/4) is equal to zero, it must be replaced by 1 in
    order
    'to avoid infinite looping

        Open "C:\ThesisImportant\Combination.txt" For Output As #2

        For indexloop1 = MinRes1 To MaxRes1 Step StepSize1
            For indexloop2 = MinRes2 To MaxRes2 Step StepSize2
                For indexloop3 = MinRes3 To MaxRes3 Step StepSize3
                    Print #2, indexloop1, indexloop2, indexloop3
                Next indexloop3
            Next indexloop2
        Next indexloop1
        If (indexloop3 - MaxRes3) < StepSize3 Then
            For IndexLoopI = MinRes1 To MaxRes1 Step StepSize1
                For IndexLoopII = MinRes2 To MaxRes2 Step StepSize2
                    Print #2, IndexLoopI, IndexLoopII, MaxRes3
                Next IndexLoopII
            Next IndexLoopI
        End If
        If (indexloop2 - MaxRes2) < StepSize2 Then
            For IndexLoopI = MinRes1 To MaxRes1 Step StepSize1
                For IndexLoopIII = MinRes3 To MaxRes3 Step StepSize3
                    Print #2, IndexLoopI, MaxRes2, IndexLoopIII
                Next IndexLoopIII
            Next IndexLoopI
        End If
        If (indexloop1 - MaxRes1) < StepSize1 Then
            For IndexLoopII = MinRes2 To MaxRes2 Step StepSize2
                For IndexLoopIII = MinRes3 To MaxRes3 Step StepSize3
                Print #2, MaxRes1, IndexLoopII, IndexLoopIII
            Next IndexLoopIII
        Next IndexLoopII
    End If
    If (indexloop3 - MaxRes3) < StepSize3 And (indexloop2 - MaxRes2) < StepSize2 And
(indexloop1 - MaxRes1) < StepSize1 Then
        Print #2, MaxRes1, MaxRes2, MaxRes3
    End If
    If (indexloop3 - MaxRes3) < StepSize3 And (indexloop2 - MaxRes2) < StepSize2 Then
        For IndexLoopA = MinRes1 To MaxRes1 Step StepSize1
            Print #2, IndexLoopA, MaxRes2, MaxRes3
        Next IndexLoopA
    End If
    If (indexloop3 - MaxRes3) < StepSize3 And (indexloop1 - MaxRes1) < StepSize1 Then
        For IndexLoopB = MinRes2 To MaxRes2 Step StepSize2
            Print #2, MaxRes1, IndexLoopB, MaxRes3
        Next IndexLoopB
    End If
    If (indexloop2 - MaxRes2) < StepSize2 And (indexloop1 - MaxRes1) < StepSize1 Then
        For IndexLoopC = MinRes3 To MaxRes3 Step StepSize3
            Print #2, MaxRes1, MaxRes2, IndexLoopC
        Next IndexLoopC
    End If

    Close #2

'Initializing the w Record

    j = TheModel.Modules.Find(smFindTag, "variables")
    Set VariableModule = TheModel.Modules(j)
    VariableModule.Data("value(1,5)") = 1

End Sub
```

D.3.3 Subroutine CheckReplications

```
Public Sub CheckReplications()

'*************************************************************************************
'This subroutine is in charge of checking the number of replications and storing
'the statistics of interest when required
'vNRep, vMRep: Store current replication value, and the maximum number of replications
'CapaNurses, CapaDoctors, CapaBeds: Store the level of each one of the resources
'AvgTimeQNurses, AvgTimeQDoctors, AvgTimeQBeds, AvgUtilNurses, AvgUtilDoctors,
'AvgUtilBeds: Take the outputs of the simulation model every 15 replications
'NursesC, DoctorsC, BedsC: Change the level of the resources every 15 replications
'Tiempo, Hora: Establish stopping time of the total number of replications

    Dim vNRep, vMRep As Integer
    Dim CapaNurses, CapaDoctors, CapaBeds As Double
    Dim AvgTimeQNurses, AvgTimeQDoctors, AvgTimeQBeds, AvgUtilNurses As Double
    Dim AvgUtilDoctors, AvgUtilBeds, NursesC, DoctorsC, BedsC As Double
    Dim Tiempo, Hora

'Connecting to the model

    Set TheModel = ThisDocument.Model
    Set OUT = TheModel.SIMAN

    vNRep = OUT.RunCurrentReplication
    vMRep = OUT.RunMaximumReplications

    j = TheModel.Modules.Find(smFindTag, "variables")
    Set VariableModule = TheModel.Modules(j)
    w = VariableModule.Data("value(1,5)")
    If vNRep = 15 * w Then

        If w = 1 Then

'The values of the different possible combinations must be moved from the text file
'created in RunBegin to a VBA array one time.
'First we have to find the size of the array from the Variables Element

            TotalComb = VariableModule.Data("value(1,4)")
            ReDim Combo(1 To TotalComb, 3)

'Second the information is passed from the file to the array

            Open "C:\ThesisImportant\Combination.txt" For Input As #3
            For i = 1 To TotalComb
                Input #3, Combo(i, 1), Combo(i, 2), Combo(i, 3)
            Next i
            Close #3

'The ASCII file where the training data is stored must be opened once for Output
(creation)
'and the columns' labels must be written
            Open "C:\ThesisImportant\TrainingData.txt" For Output As #1
            Print #1, "NoBeds", "NoDoc", "NoNur", "AveQNur", "UtilNur", "AveQDoc",
"UtilDoc", "AveQBed", "UtilBed"

'This is to store the statistics on the partial average results.
'First we have to find the modules and take the information about the capacities.

            i = TheModel.Modules.Find(smFindTag, "recursos")
            Set ResourceModule = TheModel.Modules(i)

            CapaNurses = ResourceModule.Data("CapExp(1)")
            CapaDoctors = ResourceModule.Data("CapExp(2)")
            CapaBeds = ResourceModule.Data("CapExp(3)")

'Second we have to collect information about the measures of performance

            AvgTimeQNurses = OUT.TallyAverage(8) + OUT.TallyAverage(9) +
OUT.TallyAverage(10)
            AvgTimeQDoctors = OUT.TallyAverage(11)
            AvgTimeQBeds = OUT.TallyAverage(1) + OUT.TallyAverage(2) +
OUT.TallyAverage(3)
            AvgUtilNurses = OUT.DStatAverage(14)
            AvgUtilDoctors = OUT.DStatAverage(15)
            AvgUtilBeds = OUT.DStatAverage(16)

'Every 15 records the following values must be stored.

            Print #1, CapaBeds, CapaDoctors, CapaNurses, AvgTimeQNurses, AvgUtilNurses,
AvgTimeQDoctors, AvgUtilDoctors, AvgTimeQBeds, AvgUtilBeds
            Close #1

'After the values have been stored, the tallies and dstats must be cleared

            OUT.StatisticsClearAll

'The new combination of capacities must be replaced on the Resources Element

            If w <> TotalComb Then

                ResourceModule.Data("CapExp(1)") = Val(Combo(w + 1, 1))
                ResourceModule.Data("CapExp(2)") = Val(Combo(w + 1, 2))
                ResourceModule.Data("CapExp(3)") = Val(Combo(w + 1, 3))
                ResourceModule.UpdateShapes

                w = w + 1
```

```
'The value of w must be updated in the VARIABLES Element to use it later

                VariableModule.Data("value(1,5)") = w
            End If

        Else

'After the ASCII file has been created it is opened for Append every 15 replications
            Open "C:\ThesisImportant\TrainingData.txt" For Append As #1

'This is to store the statistics on the partial average results.
'First we have to find the modules and take the information about the capacities.

            i = TheModel.Modules.Find(smFindTag, "recursos")
            Set ResourceModule = TheModel.Modules(i)

            CapaNurses = ResourceModule.Data("CapExp(1)")
            CapaDoctors = ResourceModule.Data("CapExp(2)")
            CapaBeds = ResourceModule.Data("CapExp(3)")

'Second we have to collect information about the measures of performance

            AvgTimeQNurses = OUT.TallyAverage(8) + OUT.TallyAverage(9) +
OUT.TallyAverage(10)
            AvgTimeQDoctors = OUT.TallyAverage(11)
            AvgTimeQBeds = OUT.TallyAverage(1) + OUT.TallyAverage(2) +
OUT.TallyAverage(3)
            AvgUtilNurses = OUT.DStatAverage(14)
            AvgUtilDoctors = OUT.DStatAverage(15)
            AvgUtilBeds = OUT.DStatAverage(16)

'Every 15 records the following values must be stored.

            Print #1, CapaBeds, CapaDoctors, CapaNurses, AvgTimeQNurses, AvgUtilNurses,
AvgTimeQDoctors, AvgUtilDoctors, AvgTimeQBeds, AvgUtilBeds
            Close #1

'After the values have been stored, the tallies and dstats must be cleared

            OUT.StatisticsClearAll

'The new combination of capacities must be replaced on the Resources Element

            If w <> TotalComb Then

                ResourceModule.Data("CapExp(1)") = Val(Combo(w + 1, 1))
                ResourceModule.Data("CapExp(2)") = Val(Combo(w + 1, 2))
                ResourceModule.Data("CapExp(3)") = Val(Combo(w + 1, 3))
                ResourceModule.UpdateShapes

                w = w + 1

'The value of w must be updated in the VARIABLES Element to use it later

                VariableModule.Data("value(1,5)") = w
            End If
        End If
    End If

'If it ws the last replication, train ANNs
    If vNRep = vMRep Then
'Feedback for the user
        Tiempo = Now
        MsgBox Tiempo
        MsgBox "Training file has been generated"

        Call NN2Training

'Feedback for the user
```

```
        Hora = Now
        MsgBox Hora
        MsgBox "The new networks have been trained and you can use them now"
        flag = 1
    End If

End Sub
```

D.3.4 Subroutine CheckingRanges

```
Sub CheckingRanges()

'Checks if retraining is really needed or not
    If Val(ResCapacity.MinNewCNurses.Text) >= MinRes1 And
Val(ResCapacity.MaxNewCNurses.Text) <= MaxRes1 Then
        If Val(ResCapacity.MinNewCDoctors.Text) >= MinRes2 And
Val(ResCapacity.MaxNewCDoctors.Text) <= MaxRes2 Then
            If Val(ResCapacity.MinNewCBeds.Text) >= MinRes3 And
Val(ResCapacity.MaxNewCBeds.Text) <= MaxRes3 Then
                MsgBox "These ranges of level of resources do not require to re-train the
network." & "You can use the current ANN if these are your only changes."
                ranges = 1
            End If
        End If
    End If

End Sub
```

D.4 VBA NCR Subroutines

D.4.1 Subroutine TrainingNN2

```
Sub NN2Training()

'**************************************************************************************
'NNOpen: Variable name for the execution of the NS2.exe application
'IMOpen: Variable name for the execution of the Impascii.exe application
'Testset: Variable name for the execution of the Testset.exe application
'Setinput: Variable name for the execution of Netinput.exe application
'Train: Variable name for the execution of the Begtrain.exe application
'Deff: Variable name for the execution of the Dllprime.exe application
'VBcode: Variable name for the execution of the Srcgen.exe application
'Start: Timer used to allow the system to perform the orders given by sendkeys statement
'Contador: String version of "Contando" used to store each ".def" and ".vb" files
'Contando: Integer value that records which network number is being created ("counting")
'indexloop: Used in the loop to count the number of line
'lineas: Integer value that records how many lines are in the VB file ("lines")
'dummy, reference: Variables used to store temporarly the content of the VB file lines

Dim y, w1, w2, w3, w4, w5, w6, Contando, indexloop, lineas  As Integer
Dim Start
Dim NNOpen, IMOpen, Testset
Dim Setinput, Train, Deff, VBcode
Dim dummy, reference
Dim contador As String

Contando = 0

'Creating the description file ".dsc"

    NNOpen = Shell("C:\NeuroShell 2\Ns2.exe", vbNormalFocus)
    AppActivate NNOpen
    SendKeys "%FNChanges.dsc%S%FS%FX", True

'Importing the training data that have been allocated in the text file

    IMOpen = Shell("C:\NeuroShell 2\Impascii.exe", vbNormalFocus)
    AppActivate IMOpen
    SendKeys "%FATrainingData.txt%O%FPChanges.pat%S %IB", True

    AppActivate "ASCII File Import"

    y = 3 ' Set duration.
    Start = Timer   ' Set start time.
    Do While Timer < Start + y
        DoEvents    ' Yield to other processes.
    Loop

    SendKeys "{ENTER}", True
    SendKeys "%FX", True

'Extracting testing set

    Testset = Shell("C:\NeuroShell 2\Testset.exe", vbNormalFocus)
    AppActivate Testset
    SendKeys "%FSChanges.pat%O%EB", True

    y = 3 ' Set duration.
    Start = Timer   ' Set start time.
    Do While Timer < Start + y
        DoEvents    ' Yield to other processes.
    Loop

    SendKeys "{ENTER}", True
    SendKeys "%FX", True

'Setting inputs and outputs

For w1 = 0 To 1
    For w2 = 0 To 1
        For w3 = 0 To 1
            For w4 = 0 To 1
                For w5 = 0 To 1
                    For w6 = 0 To 1
                        If w1 <> 0 Or w2 <> 0 Or w3 <> 0 Or w4 <> 0 Or w5 <> 0 Or w6 <> 0
Then
                            Setinput = Shell("C:\NeuroShell 2\Netinput.exe",
vbNormalFocus)
                            AppActivate Setinput
                            SendKeys "%FPChanges.pat%O{RIGHT}{ENTER}", True
                            SendKeys "{TAB}A{RIGHT}A{RIGHT}A{RIGHT}", True
                            If w1 = 1 Then
                                SendKeys "I{RIGHT}"
                            Else
                                SendKeys "{BACKSPACE}{RIGHT}"
                            End If
                            If w2 = 1 Then
                                SendKeys "I{RIGHT}"
                            Else
                                SendKeys "{BACKSPACE}{RIGHT}"
                            End If
                            If w3 = 1 Then
                                SendKeys "I{RIGHT}"
                            Else
                                SendKeys "{BACKSPACE}{RIGHT}"
                            End If
                            If w4 = 1 Then
                                SendKeys "I{RIGHT}"
                            Else
                                SendKeys "{BACKSPACE}{RIGHT}"
                            End If
                            If w5 = 1 Then
                                SendKeys "I{RIGHT}"
                            Else
                                SendKeys "{BACKSPACE}{RIGHT}"
                            End If
                            If w6 = 1 Then
                                SendKeys "I"
                            Else
                                SendKeys "{BACKSPACE}"
                            End If
                            SendKeys "%SC", True

                            y = 3 ' Set duration.
                            Start = Timer    ' Set start time.
                            Do While Timer < Start + y
                                DoEvents    ' Yield to other processes.
                            Loop

                            SendKeys "%FX", True

'Training a simple NN

                            Train = Shell("C:\NeuroShell 2\Begtrain.exe", vbNormalFocus)
                            AppActivate Train
                            SendKeys "%FSChanges.pat%O{RIGHT}{TAB
5}{ENTER}%TS{RIGHT}{ENTER}", True

                            y = 45  'Set duration to 45 seconds.
                            Start = Timer    ' Set start time.
                            Do While Timer < Start + y
                                DoEvents    ' Yield to other processes.
                            Loop

                            SendKeys "%TI", True

                            y = 3 ' Set duration.
                            Start = Timer   ' Set start time.
                            Do While Timer < Start + y
                                DoEvents    ' Yield to other processes.
                            Loop

                            SendKeys "{ENTER}", True
                            SendKeys "%FE", True

                            Contando = Contando + 1

                            contador = CStr(Contando)

'Generating .def file

                            Deff = Shell("C:\NeuroShell 2\Dllprime.exe", vbNormalFocus)
                            AppActivate Deff
                            SendKeys "%FCChanges.fig%O%FDChanges", True
                            SendKeys contador, True
                            SendKeys ".def", True
                            SendKeys "%S%GG", True

                            y = 15 ' Set duration.
                            Start = Timer   ' Set start time.
                            Do While Timer < Start + y
                                DoEvents    ' Yield to other processes.
                            Loop

                            SendKeys "{ENTER}", True
                            SendKeys "%FX", True

'Generating VB code

                            VBcode = Shell("C:\NeuroShell 2\Srcgen.exe", vbNormalFocus)
                            AppActivate VBcode
                            SendKeys "{DOWN}%FCChanges.fig%O%FSChanges1", True
                            SendKeys contador, True
                            SendKeys ".vb", True
                            SendKeys "%S%GG", True

                            y = 15 ' Set duration.
                            Start = Timer   ' Set start time.
                            Do While Timer < Start + y
                                DoEvents    ' Yield to other processes.
                            Loop

                            SendKeys "{ENTER}", True
                            SendKeys "%FX", True

'Cleaning the VB code to remove innecessary information

                            Open "C:\ThesisImportant\Changes1" + contador + ".vb" For
Input As #100
                                lineas = 0
                                Do While Not EOF(100)
                                    Line Input #100, reference
                                    lineas = lineas + 1
                                Loop
                            Close #100
                            Open "C:\ThesisImportant\Changes1" + contador + ".vb" For
Input As #101
                            Open "C:\ThesisImportant\Changes" + contador + ".vb" For
Output As #102
```

```
                            Open "C:\ThesisImportant\Dummy.vb" For Output As #103
                                For indexloop = 1 To lineas
                                    If indexloop < 7 Then
                                        Line Input #101, dummy
                                        Print #103, dummy
                                    End If
                                    If (indexloop >= 7) And (indexloop <= (lineas - 13))
Then

                                        Line Input #101, dummy
                                        Print #102, dummy
                                    End If
                                    If (indexloop > (lineas - 13)) And (indexloop <
(lineas - 9)) Then

                                        Line Input #101, dummy
                                        Print #103, dummy
                                    End If
                                    If (indexloop = (lineas - 9)) Then
                                        Line Input #101, dummy
                                        Print #102, dummy
                                    End If
                                    If (indexloop > (lineas - 9)) And (indexloop <
(lineas - 5)) Then

                                        Line Input #101, dummy
                                        Print #103, dummy
                                    End If
                                    If (indexloop = (lineas - 5)) Then
                                        Line Input #101, dummy
                                        Print #102, dummy
                                    End If
                                    If (indexloop > (lineas - 5)) Then
                                        Line Input #101, dummy
                                        Print #103, dummy
                                    End If
                                Next indexloop
                            Close #101: Close #102: Close #103
                            Kill "C:\ThesisImportant\Changes1" + contador + ".vb"
                            Kill "C:\ThesisImportant\Dummy.vb"

                        End If
                    Next w6
                Next w5
            Next w4
        Next w3
    Next w2
Next w1

End Sub
```

D.4.2 Subroutine SelectingNetworks

```
Sub SelectingNetwork()

'*****************************************************************************************
'Declaring local variables
'Networkarray Array that stored the different selection of inputs that could occur
'mycounter Variable used to set the position of any array in the Networkarray
'Referencia Used to create the Networkarray (Referencia, spanish work for Reference)
    Dim y1, y2, y3, y4, y5, y6 As Integer
    Dim Referencia As String
    Dim Networkarray(1 To 63)
    Dim mycounter, z As Integer

'Variables used to generated all possible combinations of inputs and to detec
'which inputs were selected by the user
```

```
mycounter = 1
For y1 = 0 To 1
    For y2 = 0 To 1
        For y3 = 0 To 1
            For y4 = 0 To 1
                For y5 = 0 To 1
                    For y6 = 0 To 1
                        If y1 <> 0 Or y2 <> 0 Or y3 <> 0 Or y4 <> 0 Or y5 <> 0 Or y6 <> 0
Then
                            Referencia = CStr(y1) + CStr(y2) + CStr(y3) + CStr(y4) +
CStr(y5) + CStr(y6)
                            If NeuralNetwork = Referencia Then NNnumber = mycounter
                            mycounter = mycounter + 1
                        End If
                    Next y6
                Next y5
            Next y4
        Next y3
    Next y2
Next y1

End Sub
```

D.4.3 Subroutine NN2Using

```
Sub NN2Using(unnecessary As Integer)

'*****************************************************************************************
'Declaring local variables
'Flags used to detect the value of the inputs selected by the user and to store these
values in the inarray
    Dim y%
    Dim loopIndex As Integer
    Dim Flags(1 To 6)
    Dim out1, out2, out3 As Integer
    Dim indexloop, NumberInputs As Integer

'Connecting to NeuroShell and opening the requested Neural Network

    If Not netisopen Then
        y = OpenNet("C:\ThesisImportant\Changes" + CStr(NNnumber) + ".def", netnumber,
inputs, output)
        If y > 0 Then
            MsgBox "Error returned from OpenNet: " + CStr(y) + ".", 16, "Error"
        End If
    End If

'Retrieving values from ENTRY form as Inputs and loading these values in the OUTS form
ReDim inarray#(inputs)
ReDim outarray#(output)

Load OUTS

'The labels are not active unless the specific input is selected to fire the network
OUTS.TNLabel.Enabled = False
OUTS.UNLabel.Enabled = False
OUTS.TDLabel.Enabled = False
OUTS.UDLabel.Enabled = False
OUTS.TBLabel.Enabled = False
OUTS.UBLabel.Enabled = False

For loopIndex = 1 To inputs
    If ENTRY.QNurses.Text <> "" Then
        inarray(loopIndex) = Val(ENTRY.QNurses.Text)
        OUTS.TNLabel.Enabled = True
```

```
            OUTS.TNOUTS.Caption = Val(ENTRY.QNurses.Text)
            ENTRY.QNurses.Text = ""
        Else
            If ENTRY.UNurses.Text <> "" Then
                inarray(loopIndex) = Val(ENTRY.UNurses.Text)
                OUTS.UNLabel.Enabled = True
                OUTS.UNOUTS.Caption = Val(ENTRY.UNurses.Text)
                ENTRY.UNurses.Text = ""
            Else
                If ENTRY.QMD.Text <> "" Then
                    inarray(loopIndex) = Val(ENTRY.QMD.Text)
                    OUTS.TDLabel.Enabled = True
                    OUTS.TDOUTS.Caption = Val(ENTRY.QMD.Text)
                    ENTRY.QMD.Text = ""
                Else
                    If ENTRY.UMD.Text <> "" Then
                        inarray(loopIndex) = Val(ENTRY.UMD.Text)
                        OUTS.UDLabel.Enabled = True
                        OUTS.UDOUTS.Caption = Val(ENTRY.UMD.Text)
                        ENTRY.UMD.Text = ""
                    Else
                        If ENTRY.QBed.Text <> "" Then
                            inarray(loopIndex) = Val(ENTRY.QBed.Text)
                            OUTS.TBLabel.Enabled = True
                            OUTS.TBOUTS.Caption = Val(ENTRY.QBed.Text)
                            ENTRY.QBed.Text = ""
                        Else
                            If ENTRY.UBed.Text <> "" Then
                                inarray(loopIndex) = Val(ENTRY.UBed.Text)
                                OUTS.UBLabel.Enabled = True
                                OUTS.UBOUTS.Caption = Val(ENTRY.UBed.Text)
                                ENTRY.UBed.Text = ""
                            End If
                        End If
                    End If
                End If
            End If
        End If
    Next

'Actual firing of the NN

    y = FireNet(netnumber, inarray(1), outarray(1))

    If y > 0 Then
        MsgBox "Error returned from FireNet:" + CStr(y) + ".", 16, "Error"
    End If

'Loading the results in the OUTS form
    If unnecessary = 0 Then
        If outarray(3) <= MinRes1 + 1 Then
            OUTS.Nurses.Caption = MinRes1 + 1
        Else
            If outarray(3) > MaxRes1 - 1 Then
                OUTS.Nurses.Caption = "N/A"
                OUTS.InfeasibleNurses.Caption = "Your current upper bound of " + (MaxRes1
- 1) + " in number of nurses does not allow to achieve your goal"
            Else
                OUTS.Nurses.Caption = Format$(outarray(3), "00")
                OUTS.InfeasibleNurses.Caption = "N/A"
            End If
        End If
        If outarray(2) <= MinRes2 + 1 Then
            OUTS.Doctors.Caption = MinRes2 + 1
        Else
            If outarray(2) > MaxRes2 - 1 Then
                OUTS.Doctors.Caption = "N/A"
                OUTS.InfeasibleMD.Caption = "Your current upper bound of " + (MaxRes2 -
1) + " in number of doctors does not allow to achieve your goal"
```

```
            Else
                OUTS.Doctors.Caption = Format$(outarray(2), "00")
                OUTS.InfeasibleMD.Caption = "N/A"
            End If
        End If
        If outarray(1) <= MinRes3 + 1 Then
            OUTS.Beds.Caption = MinRes3 + 1
        Else
            If outarray(1) > MaxRes3 - 1 Then
                OUTS.Beds.Caption = "N/A"
                OUTS.InfeasibleBeds.Caption = "Your current upper bound of " + (MaxRes3 -
1) + " in number of beds does not allow to achieve your goal"
            Else
                OUTS.Beds.Caption = Format$(outarray(1), "00")
                OUTS.InfeasibleBeds.Caption = "N/A"
            End If
        End If
    Else
        If outarray(3) <= MinRes1 + 1 Then
            OUTS.Nurses.Caption = MinRes1 + 1
        Else
            If outarray(3) > MaxRes1 - 1 Then
                OUTS.Nurses.Caption = Format$(outarray(3), "00")
                OUTS.InfeasibleNurses.Caption = "Your current upper bound of " + (MaxRes1
- 1) + " in number of nurses does not allow to achieve your goal.  Suggested value is
displayed for your reference."
            Else
                OUTS.Nurses.Caption = Format$(outarray(3), "00")
                OUTS.InfeasibleNurses.Caption = "N/A"
            End If
        End If
        If outarray(2) <= MinRes2 + 1 Then
            OUTS.Doctors.Caption = MinRes2 + 1
        Else
            If outarray(2) > MaxRes2 - 1 Then
                OUTS.Doctors.Caption = Format$(outarray(2), "00")
                OUTS.InfeasibleMD.Caption = "Your current upper bound of " + (MaxRes2 -
1) + " in number of doctors does not allow to achieve your goal.  Suggested value is
displayed for your reference."
            Else
                OUTS.Doctors.Caption = Format$(outarray(2), "00")
                OUTS.InfeasibleMD.Caption = "N/A"
            End If
        End If
        If outarray(1) <= MinRes3 + 1 Then
            OUTS.Beds.Caption = MinRes3 + 1
        Else
            If outarray(1) > MaxRes3 - 1 Then
                OUTS.Beds.Caption = Format$(outarray(1), "00")
                OUTS.InfeasibleBeds.Caption = "Your current upper bound of " + (MaxRes3 -
1) + " in number of beds does not allow to achieve your goal.  Suggested value is
displayed for your reference."
            Else
                OUTS.Beds.Caption = Format$(outarray(1), "00")
                OUTS.InfeasibleBeds.Caption = "N/A"
            End If
        End If
    End If

'Closing the NN after it has been used

    y = CloseNet(netnumber)

End Sub
```

D.4.4 Subroutine RunNewANN

```
Sub RunNewANN()
'Declaration of local variables
    Dim StopProcess As Boolean
    Dim MoreRuns, unnecessary As Integer
    unnecessary = 0
    MoreRuns = 1

    Do While MoreRuns = 1
        Load Aftertraining
        Aftertraining.Show

        StopProcess = Aftertraining.CancelButton.Cancel

        Unload Aftertraining

        If StopProcess Then
            MsgBox "Thanks for using this program"
            MoreRuns = 0
        Else
            Call NN2Querying(unnecessary)
        End If
    Loop

End Sub
```

D.4.5 Subroutine NN2Querying

```
Sub NN2Querying(unnecessary As Integer)

'Declarin local variables
    Dim StopProcess As Boolean

'This routine takes all the actions related with querying the already trained networks
    Set TheModel = ThisDocument.Model
    Set OUT = TheModel.SIMAN

    j = TheModel.Modules.Find(smFindTag, "variables")
    Set VariableModule = TheModel.Modules(j)

    Load ENTRY

    ENTRY.MinCurCNurses.Caption = VariableModule.Data("value(1,6)") + 1
    ENTRY.MaxCurCNurses.Caption = VariableModule.Data("value(1,1)") - 1
    ENTRY.MinCurCDoctors.Caption = VariableModule.Data("value(1,7)") + 1
    ENTRY.MaxCurCDoctors.Caption = VariableModule.Data("value(1,2)") - 1
    ENTRY.MinCurCBeds.Caption = VariableModule.Data("value(1,8)") + 1
    ENTRY.MaxCurCBeds.Caption = VariableModule.Data("value(1,3)") - 1

    ENTRY.Show

    StopProcess = ENTRY.CancelButton.Cancel

'ENTRY form remains loaded but hiddent for th use of the following routines
    If Not StopProcess Then
        Call SelectingNetwork
        Call NN2Using(unnecessary)

        OUTS.Show
        Unload OUTS
    End If
End Sub
```

D.5 VBA Support Subroutines

```
Sub Letter(tecla As MSForms.ReturnInteger)

'Checks to make sure numeric values are entered
'tecla = Spanish work for "key", checks ASCII values of keys pressed

    If tecla < 46 Or tecla > 57 Then
        MsgBox "Please enter a numeric value."
        tecla = 0
    End If

End Sub


Sub LetterInt(tecla As MSForms.ReturnInteger)

'Checks to make sure integer values are entered

    If tecla < 48 Or tecla > 57 Then
        MsgBox "Please enter an integer value."
        tecla = 0
    End If

End Sub
```

APPENDIX E

E.1 Example of VB Code Generated by NeuroShell 2

```
Dim netsum As Double
    Static feature2(14) As Double

    'outarray(1) is #_beds
    'outarray(2) is #_doctors
    'outarray(3) is #_nurses
    'inarray(1) is AveQNurse
    'inarray(2) is Util_Nurse
    'inarray(3) is AveQMD
    'inarray(4) is UtilMD
    'inarray(5) is AveQBed
    'inarray(6) is UtilBed

    If (InArray(1) < 0) Then InArray(1) = 0
    If (InArray(1) > 10.359) Then InArray(1) = 10.359
    InArray(1) = InArray(1) / 10.359

    If (InArray(2) < 0.468) Then InArray(2) = 0.468
    If (InArray(2) > 1) Then InArray(2) = 1
    InArray(2) = (InArray(2) - 0.468) / 0.532

    If (InArray(3) < 0) Then InArray(3) = 0
    If (InArray(3) > 2.637) Then InArray(3) = 2.637
    InArray(3) = InArray(3) / 2.637

    If (InArray(4) < 0.106) Then InArray(4) = 0.106
    If (InArray(4) > 0.979) Then InArray(4) = 0.979
    InArray(4) = (InArray(4) - 0.106) / 0.873

    If (InArray(5) < 0.465) Then InArray(5) = 0.465
    If (InArray(5) > 5031) Then InArray(5) = 5031
    InArray(5) = (InArray(5) - 0.465) / 5030.535

    If (InArray(6) < 0.553) Then InArray(6) = 0.553
    If (InArray(6) > 1) Then InArray(6) = 1
    InArray(6) = (InArray(6) - 0.553) / 0.447

    netsum = 0.1042796
    netsum = netsum + InArray(1) * 1.006627
    netsum = netsum + InArray(2) * -0.6878272
    netsum = netsum + InArray(3) * 0.9260939
    netsum = netsum + InArray(4) * 1.469276
    netsum = netsum + InArray(5) * 0.7912758
    netsum = netsum + InArray(6) * 0.3663329
    feature2(1) = 1 / (1 + Exp(-netsum))

    netsum = -0.3040561
    netsum = netsum + InArray(1) * 0.7371953
    netsum = netsum + InArray(2) * 2.447581
    netsum = netsum + InArray(3) * 0.5535823
    netsum = netsum + InArray(4) * 0.9191557
    netsum = netsum + InArray(5) * -0.6306052
    netsum = netsum + InArray(6) * -0.3387515
    feature2(2) = 1 / (1 + Exp(-netsum))

    netsum = 0.8736666
    netsum = netsum + InArray(1) * -4.909794
    netsum = netsum + InArray(2) * 0.1738508
    netsum = netsum + InArray(3) * -0.2231777
netsum = netsum + InArray(4) * 0.1888949
netsum = netsum + InArray(5) * -3.22939
netsum = netsum + InArray(6) * -2.364963
feature2(3) = 1 / (1 + Exp(-netsum))

netsum = 1.376303
netsum = netsum + InArray(1) * 0.07068385
netsum = netsum + InArray(2) * -0.0350857
netsum = netsum + InArray(3) * -0.7291315
netsum = netsum + InArray(4) * -16.32992
netsum = netsum + InArray(5) * -2.29684
netsum = netsum + InArray(6) * -0.601949
feature2(4) = 1 / (1 + Exp(-netsum))

netsum = -2.001306
netsum = netsum + InArray(1) * 2.38802
netsum = netsum + InArray(2) * -2.828386
netsum = netsum + InArray(3) * -0.5921768
netsum = netsum + InArray(4) * -1.173817
netsum = netsum + InArray(5) * 7.261665
netsum = netsum + InArray(6) * 1.133806
feature2(5) = 1 / (1 + Exp(-netsum))

netsum = -0.5886854
netsum = netsum + InArray(1) * 0.0772596
netsum = netsum + InArray(2) * 4.488652
netsum = netsum + InArray(3) * -0.1507872
netsum = netsum + InArray(4) * -0.9915754
netsum = netsum + InArray(5) * 1.332071
netsum = netsum + InArray(6) * 1.277141
feature2(6) = 1 / (1 + Exp(-netsum))

netsum = -0.7312872
netsum = netsum + InArray(1) * 0.7232761
netsum = netsum + InArray(2) * 0.4206014
netsum = netsum + InArray(3) * 0.2214112
netsum = netsum + InArray(4) * 0.2579994
netsum = netsum + InArray(5) * 0.67212
netsum = netsum + InArray(6) * 0.1677995
feature2(7) = 1 / (1 + Exp(-netsum))

netsum = -0.1207742
netsum = netsum + InArray(1) * -3.367627
netsum = netsum + InArray(2) * -0.5818383
netsum = netsum + InArray(3) * -1.172969
netsum = netsum + InArray(4) * 0.1938878
netsum = netsum + InArray(5) * 5.607226
netsum = netsum + InArray(6) * 2.862845
feature2(8) = 1 / (1 + Exp(-netsum))

netsum = 0.07232291
netsum = netsum + InArray(1) * -1.174228
netsum = netsum + InArray(2) * 0.5826449
netsum = netsum + InArray(3) * -0.01743032
netsum = netsum + InArray(4) * -3.745018
netsum = netsum + InArray(5) * 1.747035
netsum = netsum + InArray(6) * 0.4398864
feature2(9) = 1 / (1 + Exp(-netsum))

netsum = 0.3268138
netsum = netsum + InArray(1) * -2.094425
netsum = netsum + InArray(2) * -1.241517
netsum = netsum + InArray(3) * -0.007554269
netsum = netsum + InArray(4) * 0.8405895
netsum = netsum + InArray(5) * 1.056902
netsum = netsum + InArray(6) * 0.8749657
feature2(10) = 1 / (1 + Exp(-netsum))

netsum = -0.1117487
netsum = netsum + InArray(1) * -7.109506
netsum = netsum + InArray(2) * 1.532805
netsum = netsum + InArray(3) * -0.5597914
netsum = netsum + InArray(4) * 0.3092887
netsum = netsum + InArray(5) * -2.191923
netsum = netsum + InArray(6) * 1.751261
feature2(11) = 1 / (1 + Exp(-netsum))

netsum = 1.153579
netsum = netsum + InArray(1) * -0.6705236
netsum = netsum + InArray(2) * 0.167992
netsum = netsum + InArray(3) * -0.1313259
netsum = netsum + InArray(4) * 2.44497
netsum = netsum + InArray(5) * -0.4890198
netsum = netsum + InArray(6) * 0.2461803
feature2(12) = 1 / (1 + Exp(-netsum))

netsum = -1.707224
netsum = netsum + InArray(1) * 1.425442
netsum = netsum + InArray(2) * 1.828459
netsum = netsum + InArray(3) * 2.210742
netsum = netsum + InArray(4) * 0.7633827
netsum = netsum + InArray(5) * -0.5835367
netsum = netsum + InArray(6) * -3.586008
feature2(13) = 1 / (1 + Exp(-netsum))

netsum = 1.623453
netsum = netsum + InArray(1) * 0.5754997
netsum = netsum + InArray(2) * -0.05540909
netsum = netsum + InArray(3) * -0.3254807
netsum = netsum + InArray(4) * 0.4176272
netsum = netsum + InArray(5) * -5.029757
netsum = netsum + InArray(6) * -2.107037
feature2(14) = 1 / (1 + Exp(-netsum))

netsum = 1.796285
netsum = netsum + feature2(1) * 2.517645
netsum = netsum + feature2(2) * 0.4948037
netsum = netsum + feature2(3) * -5.311374
netsum = netsum + feature2(4) * -0.0650139
netsum = netsum + feature2(5) * -3.724136
netsum = netsum + feature2(6) * 0.3030719
netsum = netsum + feature2(7) * 1.19422
netsum = netsum + feature2(8) * -4.369812
netsum = netsum + feature2(9) * 2.221775
netsum = netsum + feature2(10) * 2.710334
netsum = netsum + feature2(11) * -4.627873
netsum = netsum + feature2(12) * 0.2571157
netsum = netsum + feature2(13) * 3.835396
netsum = netsum + feature2(14) * 3.163106
OutArray(1) = 1 / (1 + Exp(-netsum))

netsum = 0.6977847
netsum = netsum + feature2(1) * -1.129638
netsum = netsum + feature2(2) * -2.135933
netsum = netsum + feature2(3) * -0.5673947
netsum = netsum + feature2(4) * 10.47204
netsum = netsum + feature2(5) * -1.8042
netsum = netsum + feature2(6) * 1.694486
netsum = netsum + feature2(7) * -1.139969
netsum = netsum + feature2(8) * 0.851368
netsum = netsum + feature2(9) * 3.468216
netsum = netsum + feature2(10) * -0.1665898
netsum = netsum + feature2(11) * -1.492202
netsum = netsum + feature2(12) * -1.271609
netsum = netsum + feature2(13) * -0.2631445
netsum = netsum + feature2(14) * 2.396252
OutArray(2) = 1 / (1 + Exp(-netsum))
```

```
    netsum = 1.137034
    netsum = netsum + feature2(1) * 0.01847009
    netsum = netsum + feature2(2) * -1.062954
    netsum = netsum + feature2(3) * -0.4108751
    netsum = netsum + feature2(4) * 0.1466096
    netsum = netsum + feature2(5) * -1.66192
    netsum = netsum + feature2(6) * -3.215984
    netsum = netsum + feature2(7) * -0.6938019
    netsum = netsum + feature2(8) * -0.04847045
    netsum = netsum + feature2(9) * 1.362473
    netsum = netsum + feature2(10) * 3.452625
    netsum = netsum + feature2(11) * -1.822165
    netsum = netsum + feature2(12) * 1.784463
    netsum = netsum + feature2(13) * -0.7882335
    netsum = netsum + feature2(14) * 2.531423
    OutArray(3) = 1 / (1 + Exp(-netsum))


    OutArray(1) = CInt(30 * (OutArray(1) - 0.1) / 0.8 + 10)
    If (OutArray(1) < 10) Then OutArray(1) = 10
    If (OutArray(1) > 40) Then OutArray(1) = 40

    OutArray(2) = CInt(3 * (OutArray(2) - 0.1) / 0.8 + 1)
    If (OutArray(2) < 1) Then OutArray(2) = 1
    If (OutArray(2) > 4) Then OutArray(2) = 4

    OutArray(3) = CInt(10 * (OutArray(3) - 0.1) / 0.8 + 3)
    If (OutArray(3) < 3) Then OutArray(3) = 3
    If (OutArray(3) > 13) Then OutArray(3) = 13

End Sub
```

APPENDIX F

F.1 Definition of Global Variables

```
Option Explicit
'*****************************************************************************************
'Variables used to call and fire an already trained ANN
'inarray#(), outarray#(): Define the size of the input and output arrays
'inputs%, output%: Take their value from the network that has been called
'netnumber%: Variable to assign a number to the network
'netisopen%: Variable to determine if a network has been opened or not
'defpath%: Variable to define the path to the correspondent NN network

    Public inarray#(), outarray#()
    Public inputs%, output%
    Public netnumber%, netisopen%
    Public defpath%

'*****************************************************************************************
'The following functions are used to open, fire and close each one of the networks
'Declaring functions NS2-32.DLL

    Declare Function OpenNet% Lib "Ns2-32.dll" (ByVal defpath$, netnumber%, inputs%,
output%)
    Declare Function FireNet% Lib "Ns2-32.dll" (netnumber%, inputsarray#, outputsarray#)
    Declare Function CloseNet% Lib "Ns2-32.dll" (netnumber%)

'*****************************************************************************************
'Variables used to manage the training of the ANNs
'TotalComb: Stores number of possible combinations of resources' capacities
'w: Used to detect when to write the simulation outputs and change the level of services
'NeuralNetwork: Keeps track of the network selected by the user
'NNnumber: Takes the value of the network that must be fired by the user's inputs
'decision: Takes its value according with the interaction that a user has with the NN_GDS
'user: Defines which type of user is interacting with the prototype
'flag: Flag to indicate that the training file is ready
'changetype: Takes a value according with the change that the non-expert user selects
'experttask: Takes a value according with the interaction that the expert user selects

    Public TotalComb, w, changetype As Integer
    Public NeuralNetwork As String
    Public decision, user, flag, NNnumber As Integer
    Public experttask, ranges, warning As Integer
    Public unnecessary As Integer
    Public Combo() As Variant

'Constants used to manage the changes requested by the user and to identify the of user

    Public Const ChangeRes As Integer = 1
    Public Const ChangeST As Integer = 2
    Public Const ChangeRT As Integer = 3
    Public Const Expert As Integer = 1
    Public Const Nonexpert As Integer = 2

'*****************************************************************************************
'Variables used to make changes in the simulation model
'i,j,k,l,o,p: Used for tagged elements
'MinRes1, MaxRes1, MinRes2, MaxRes2, MinRes3, MaxRes3:
'Store lower/upper bounds of the ranges for the resources (1:Lathe, 2:RUnld, 3:CGrinder)
'TheModel: Definition of the Arena model name
'ResourceModule: Definition of the element "resources" as an Arena Module
'VariableModule: Definition of the element "variables" as an Arena Module
'ExpressionsModule: Definition of the element "expressions" as an Arena Module
```

```
'Delay1Module, Delay2Module, Delay3Module: Definition of block "delay" as Arena Module
'Rout1Module, Rout2Module, Rout3Module, Rout4Module, Rout5Module, Rout6Module:
'Definition of the block "route" as an Arena Module
'ReplicaModule: Definition of the element "replicate" as an Arena Module
'OUT: Definition of the Siman model name

    Public i, j, k, l, o, p As Integer
    Public MinRes1, MaxRes1, MinRes2, MaxRes2, MinRes3, MaxRes3, MinRes4, MaxRes4 As
Integer
    Public TheModel As Arena.Model
    Public ResourceModule As Arena.Module
    Public VariableModule As Arena.Module
    Public SequencesModule As Arena.Module
    Public ReplicaModule As Arena.Module
    Public OUT As SIMAN
```

F.2 VBA UIR Subroutines

F.2.1 Subroutine GDS_main

```
Sub GDS_main()

'*****************************************************************************************
'Definition of local variables
'MoreInteraction Variable that determines when the loop stops

    Dim StopProcess As Boolean
    Dim MoreInteraction As Integer
    Dim Hora

    flag = 0: MoreInteraction = 1
    unnecessary = 0: decision = 0: experttask = 0

    Do While MoreInteraction = 1
        Load UserType
        UserType.Show

        StopProcess = UserType.CancelButton.Cancel

        Unload UserType

'When the user cancels the program, the replication lenght changes to one (1)

        If StopProcess = True Then
            MoreInteraction = CancelRun
        Else
            If user = Nonexpert Then
                Load UserMenu
                UserMenu.Show

                StopProcess = UserMenu.CancelButton.Cancel

                Unload UserMenu

                If StopProcess = True Then MoreInteraction = CancelRun

'Make Changes
                If decision = 1 Then
                    Call ChangesMain(unnecessary)
                    If unnecessary = 0 Then
                        Call SetScenarios
                        MoreInteraction = 0
                    End If
                    If unnecessary = 1 Then MoreInteraction = CancelRun
                End If

'Use Current Neural Network
                If decision = 2 Then
                    Call NN2Querying(unnecessary)
                    Unload ENTRY
                End If
            Else
                Load Expertform
                Expertform.Show

                StopProcess = Expertform.CancelButton.Cancel

                Unload Expertform
                If StopProcess = True Then MoreInteraction = CancelRun

'Use Current Neural Network
                If experttask = 1 Then
                    Call NN2Querying(unnecessary)
                    Unload ENTRY
                End If

'Training new ANNs
                If experttask = 2 Then
                    Call SetScenarios
                    MoreInteraction = 0
                End If
            End If
        End If
    Loop
End Sub
```

F.2.2 Subroutine CancelRun

```
Public Function CancelRun() As Integer
    Set TheModel = ThisDocument.Model
    i = TheModel.Modules.Find(smFindTag, "Replicas")
    Set ReplicaModule = TheModel.Modules(i)
    ReplicaModule.Data("NumReps") = 1
    ReplicaModule.Data("Length") = 1
    ReplicaModule.UpdateShapes
    CancelRun = 0

End Function
```

F.3 VBA ACR Subroutines

F.3.1 Subroutine ChangesMain

```
Sub ChangesMain(unnecessary As Integer)

'Connecting to the model itself.  The replication time is returned to its normal value.
'When the user cancels the program, the replication lenght changes to one (1).

    Set TheModel = ThisDocument.Model
    Set OUT = TheModel.SIMAN

    i = TheModel.Modules.Find(smFindTag, "Replicas")
    Set ReplicaModule = TheModel.Modules(i)
    ReplicaModule.Data("Length") = 672 'A month in hours
    ReplicaModule.UpdateShapes

'Declaring local variables
    Dim MoreChanges As Integer
    Dim StopProcess As Boolean
    Dim STLUR, STLSR, STLSC, STGUC, STGSC, STOS As String
    Dim TTR, TTC, TTFO As String

    changetype = -1: MoreChanges = 1

Do While MoreChanges = 1
    Load Changesform
    Changesform.Show

    StopProcess = Changesform.CancelButton.Cancel

    Unload Changesform

    If changetype = 0 Then
        MoreChanges = 0: unnecessary = 0
    End If

    If StopProcess = True Then
        MoreChanges = 0: unnecessary = 1
    End If

    If changetype = ChangeRes Then
'To show current capacities on the ResCapacity form as reference for the user.

        i = TheModel.Modules.Find(smFindTag, "recursos")
        j = TheModel.Modules.Find(smFindTag, "variables")
        Set ResourceModule = TheModel.Modules(i)
        Set VariableModule = TheModel.Modules(j)

        Load ResCapacity

        MinRes1 = VariableModule.Data("value(1,4)")
        MaxRes1 = VariableModule.Data("value(1,8)")
        MinRes2 = VariableModule.Data("value(1,5)")
        MaxRes2 = VariableModule.Data("value(1,9)")
        MinRes3 = VariableModule.Data("value(1,6)")
        MaxRes3 = VariableModule.Data("value(1,10)")
        MinRes4 = VariableModule.Data("value(1,7)")
        MaxRes4 = VariableModule.Data("value(1,11)")
        ResCapacity.MinCurCLathe.Caption = MinRes1 + 1
        ResCapacity.MaxCurCLathe.Caption = MaxRes1 - 1
        ResCapacity.MinCurCRUnld.Caption = MinRes2 + 1
        ResCapacity.MaxCurCRUnld.Caption = MaxRes2 - 1
        ResCapacity.MinCurCCGrinder.Caption = MinRes3 + 1
        ResCapacity.MaxCurCCGrinder.Caption = MaxRes3 - 1
        ResCapacity.MinCurCForklift.Caption = MinRes4 + 1
        ResCapacity.MaxCurCForklift.Caption = MaxRes4 - 1
        ranges = 0
        StopProcess = ResCapacity.CancelButton.Cancel

        ResCapacity.Show

        Call CheckingRanges

        If StopProcess = False And ranges = 0 Then

'The values collected from ResCapacity form should be transferred to the variables
'and then to the Arena Model

            MinRes1 = Val(ResCapacity.MinNewCLathe.Text) - 1
            MaxRes1 = Val(ResCapacity.MaxNewCLathe.Text) + 1
            MinRes2 = Val(ResCapacity.MinNewCRUnld.Text) - 1
            MaxRes2 = Val(ResCapacity.MaxNewCRUnld.Text) + 1
            MinRes3 = Val(ResCapacity.MinNewCCGrinder.Text) - 1
            MaxRes3 = Val(ResCapacity.MaxNewCCGrinder.Text) + 1
            MinRes4 = Val(ResCapacity.MinNewCForklift.Text) - 1
            MaxRes4 = Val(ResCapacity.MaxNewCForklift.Text) + 1

'Inserting lower bound of resources capacities in RESOURCES element

            ResourceModule.Data("CapExp(1)") = MinRes3
            ResourceModule.Data("CapExp(2)") = MinRes1
            ResourceModule.Data("CapExp(3)") = MinRes4
            ResourceModule.Data("CapExp(4)") = MinRes2
            ResourceModule.UpdateShapes

'Inserting upper bound of resources capacities in VARIABLES element for future use
(events)

            VariableModule.Data("value(1,8)") = MaxRes1
            VariableModule.Data("value(1,9)") = MaxRes2
            VariableModule.Data("value(1,10)") = MaxRes3
            VariableModule.Data("value(1,11)") = MaxRes4
            VariableModule.Data("value(1,4)") = MinRes1
            VariableModule.Data("value(1,5)") = MinRes2
            VariableModule.Data("value(1,6)") = MinRes3
            VariableModule.Data("value(1,7)") = MinRes4
            VariableModule.UpdateShapes
        End If

        Unload ResCapacity

    End If

    If changetype = ChangeST Then
'To show currente service times into ResServiceT form as reference for the user.

        i = TheModel.Modules.Find(smFindTag, "secuencias")

        Set SequencesModule = TheModel.Modules(i)


        Load ResServiceT

        ResServiceT.CSTLUR.Caption = SequencesModule.Data("value(1,2,1)")
        ResServiceT.CSTLSR.Caption = SequencesModule.Data("value(1,3,1)")
        ResServiceT.CSTLSC.Caption = SequencesModule.Data("value(1,5,2)")
        ResServiceT.CSTGUC.Caption = SequencesModule.Data("value(1,2,2)")
        ResServiceT.CSTGSC.Caption = SequencesModule.Data("value(1,3,2)")
        ResServiceT.CSTOS.Caption = SequencesModule.Data("value(1,2,3)")
        StopProcess = ResServiceT.CancelButton.Cancel
        ResServiceT.Show

        If StopProcess = False Then

'The values collected from ResServiceT form should be transferred to the variables and
'then to the Arena Model

            STLUR = ResServiceT.NSTLUR.Text
            STLSR = ResServiceT.NSTLSR.Text
            STLSC = ResServiceT.NSTLSC.Text
            STGUC = ResServiceT.NSTGUC.Text
            STGSC = ResServiceT.NSTGSC.Text
            STOS = ResServiceT.NSTOS.Text

'Inserting new service times in the SEQUENCES element

            SequencesModule.Data("value(1,2,1)") = STLUR
            SequencesModule.Data("value(1,3,1)") = STLSR
            SequencesModule.Data("value(1,5,2)") = STLSC
            SequencesModule.Data("value(1,2,2)") = STGUC
            SequencesModule.Data("value(1,3,2)") = STGSC
            SequencesModule.Data("value(1,2,3)") = STOS
            SequencesModule.UpdateShapes

        End If
    End If

    If changetype = ChangeRT Then
'To show the current routing times on the RoutTimes form as reference to the user.

        i = TheModel.Modules.Find(smFindTag, "secuencias")

        Set SequencesModule = TheModel.Modules(i)

        Load RoutTimes

        RoutTimes.CTTR.Caption = SequencesModule.Data("value(1,1,1)")
        RoutTimes.CTTC.Caption = SequencesModule.Data("value(1,1,2)")
        RoutTimes.CTTFO.Caption = SequencesModule.Data("value(1,1,3)")
        StopProcess = RoutTimes.CancelButton.Cancel

        RoutTimes.Show

        If StopProcess = False Then

            TTR = RoutTimes.NTTR.Text
            TTC = RoutTimes.NTTC.Text
            TTFO = RoutTimes.NTTFO.Text

 'Inserting new routing times into the ROUTE blocks in the model

            SequencesModule.Data("value(1,1,1)") = TTR
            SequencesModule.Data("value(1,1,2)") = TTC
            SequencesModule.Data("value(1,1,3)") = TTFO
            SequencesModule.UpdateShapes

            Unload RoutTimes

        End If
    End If

Loop
End Sub
```

F.3.2 Subroutine SetScenarios

```
Sub SetScenarios()

'******************************************************************************************
'Once the user has inputted the new capacities and/or service times and/or routing times
'it is necessary to set the stage for re-training process by generating all possible
'combinations of the capacities' values
'Defining local variables
    Dim TotalRep As Integer
    Dim StepSize1, StepSize2, StepSize3, StepSize4 As Integer
    Dim NRep1, NRep2, NRep3, NRep4 As Integer
    Dim indexloop1, indexloop2, indexloop3, indexloop4 As Integer
    Dim IndexLoopI, IndexLoopII, IndexLoopIII, IndexLoopIV As Integer
    Dim IndexLoopA, IndexLoopB, IndexLoopC, IndexLoopD As Integer

    NRep1 = 0: NRep2 = 0: NRep3 = 0: NRep4 = 0

'Connecting to the model to extract the value of the variables

    i = TheModel.Modules.Find(smFindTag, "recursos")
    j = TheModel.Modules.Find(smFindTag, "variables")

    Set ResourceModule = TheModel.Modules(i)
    Set VariableModule = TheModel.Modules(j)

    MinRes1 = Val(VariableModule.Data("value(1,4)"))
    MaxRes1 = Val(VariableModule.Data("value(1,8)"))
    MinRes2 = Val(VariableModule.Data("value(1,5)"))
    MaxRes2 = Val(VariableModule.Data("value(1,9)"))
    MinRes3 = Val(VariableModule.Data("value(1,6)"))
    MaxRes3 = Val(VariableModule.Data("value(1,10)"))
    MinRes4 = Val(VariableModule.Data("value(1,7)"))
    MaxRes4 = Val(VariableModule.Data("value(1,11)"))

    indexloop1 = MinRes1: indexloop2 = MinRes2
    indexloop3 = MinRes3: indexloop4 = MinRes4

'The total number of replications must be determined.
'The Step Size is variable and it depends on the range of the capacities.
'For each resource the total number of possible iterations is calculated.

    If (MaxRes1 - MinRes1) <= 5 Then
        StepSize1 = 1
        NRep1 = (MaxRes1 - MinRes1) + 1
    Else
        StepSize1 = CLng((MaxRes1 - MinRes1) / 4)
        If (MaxRes1 - MinRes1) = 6 Then
            NRep1 = 4
        Else
            If MinRes1 + 4 * StepSize1 >= MaxRes1 Then
                NRep1 = 5
            Else
                NRep1 = 6
            End If
        End If
    End If
    If (MaxRes2 - MinRes2) <= 5 Then
        StepSize2 = 1
        NRep2 = (MaxRes2 - MinRes2) + 1
    Else
        StepSize2 = CLng((MaxRes2 - MinRes2) / 4)
        If (MaxRes2 - MinRes2) = 6 Then
            NRep2 = 4
        Else
            If MinRes2 + 4 * StepSize2 >= MaxRes2 Then
                NRep2 = 5
            Else
                NRep2 = 6
            End If
        End If
    End If
    If (MaxRes3 - MinRes3) <= 4 Then
        StepSize3 = 1
        NRep3 = (MaxRes3 - MinRes3) + 1
    Else
        StepSize3 = CLng((MaxRes3 - MinRes3) / 4)
        If (MaxRes3 - MinRes3) = 6 Then
            NRep3 = 4
        Else
            If MinRes3 + 4 * StepSize3 >= MaxRes3 Then
                NRep3 = 5
            Else
                NRep3 = 6
            End If
        End If
    End If
    If (MaxRes4 - MinRes4) <= 4 Then
        StepSize4 = 1
        NRep4 = (MaxRes4 - MinRes4) + 1
    Else
        StepSize4 = CLng((MaxRes4 - MinRes4) / 4)
        If (MaxRes4 - MinRes4) = 6 Then
            NRep4 = 4
        Else
            If MinRes4 + 4 * StepSize4 >= MaxRes4 Then
                NRep4 = 5
            Else
                NRep4 = 6
            End If
        End If
    End If

'The total number of replications can be calculated

    TotalComb = NRep1 * NRep2 * NRep3 * NRep4
    TotalRep = TotalComb * 15

'The following command will store the total number of combinations in an ARENA variable.

    j = TheModel.Modules.Find(smFindTag, "variables")

    Set VariableModule = TheModel.Modules(j)
    VariableModule.Data("value(1,12)") = TotalComb

'The Replicate element must be found and modified.

    i = TheModel.Modules.Find(smFindTag, "Replicas")

    Set ReplicaModule = TheModel.Modules(i)
    ReplicaModule.Data("NumReps") = TotalRep
    ReplicaModule.UpdateShapes

'Now the combinations must be storaged in a file for later use.
'If the value of CLng((MaxRes - MinRes)/4) is equal to zero, it must be replaced by 1 in
order
'to avoid infinite looping

    Open "C:\ThesisImportant\Combination.txt" For Output As #2

    For indexloop1 = MinRes1 To MaxRes1 Step StepSize1
        For indexloop2 = MinRes2 To MaxRes2 Step StepSize2
            For indexloop3 = MinRes3 To MaxRes3 Step StepSize3
                For indexloop4 = MinRes4 To MaxRes4 Step StepSize4
                    Print #2, indexloop1, indexloop2, indexloop3, indexloop4
                Next indexloop4
            Next indexloop3
        Next indexloop2
    Next indexloop1
    If (indexloop4 - MaxRes4) < StepSize4 Then
        For IndexLoopI = MinRes1 To MaxRes1 Step StepSize1
            For IndexLoopII = MinRes2 To MaxRes2 Step StepSize2
                For IndexLoopIII = MinRes3 To MaxRes3 Step StepSize3
                    Print #2, IndexLoopI, IndexLoopII, IndexLoopIII, MaxRes4
                Next IndexLoopIII
            Next IndexLoopII
        Next IndexLoopI
    End If
    If (indexloop3 - MaxRes3) < StepSize3 Then
        For IndexLoopI = MinRes1 To MaxRes1 Step StepSize1
            For IndexLoopII = MinRes2 To MaxRes2 Step StepSize2
                For IndexLoopIV = MinRes4 To MaxRes4 Step StepSize4
                    Print #2, IndexLoopI, IndexLoopII, MaxRes3, IndexLoopIV
                Next IndexLoopIV
            Next IndexLoopII
        Next IndexLoopI
    End If
    If (indexloop2 - MaxRes2) < StepSize2 Then
        For IndexLoopI = MinRes1 To MaxRes1 Step StepSize1
            For IndexLoopIII = MinRes3 To MaxRes3 Step StepSize3
                For IndexLoopIV = MinRes4 To MaxRes4 Step StepSize4
                    Print #2, IndexLoopI, MaxRes2, IndexLoopIII, IndexLoopIV
                Next IndexLoopIV
            Next IndexLoopIII
        Next IndexLoopI
    End If
    If (indexloop1 - MaxRes1) < StepSize1 Then
        For IndexLoopII = MinRes2 To MaxRes2 Step StepSize2
            For IndexLoopIII = MinRes3 To MaxRes3 Step StepSize3
                For IndexLoopIV = MinRes4 To MaxRes4 Step StepSize4
                    Print #2, MaxRes1, IndexLoopII, IndexLoopIII, IndexLoopIV
                Next IndexLoopIV
            Next IndexLoopIII
        Next IndexLoopII
    End If
    If (indexloop3 - MaxRes3) < StepSize3 And (indexloop2 - MaxRes2) < StepSize2 And
(indexloop1 - MaxRes1) < StepSize1 And (indexloop4 - MaxRes4) < StepSize4 Then
        Print #2, MaxRes1, MaxRes2, MaxRes3, MaxRes4
    End If
    If (indexloop3 - MaxRes3) < StepSize3 And (indexloop2 - MaxRes2) < StepSize2 And
(indexloop4 - MaxRes4) < StepSize4 Then
        For IndexLoopA = MinRes1 To MaxRes1 Step StepSize1
            Print #2, IndexLoopA, MaxRes2, MaxRes3, MaxRes4
        Next IndexLoopA
    End If
    If (indexloop3 - MaxRes3) < StepSize3 And (indexloop1 - MaxRes1) < StepSize1 And
(indexloop4 - MaxRes4) < StepSize4 Then
        For IndexLoopB = MinRes2 To MaxRes2 Step StepSize2
            Print #2, MaxRes1, IndexLoopB, MaxRes3, MaxRes4
        Next IndexLoopB
    End If
    If (indexloop2 - MaxRes2) < StepSize2 And (indexloop1 - MaxRes1) < StepSize1 And
(indexloop4 - MaxRes4) < StepSize4 Then
        For IndexLoopC = MinRes3 To MaxRes3 Step StepSize3
            Print #2, MaxRes1, MaxRes2, IndexLoopC, MaxRes4
        Next IndexLoopC
    End If
    If (indexloop2 - MaxRes2) < StepSize2 And (indexloop1 - MaxRes1) < StepSize1 And
(indexloop3 - MaxRes3) < StepSize3 Then
        For IndexLoopD = MinRes4 To MaxRes4 Step StepSize4
            Print #2, MaxRes1, MaxRes2, MaxRes3, IndexLoopD
        Next IndexLoopD
    End If
    If (indexloop2 - MaxRes2) < StepSize2 And (indexloop1 - MaxRes1) < StepSize1 Then
        For IndexLoopC = MinRes3 To MaxRes3 Step StepSize3
```

```
            For IndexLoopD = MinRes4 To MaxRes4 Step StepSize4
                Print #2, MaxRes1, MaxRes2, IndexLoopC, IndexLoopD
            Next IndexLoopD
        Next IndexLoopC
    End If
    If (indexloop2 - MaxRes2) < StepSize2 And (indexloop3 - MaxRes3) < StepSize3 Then
        For IndexLoopA = MinRes1 To MaxRes1 Step StepSize1
            For IndexLoopD = MinRes4 To MaxRes4 Step StepSize4
                Print #2, IndexLoopA, MaxRes2, MaxRes3, IndexLoopD
            Next IndexLoopD
        Next IndexLoopA
    End If
    If (indexloop4 - MaxRes4) < StepSize4 And (indexloop3 - MaxRes3) < StepSize3 Then
        For IndexLoopA = MinRes1 To MaxRes1 Step StepSize1
            For IndexLoopB = MinRes2 To MaxRes2 Step StepSize2
                Print #2, IndexLoopA, IndexLoopB, MaxRes3, MaxRes4
            Next IndexLoopB
        Next IndexLoopA
    End If
    If (indexloop4 - MaxRes4) < StepSize4 And (indexloop1 - MaxRes1) < StepSize1 Then
        For IndexLoopB = MinRes2 To MaxRes2 Step StepSize2
            For IndexLoopC = MinRes3 To MaxRes3 Step StepSize3
                Print #2, MaxRes1, IndexLoopB, IndexLoopC, MaxRes4
            Next IndexLoopC
        Next IndexLoopB
    End If
    If (indexloop4 - MaxRes4) < StepSize4 And (indexloop2 - MaxRes2) < StepSize2 Then
        For IndexLoopA = MinRes1 To MaxRes1 Step StepSize1
            For IndexLoopC = MinRes3 To MaxRes3 Step StepSize3
                Print #2, IndexLoopA, MaxRes2, IndexLoopC, MaxRes4
            Next IndexLoopC
        Next IndexLoopA
    End If
    If (indexloop1 - MaxRes1) < StepSize1 And (indexloop3 - MaxRes3) < StepSize3 Then
        For IndexLoopB = MinRes2 To MaxRes2 Step StepSize2
            For IndexLoopD = MinRes4 To MaxRes4 Step StepSize4
                Print #2, MaxRes1, IndexLoopB, MaxRes3, IndexLoopD
            Next IndexLoopD
        Next IndexLoopB
    End If
    Close #2

'Initializing the w Record

    j = TheModel.Modules.Find(smFindTag, "variables")
    Set VariableModule = TheModel.Modules(j)
    VariableModule.Data("value(1,13)") = 1

End Sub
```

F.3.3 Subroutine CheckReplications

```
Public Sub CheckReplications()

'*****************************************************************************
'This subroutine is in charge of checking the number of replications and storing
'the statistics of interest when required
'vNRep, vMRep: Store current replication value, and the maximum number of replications
'CapaLathe, CapaRUnld, CapaCGrinder, CapaForklift: Store the level of the resources
'AvgTimeQLathe, AvgTimeQRUnld, AvgTimeQCGrinder, AvgUtilLathe, AvgUtilRUnld,
'AvgUtilCGrinder, AvgUtilForklift: Take the outputs of the simulation model every 15 rep
'LatheC, RUnldC, CGrinderC, ForkliftC: Change the level of the resources every 15 rep
'Tiempo, Hora: Establish stopping time of the total number of replications

    Dim vNRep, vMRep As Integer
    Dim CapaLathe, CapaRUnld, CapaCGrinder, CapaForklift As Double
    Dim AvgTimeQLathe, AvgTimeQRUnld, AvgTimeQCGrinder, AvgTimeQForklift, AvgUtilLathe As
Double
    Dim AvgUtilRUnld, AvgUtilCGrinder, AvgUtilForklift, LatheC, RUnldC, CGrinderC,
ForkliftC As Double
    Dim Tiempo, Hora

'Connecting to the model

    Set TheModel = ThisDocument.Model
    Set OUT = TheModel.SIMAN

    vNRep = OUT.RunCurrentReplication
    vMRep = OUT.RunMaximumReplications

    j = TheModel.Modules.Find(smFindTag, "variables")
    Set VariableModule = TheModel.Modules(j)
    w = VariableModule.Data("value(1,13)")

    If vNRep = 15 * w Then

        If w = 1 Then

'The values of the different possible combinations must be moved from the text file
'created in RunBegin to a VBA array one time.
'First we have to find the size of the array from the Variables Element

            TotalComb = VariableModule.Data("value(1,12)")
            ReDim Combo(1 To TotalComb, 4)

'Second the information is passed from the file to the array

            Open "C:\ThesisImportant\Combination.txt" For Input As #3
            For i = 1 To TotalComb
                Input #3, Combo(i, 1), Combo(i, 2), Combo(i, 3), Combo(i, 4)
            Next i
            Close #3

'The ASCII file where the training data is stored must be opened once for Output
(creation)
'and the columns' labels must be written
            Open "C:\ThesisImportant\TrainingData.txt" For Output As #1
            Print #1, "NoCGrinder", "NoRUnld", "NoLathe", "NoForklift", "AveQLathe",
"UtilLathe", "AveQRUnld", "UtilRUnld", "AveQCGrinder", "UtilCGrinder", "AveQForklift",
"UtilForklift"

'This is to store the statistics on the partial average results.
'First we have to find the modules and take the information about the capacities.

            i = TheModel.Modules.Find(smFindTag, "recursos")
            Set ResourceModule = TheModel.Modules(i)

            CapaLathe = ResourceModule.Data("CapExp(2)")
            CapaRUnld = ResourceModule.Data("CapExp(4)")
            CapaCGrinder = ResourceModule.Data("CapExp(1)")
            CapaForklift = ResourceModule.Data("CapExp(3)")

'Second we have to collect information about the measures of performance

            AvgTimeQLathe = OUT.TallyAverage(5) + OUT.TallyAverage(6)
            AvgTimeQRUnld = OUT.TallyAverage(13)
            AvgTimeQCGrinder = OUT.TallyAverage(7)
            AvgTimeQForklift = OUT.TallyAverage(14)
            AvgUtilLathe = OUT.DStatAverage(2)
            AvgUtilRUnld = OUT.DStatAverage(4)
            AvgUtilCGrinder = OUT.DStatAverage(1)
            AvgUtilForklift = OUT.DStatAverage(3)

'Every 15 records the following values must be stored.
```

```
            Print #1, CapaCGrinder, CapaRUnld, CapaLathe, CapaForklift, AvgTimeQLathe,
AvgUtilLathe, AvgTimeQRUnld, AvgUtilRUnld, AvgTimeQCGrinder, AvgUtilCGrinder,
AvgTimeQForklift, AvgTimeQForklift
            Close #1

'After the values have been stored, the tallies and dstats must be cleared

            OUT.StatisticsClearAll

'The new combination of capacities must be replaced on the Resources Element

            If w <> TotalComb Then

                ResourceModule.Data("CapExp(2)") = Val(Combo(w + 1, 1))
                ResourceModule.Data("CapExp(4)") = Val(Combo(w + 1, 2))
                ResourceModule.Data("CapExp(1)") = Val(Combo(w + 1, 3))
                ResourceModule.Data("CapExp(3)") = Val(Combo(w + 1, 4))
                ResourceModule.UpdateShapes

                w = w + 1

'The value of w must be updated in the VARIABLES Element to use it later

                VariableModule.Data("value(1,13)") = w
            End If

        Else

'After the ASCII file has been created it is opened for Append every 15 replications
            Open "C:\ThesisImportant\TrainingData.txt" For Append As #1

'This is to store the statistics on the partial average results.
'First we have to find the modules and take the information about the capacities.

            i = TheModel.Modules.Find(smFindTag, "recursos")
            Set ResourceModule = TheModel.Modules(i)

            CapaLathe = ResourceModule.Data("CapExp(2)")
            CapaRUnld = ResourceModule.Data("CapExp(4)")
            CapaCGrinder = ResourceModule.Data("CapExp(1)")
            CapaForklift = ResourceModule.Data("CapExp(3)")

'Second we have to collect information about the measures of performance

            AvgTimeQLathe = OUT.TallyAverage(5) + OUT.TallyAverage(6)
            AvgTimeQRUnld = OUT.TallyAverage(13)
            AvgTimeQCGrinder = OUT.TallyAverage(7)
            AvgTimeQForklift = OUT.TallyAverage(14)
            AvgUtilLathe = OUT.DStatAverage(2)
            AvgUtilRUnld = OUT.DStatAverage(4)
            AvgUtilCGrinder = OUT.DStatAverage(1)
            AvgUtilForklift = OUT.DStatAverage(3)

'Every 15 records the following values must be stored.

            Print #1, CapaCGrinder, CapaRUnld, CapaLathe, CapaForklift, AvgTimeQLathe,
AvgUtilLathe, AvgTimeQRUnld, AvgUtilRUnld, AvgTimeQCGrinder, AvgUtilCGrinder,
AvgTimeQForklift, AvgTimeQForklift
            Close #1

'After the values have been stored, the tallies and dstats must be cleared

            OUT.StatisticsClearAll

'The new combination of capacities must be replaced on the Resources Element

            If w <> TotalComb Then
```

```
                ResourceModule.Data("CapExp(2)") = Val(Combo(w + 1, 1))
                ResourceModule.Data("CapExp(4)") = Val(Combo(w + 1, 2))
                ResourceModule.Data("CapExp(1)") = Val(Combo(w + 1, 3))
                ResourceModule.Data("CapExp(3)") = Val(Combo(w + 1, 4))
                ResourceModule.UpdateShapes

                w = w + 1

'The value of w must be updated in the VARIABLES Element to use it later

                VariableModule.Data("value(1,13)") = w
            End If
        End If
    End If

'If it ws the last replication, train ANNs
    If vNRep = vMRep Then
'Feedback for the user
        Tiempo = Now
        MsgBox Tiempo
        MsgBox "Training file has been generated"

        Call NN2Training

'Feedback for the user
        Hora = Now
        MsgBox Hora
        MsgBox "The new networks have been trained and you can use them now"
        flag = 1
    End If

End Sub
```

F.3.4 Subroutine CheckingRanges

```
Sub CheckingRanges()

'Checks if retraining is really needed or not
    If Val(ResCapacity.MinNewCLathe.Text) >= MinRes1 And
Val(ResCapacity.MaxNewCLathe.Text) <= MaxRes1 Then
        If Val(ResCapacity.MinNewCRUnld.Text) >= MinRes2 And
Val(ResCapacity.MaxNewCRUnld.Text) <= MaxRes2 Then
            If Val(ResCapacity.MinNewCCGrinder.Text) >= MinRes3 And
Val(ResCapacity.MaxNewCCGrinder.Text) <= MaxRes3 Then
                If Val(ResCapacity.MinNewCForklift.Text) >= MinRes4 And
Val(ResCapacity.MaxNewCForklift.Text) <= MaxRes4 Then
                    MsgBox "These ranges of level of resources do not require to re-train
the network." & "You can use the current ANN if these are your only changes."
                    ranges = 1
                End If
            End If
        End If
    End If

End Sub
```

F.4 VBA NCR Subroutines

F.4.1 Subroutine TrainingNN2

```
Sub NN2Training()

'******************************************************************************************
'NNOpen: Variable name for the execution of the NS2.exe application
'IMOpen: Variable name for the execution of the Impascii.exe application
'Testset: Variable name for the execution of the Testset.exe application
'Setinput: Variable name for the execution of Netinput.exe application
'Train: Variable name for the execution of the Begtrain.exe application
'Deff: Variable name for the execution of the Dllprime.exe application
'VBcode: Variable name for the execution of the Srcgen.exe application
'Start: Timer used to allow the system to perform the orders given by sendkeys statement
'Contador: String version of "Contando" used to store each ".def" and ".vb" files
'Contando: Integer value that records which network number is being created ("counting")
'indexloop: Used in the loop to count the number of line
'lineas: Integer value that records how many lines are in the VB file ("lines")
'dummy, reference: Variables used to store temporarily the content of the VB file lines

Dim y, w1, w2, w3, w4, w5, w6, w7, w8, Contando, indexloop, lineas As Integer
Dim Start
Dim NNOpen, IMOpen, Testset
Dim Setinput, Train, Deff, VBcode
Dim dummy, reference
Dim contador As String

Contando = 0

'Creating the description file ".dsc"

    NNOpen = Shell("C:\NeuroShell 2\Ns2.exe", vbNormalFocus)
    AppActivate NNOpen
    SendKeys "%FNChanges.dsc%S%FS%FX", True

'Importing the training data that have been allocated in the text file

    IMOpen = Shell("C:\NeuroShell 2\Impascii.exe", vbNormalFocus)
    AppActivate IMOpen
    SendKeys "%FATrainingData.txt%O%FPChanges.pat%S %IB", True

    AppActivate "ASCII File Import"

    y = 3 ' Set duration.
    Start = Timer   ' Set start time.
    Do While Timer < Start + y
        DoEvents    ' Yield to other processes.
    Loop

    SendKeys "{ENTER}", True
    SendKeys "%FX", True

'Extracting testing set

    Testset = Shell("C:\NeuroShell 2\Testset.exe", vbNormalFocus)
    AppActivate Testset
    SendKeys "%FSChanges.pat%O%EB", True

    y = 3 ' Set duration.
    Start = Timer   ' Set start time.
    Do While Timer < Start + y
        DoEvents    ' Yield to other processes.
    Loop

    SendKeys "{ENTER}", True
    SendKeys "%FX", True

'Setting inputs and outputs

For w1 = 0 To 1
    For w2 = 0 To 1
        For w3 = 0 To 1
            For w4 = 0 To 1
                For w5 = 0 To 1
                    For w6 = 0 To 1
                        For w7 = 0 To 1
                            For w8 = 0 To 1
                                If w1 <> 0 Or w2 <> 0 Or w3 <> 0 Or w4 <> 0 Or w5 <> 0 Or
w6 <> 0 Or w7 <> 0 Or w8 <> 0 Then
                                    Setinput = Shell("C:\NeuroShell 2\Netinput.exe",
vbNormalFocus)
                                    AppActivate Setinput
                                    SendKeys "%FPChanges.pat%O{RIGHT}{ENTER}", True
                                    SendKeys "{TAB}A{RIGHT}A{RIGHT}A{RIGHT}", True
                                    If w1 = 1 Then
                                        SendKeys "I{RIGHT}"
                                    Else
                                        SendKeys "{BACKSPACE}{RIGHT}"
                                    End If
                                    If w2 = 1 Then
                                        SendKeys "I{RIGHT}"
                                    Else
                                        SendKeys "{BACKSPACE}{RIGHT}"
                                    End If
                                    If w3 = 1 Then
                                        SendKeys "I{RIGHT}"
                                    Else
                                        SendKeys "{BACKSPACE}{RIGHT}"
                                    End If
                                    If w4 = 1 Then
                                        SendKeys "I{RIGHT}"
                                    Else
                                        SendKeys "{BACKSPACE}{RIGHT}"
                                    End If
                                    If w5 = 1 Then
                                        SendKeys "I{RIGHT}"
                                    Else
                                        SendKeys "{BACKSPACE}{RIGHT}"
                                    End If
                                    If w6 = 1 Then
                                        SendKeys "I"
                                    Else
                                        SendKeys "{BACKSPACE}"
                                    End If
                                    If w7 = 1 Then
                                        SendKeys "I"
                                    Else
                                        SendKeys "{BACKSPACE}"
                                    End If
                                    If w8 = 1 Then
                                        SendKeys "I"
                                    Else
                                        SendKeys "{BACKSPACE}"
                                    End If
                                    SendKeys "%SC", True

                                    y = 3 ' Set duration.
                                    Start = Timer   ' Set start time.
                                    Do While Timer < Start + y
                                        DoEvents    ' Yield to other processes.
                                    Loop
```

```
                                        SendKeys "%FX", True

        'Training a simple NN

                                        Train = Shell("C:\NeuroShell 2\Begtrain.exe",
vbNormalFocus)
                                        AppActivate Train
                                        SendKeys "%FSChanges.pat%O{RIGHT}{TAB
5}{ENTER}%TS{RIGHT}{ENTER}", True

                                        y = 45  'Set duration to 45 seconds.
                                        Start = Timer   ' Set start time.
                                        Do While Timer < Start + y
                                            DoEvents    ' Yield to other processes.
                                        Loop

                                        SendKeys "%TI", True

                                        y = 3 ' Set duration.
                                        Start = Timer   ' Set start time.
                                        Do While Timer < Start + y
                                            DoEvents    ' Yield to other processes.
                                        Loop

                                        SendKeys "{ENTER}", True
                                        SendKeys "%FE", True

                                        Contando = Contando + 1

                                        contador = CStr(Contando)

        'Generating .def file

                                        Deff = Shell("C:\NeuroShell 2\Dllprime.exe",

vbNormalFocus)
                                        AppActivate Deff
                                        SendKeys "%FCChanges.fig%O%FDChanges", True
                                        SendKeys contador, True
                                        SendKeys ".def", True
                                        SendKeys "%S%GG", True

                                        y = 15 ' Set duration.
                                        Start = Timer   ' Set start time.
                                        Do While Timer < Start + y
                                            DoEvents    ' Yield to other processes.
                                        Loop

                                        SendKeys "{ENTER}", True
                                        SendKeys "%FX", True

        'Generating VB code

                                        VBcode = Shell("C:\NeuroShell 2\Srcgen.exe",

vbNormalFocus)
                                        AppActivate VBcode
                                        SendKeys "{DOWN}%FCChanges.fig%O%FSChanges1", True
                                        SendKeys contador, True
                                        SendKeys ".vb", True
                                        SendKeys "%S%GG", True

                                        y = 15 ' Set duration.
                                        Start = Timer   ' Set start time.
                                        Do While Timer < Start + y
                                            DoEvents    ' Yield to other processes.
                                        Loop

                                        SendKeys "{ENTER}", True
                                        SendKeys "%FX", True
        'Cleaning the VB code to remove innecessary information
                                    Open "C:\ThesisImportant\Changes1" + contador + ".vb"
For Input As #100
                                        lineas = 0
                                        Do While Not EOF(100)
                                            Line Input #100, reference
                                            lineas = lineas + 1
                                        Loop
                                    Close #100
                                    Open "C:\ThesisImportant\Changes1" + contador + ".vb"
For Input As #101
                                    Open "C:\ThesisImportant\Changes" + contador + ".vb"
For Output As #102
                                    Open "C:\ThesisImportant\Dummy.vb" For Output As #103
                                        For indexloop = 1 To lineas
                                            If indexloop < 7 Then
                                                Line Input #101, dummy
                                                Print #103, dummy
                                            End If
                                            If (indexloop >= 7) And (indexloop <= (lineas
- 17)) Then
                                                Line Input #101, dummy
                                                Print #102, dummy
                                            End If
                                            If (indexloop > (lineas - 17)) And (indexloop
< (lineas - 13)) Then
                                                Line Input #101, dummy
                                                Print #103, dummy
                                            End If
                                            If (indexloop = (lineas - 13)) Then
                                                Line Input #101, dummy
                                                Print #102, dummy
                                            End If
                                            If (indexloop > (lineas - 13)) And (indexloop
< (lineas - 9)) Then
                                                Line Input #101, dummy
                                                Print #103, dummy
                                            End If
                                            If (indexloop = (lineas - 9)) Then
                                                Line Input #101, dummy
                                                Print #102, dummy
                                            End If
                                            If (indexloop > (lineas - 9)) And (indexloop
< (lineas - 5)) Then
                                                Line Input #101, dummy
                                                Print #103, dummy
                                            End If
                                            If (indexloop = (lineas - 5)) Then
                                                Line Input #101, dummy
                                                Print #102, dummy
                                            End If
                                            If (indexloop > (lineas - 5)) Then
                                                Line Input #101, dummy
                                                Print #103, dummy
                                            End If
                                        Next indexloop
                                    Close #101: Close #102: Close #103
                                    Kill "C:\ThesisImportant\Changes1" + contador + ".vb"
                                    Kill "C:\ThesisImportant\Dummy.vb"
                                End If
                            Next w8
                        Next w7
                    Next w6
                Next w5
            Next w4
        Next w3
    Next w2
Next w1

End Sub
```

F.4.2 Subroutine SelectingNetworks

```
Sub SelectingNetwork()

'*************************************************************************************
'Declaring local variables
'Networkarray Array that stored the different selection of inputs that could occur
'mycounter Variable used to set the position of any array in the Networkarray
'Referencia Used to create the Networkarray (Referencia, spanish work for Reference)
    Dim y1, y2, y3, y4, y5, y6, y7, y8 As Integer
    Dim Referencia As String
    Dim Networkarray(1 To 255)
    Dim mycounter, z As Integer

'Variables used to generated all possible combinations of inputs and to detec
'which inputs were selected by the user

mycounter = 1
For y1 = 0 To 1
    For y2 = 0 To 1
        For y3 = 0 To 1
            For y4 = 0 To 1
                For y5 = 0 To 1
                    For y6 = 0 To 1
                        For y7 = 0 To 1
                            For y8 = 0 To 1
                                If y1 <> 0 Or y2 <> 0 Or y3 <> 0 Or y4 <> 0 Or y5 <> 0 Or
y6 <> 0 Then
                                    Referencia = CStr(y1) + CStr(y2) + CStr(y3) +
CStr(y4) + CStr(y5) + CStr(y6)
                                    If NeuralNetwork = Referencia Then NNnumber =
mycounter
                                    mycounter = mycounter + 1
                                End If
                            Next y8
                        Next y7
                    Next y6
                Next y5
            Next y4
        Next y3
    Next y2
Next y1

End Sub
```

F.4.3 Subroutine NN2Using

```
Sub NN2Using(unnecessary As Integer)

'*************************************************************************************
'Declaring local variables
'Flags used to detect the value of the inputs selected by the user and to store them
'in the inarray
    Dim y%
    Dim loopIndex As Integer
    Dim Flags(1 To 8)
    Dim out1, out2, out3, out4 As Integer
    Dim indexloop, NumberInputs As Integer
```

```
'Connecting to NeuroShell and opening the requested Neural Network

    If Not netisopen Then
        y = OpenNet("C:\ThesisImportant\Changes" + CStr(NNnumber) + ".def", netnumber,
inputs, output)
        If y > 0 Then
            MsgBox "Error returned from OpenNet: " + CStr(y) + ".", 16, "Error"
        End If
    End If

'Retrieving values from ENTRY form as Inputs and loading these values in the OUTS form
ReDim inarray#(inputs)
ReDim outarray#(output)

Load OUTS

'The labels are not active unless the specific input is selected to fire the network
OUTS.TLLabel.Enabled = False
OUTS.ULLabel.Enabled = False
OUTS.TRULabel.Enabled = False
OUTS.URULabel.Enabled = False
OUTS.TCGLabel.Enabled = False
OUTS.UCGLabel.Enabled = False
OUTS.TFOUTS.Enabled = False
OUTS.UFOUTS.Enabled = False

For loopIndex = 1 To inputs
    If ENTRY.QLathe.Text <> "" Then
        inarray(loopIndex) = Val(ENTRY.QLathe.Text)
        OUTS.TLLabel.Enabled = True
        OUTS.TLOUTS.Caption = Val(ENTRY.QLathe.Text)
        ENTRY.QLathe.Text = ""
    Else
        If ENTRY.ULathe.Text <> "" Then
            inarray(loopIndex) = Val(ENTRY.ULathe.Text)
            OUTS.ULLabel.Enabled = True
            OUTS.ULOUTS.Caption = Val(ENTRY.ULathe.Text)
            ENTRY.ULathe.Text = ""
        Else
            If ENTRY.QRUnld.Text <> "" Then
                inarray(loopIndex) = Val(ENTRY.QRUnld.Text)
                OUTS.TRULabel.Enabled = True
                OUTS.TRUOUTS.Caption = Val(ENTRY.QRUnld.Text)
                ENTRY.QRUnld.Text = ""
            Else
                If ENTRY.URUnld.Text <> "" Then
                    inarray(loopIndex) = Val(ENTRY.URUnld.Text)
                    OUTS.URULabel.Enabled = True
                    OUTS.URUOUTS.Caption = Val(ENTRY.URUnld.Text)
                    ENTRY.URUnld.Text = ""
                Else
                    If ENTRY.QCGrinder.Text <> "" Then
                        inarray(loopIndex) = Val(ENTRY.QCGrinder.Text)
                        OUTS.TCGLabel.Enabled = True
                        OUTS.TCGOUTS.Caption = Val(ENTRY.QCGrinder.Text)
                        ENTRY.QCGrinder.Text = ""
                    Else
                        If ENTRY.UCGrinder.Text <> "" Then
                            inarray(loopIndex) = Val(ENTRY.UCGrinder.Text)
                            OUTS.UCGLabel.Enabled = True
                            OUTS.UCGOUTS.Caption = Val(ENTRY.UCGrinder.Text)
                            ENTRY.UCGrinder.Text = ""
                            If ENTRY.QForklift.Text <> "" Then
                                inarray(loopIndex) = Val(ENTRY.QForklift.Text)
                                OUTS.TFLabel.Enabled = True
                                OUTS.TFOUTS.Caption = Val(ENTRY.QForklift.Text)
                                ENTRY.QForklift.Text = ""
                            Else
```

```
                                If ENTRY.UForklift.Text <> "" Then
                                    inarray(loopIndex) = Val(ENTRY.UForklift.Text)
                                    OUTS.UFLabel.Enabled = True
                                    OUTS.UFOUTS.Caption = Val(ENTRY.UForklift.Text)
                                    ENTRY.UForklift.Text = ""
                                End If
                            End If
                        End If
                    End If
                End If
            End If
        End If
    End If
Next

'Actual firing of the NN

    y = FireNet(netnumber, inarray(1), outarray(1))

    If y > 0 Then
        MsgBox "Error returned from FireNet:" + CStr(y) + ".", 16, "Error"
    End If

'Loading the results in the OUTS form

    If unnecessary = 0 Then
        If outarray(3) <= MinRes1 + 1 Then
            OUTS.Lathe.Caption = MinRes1 + 1
        Else
            If outarray(3) > MaxRes1 - 1 Then
                OUTS.Lathe.Caption = "N/A"
                OUTS.InfeasibleLathe.Caption = "Your current upper bound of " + (MaxRes1
- 1) + " in number of lathes does not allow to achieve your goal"
            Else
                OUTS.Lathe.Caption = Format$(outarray(3), "00")
                OUTS.InfeasibleLathe.Caption = "N/A"
            End If
        End If
        If outarray(2) <= MinRes2 + 1 Then
            OUTS.RUnld.Caption = MinRes2 + 1
        Else
            If outarray(2) > MaxRes2 - 1 Then
                OUTS.RUnld.Caption = "N/A"
                OUTS.InfeasibleRUnld.Caption = "Your current upper bound of " + (MaxRes2
- 1) + " in number of ring unloaders does not allow to achieve your goal"
            Else
                OUTS.RUnld.Caption = Format$(outarray(2), "00")
                OUTS.InfeasibleRUnld.Caption = "N/A"
            End If
        End If
        If outarray(1) <= MinRes3 + 1 Then
            OUTS.CGrinder.Caption = MinRes3 + 1
        Else
            If outarray(1) > MaxRes3 - 1 Then
                OUTS.CGrinder.Caption = "N/A"
                OUTS.InfeasibleCGrinder.Caption = "Your current upper bound of " +
(MaxRes3 - 1) + " in number of cover grinders does not allow to achieve your goal"
            Else
                OUTS.CGrinder.Caption = Format$(outarray(1), "00")
                OUTS.InfeasibleCGrinder.Caption = "N/A"
            End If
        End If
        If outarray(4) <= MinRes4 + 1 Then
            OUTS.Forklifts.Caption = MinRes3 + 1
        Else
            If outarray(4) > MaxRes4 - 1 Then
                OUTS.Forklifts.Caption = "N/A"
                OUTS.InfeasibleForklift.Caption = "Your current upper bound of " +
(MaxRes3 - 1) + " in number of forklifts does not allow to achieve your goal"
```

```
            Else
                OUTS.Forklifts.Caption = Format$(outarray(4), "00")
                OUTS.InfeasibleForklift.Caption = "N/A"
            End If
        End If
    Else
        If outarray(3) <= MinRes1 + 1 Then
            OUTS.Lathe.Caption = MinRes1 + 1
        Else
            If outarray(3) > MaxRes1 - 1 Then
                OUTS.Lathe.Caption = Format$(outarray(3), "00")
                OUTS.InfeasibleLathe.Caption = "Your current upper bound of " + (MaxRes1
- 1) + " in number of lathes does not allow to achieve your goal.  Suggested value is
displayed for your reference."
            Else
                OUTS.Lathe.Caption = Format$(outarray(3), "00")
                OUTS.InfeasibleLathe.Caption = "N/A"
            End If
        End If
        If outarray(2) <= MinRes2 + 1 Then
            OUTS.RUnld.Caption = MinRes2 + 1
        Else
            If outarray(2) > MaxRes2 - 1 Then
                OUTS.RUnld.Caption = Format$(outarray(2), "00")
                OUTS.InfeasibleRUnld.Caption = "Your current upper bound of " + (MaxRes2
- 1) + " in number of RUnld does not allow to achieve your goal.  Suggested value is
displayed for your reference."
            Else
                OUTS.RUnld.Caption = Format$(outarray(2), "00")
                OUTS.InfeasibleRUnld.Caption = "N/A"
            End If
        End If
        If outarray(1) <= MinRes3 + 1 Then
            OUTS.CGrinder.Caption = MinRes3 + 1
        Else
            If outarray(1) > MaxRes3 - 1 Then
                OUTS.CGrinder.Caption = Format$(outarray(1), "00")
                OUTS.InfeasibleCGrinder.Caption = "Your current upper bound of " +
(MaxRes3 - 1) + " in number of CGrinder does not allow to achieve your goal.  Suggested
value is displayed for your reference."
            Else
                OUTS.CGrinder.Caption = Format$(outarray(1), "00")
                OUTS.InfeasibleCGrinder.Caption = "N/A"
            End If
        End If
         If outarray(4) <= MinRes4 + 1 Then
            OUTS.Forklifts.Caption = MinRes4 + 1
        Else
            If outarray(4) > MaxRes4 - 1 Then
                OUTS.Forklifts.Caption = Format$(outarray(4), "00")
                OUTS.InfeasibleForklift.Caption = "Your current upper bound of " +
(MaxRes4 - 1) + " in number of forklifts does not allow to achieve your goal.  Suggested
value is displayed for your reference."
            Else
                OUTS.Forklifts.Caption = Format$(outarray(4), "00")
                OUTS.InfeasibleForklift.Caption = "N/A"
            End If
        End If
    End If

'Closing the NN after it has been used

    y = CloseNet(netnumber)

End Sub
```

F.4.4 Subroutine RunNewANN

```
Sub RunNewANN()
'Declaration of local variables
    Dim StopProcess As Boolean
    Dim MoreRuns, unnecessary As Integer
    unnecessary = 0
    MoreRuns = 1

    Do While MoreRuns = 1
        Load Aftertraining
        Aftertraining.Show

        StopProcess = Aftertraining.CancelButton.Cancel

        Unload Aftertraining

        If StopProcess Then
            MsgBox "Thanks for using this program"
            MoreRuns = 0
        Else
            Call NN2Querying(unnecessary)
        End If
    Loop

End Sub
```

F.4.5 Subroutine NN2Querying

```
Sub NN2Querying(unnecessary As Integer)
'Declaring local variables
    Dim StopProcess As Boolean
'This routine takes all the actions related with querying the already trained networks
    Set TheModel = ThisDocument.Model
    Set OUT = TheModel.SIMAN

    j = TheModel.Modules.Find(smFindTag, "variables")
    Set VariableModule = TheModel.Modules(j)

    Load ENTRY

    ENTRY.MinCurCLathe.Caption = VariableModule.Data("value(1,4)") + 1
    ENTRY.MaxCurCLathe.Caption = VariableModule.Data("value(1,8)") - 1
    ENTRY.MinCurCRUnld.Caption = VariableModule.Data("value(1,5)") + 1
    ENTRY.MaxCurCRUnld.Caption = VariableModule.Data("value(1,9)") - 1
    ENTRY.MinCurCCGrinder.Caption = VariableModule.Data("value(1,6)") + 1
    ENTRY.MaxCurCCGrinder.Caption = VariableModule.Data("value(1,10)") - 1
    ENTRY.MinCurCForklift.Caption = VariableModule.Data("value(1,7)") + 1
    ENTRY.MaxCurCForklift.Caption = VariableModule.Data("value(1,11)") - 1

    ENTRY.Show

    StopProcess = ENTRY.CancelButton.Cancel

'ENTRY form remains loaded but hiddent for th use of the following routines
    If Not StopProcess Then
        Call SelectingNetwork
        Call NN2Using(unnecessary)

        OUTS.Show
        Unload OUTS
    End If
End Sub
```

F.5 VBA Support Subroutines

```
Sub Letter(tecla As MSForms.ReturnInteger)

'Checks to make sure numeric values are entered
'tecla = Spanish work for "key", checks ASCII values of keys pressed

    If tecla < 46 Or tecla > 57 Then
        MsgBox "Enter a numeric value."
        tecla = 0
    End If

End Sub


Sub LetterInt(tecla As MSForms.ReturnInteger)

'Checks to make sure integer values are entered

    If tecla < 48 Or tecla > 57 Then
        MsgBox "Enter an integer value."
        tecla = 0
    End If

End Sub
```

APPENDIX G

G.1 SPSS

τ_r = Time per replication

τ_I = Time for training part I

τ_{II} = Time for training part II

η = Number of replications

Table 14: 95% CI Training Time for NN-Based GDS

Variable	τ_r	τ_I	τ_{II}	η
Statistic				
Mean	0.3108	487.9000	228.4286	1608
Mean Std. Deviation	0.02326	29.9015	25.84096	83.8987
Lower Bound (95%CI)	0.2582	420.2581	165.1980	1418.2078
Upper Bound (95%CI)	0.3634	555.5419	291.6591	1797.7922
5% Trimmed Mean	0.3107	485.3333	224.6984	1618.3333
Median	0.3123	454.5000	191.0000	1500.0000
Variance	0.00541	8940.989	4674.286	70390.000
Std. Deviation (STD)	0.07355	94.5568	68.36875	265.3111
Minimum	0.19	380.00	169.00	1080.00
Maximum	0.43	642.00	355.00	1950.00
Range	0.23	262.00	186.00	870.00
Interquartile Range	0.1389	165.2500	106.0000	375.0000
Skewness	-0.017	0.888	1.319	-0.508
Skewness STD	0.687	0.687	0.794	0.687
Kurtosis	-0.834	-0.711	0.791	0.224
Kurtosis STD	1.334	1.334	1.587	1.334