

# An Introduction to the Semantic- Neural Method

by Andrew J. Ryan

DR. BROUSE SYST 512

<b>1.0 OVERVIEW</b> .....	<b>4</b>
<b>2.0 BACKGROUND</b> .....	<b>4</b>
<b>3.0 METHOD WARS</b> .....	<b>6</b>
3.1 COAD'S OBJECT ORIENTED ANALYSIS .....	6
3.2 JACKSON'S SYSTEM DEVELOPMENT .....	7
3.3 CHEN'S ENTITY-RELATIONSHIP MODEL .....	7
3.4 ORR'S STRUCTURED REQUIREMENTS DEFINITION .....	8
3.5 ROSS' STRUCTURED ANALYSIS AND DESIGN TECHNIQUE .....	8
3.6 TEICHROEW'S PSL/PSA <sup>TM</sup> .....	9
<b>4.0 A UML PRIMER</b> .....	<b>9</b>
4.1 BACKGROUND .....	9
4.2 DEVELOPMENT EFFORT .....	10
4.3 UML TODAY .....	12
<b>5.0 INSIDE THE UNIFIED PROCESS</b> .....	<b>12</b>
<b>6.0 INTRODUCTION TO SEMANTIC NETWORKS</b> .....	<b>13</b>
<b>7.0 NEURAL NETWORKS</b> .....	<b>15</b>
7.1 INTRODUCTION TO NEURAL NETWORKS .....	15
7.2 USAGE IN SEMANTIC-NEURAL METHOD .....	16
7.3 FEASIBILITY .....	17
<b>8.0 GOAL ORIENTED REQUIREMENTS ANALYSIS</b> .....	<b>17</b>
8.1 SOFTGOALS DEFINED.....	18
8.2 FUNCTIONAL REQUIREMENTS ANALYSIS.....	20
<b>9.0 THE SEMANTIC-NEURAL METHOD</b> .....	<b>22</b>
9.1 SEMANTIC-NEURAL METHOD IN THEORY .....	22
9.2 THE SEMANTIC-NEURAL METHOD – A PEDAGOGICAL APPROACH .....	23
9.2.1 <i>Setting the boundary for the system</i> .....	23
9.2.2 <i>Identify important concepts</i> .....	24
9.2.3 <i>Create, define and elaborate nodes</i> .....	24
9.2.4 <i>Construct links and link concepts</i> .....	24
9.2.5 <i>Create and Evaluate the semantic network</i> .....	24
9.2.6 <i>Develop the Neural Network</i> .....	25
9.2.7 <i>Train the Neural Network</i> .....	26
9.3 REQUIREMENTS ANALYSIS .....	26
9.4 THE MATURATION PROCESS.....	27
<b>10.0 THE SEMANTIC-NEURAL METHOD IN PRACTICE</b> .....	<b>27</b>
10.1 DEVELOPING THE NETWORKS: SEMANTIC AND NEURAL .....	27
<b>11.0 BENEFITS OF THE SEMANTIC-NEURAL METHOD</b> .....	<b>29</b>
<b>12.0 CONCLUSIONS</b> .....	<b>31</b>
<b>13.0 BIBLIOGRAPHY</b> .....	<b>32</b>
<b>APPENDIX A – MODELS OF THE UNIFIED PROCESS</b> .....	<b>36</b>
<b>APPENDIX B – SEMANTIC MODEL OF WIND CHILL SYSTEM</b> .....	<b>37</b>



## 1.0 Overview

The field of systems engineering is nearly fifty years old (according to Steven Brill) but is amazingly, relatively untouched by the world of automation. Software engineers now have code generators which can produce code at rates no human programmer could duplicate. There are also CAD tools that engineers use to decrease their computation time and generate alternate designs. On the other hand, most systems engineering processes are still done by the “seat of the pants” approach. This paper will introduce a method to automate some of the key systems engineering concepts. The Semantic-Neural Method (SNM) attempts to use semantic knowledge trees in combination with neural networks to streamline both the requirements and testing phases. Moreover, it provides a uniform manner for developing evolutionary and repeated systems. Once a network is trained, it can identify and evaluate how the new changes (i.e. an upgrade, new version) affect functionality and pre-existing requirements. This paper will describe this process, dubbed the Semantic-Neural Method, and introduce the various elements that it is composed of.

## 2.0 Background

Systems engineering, art or science? This question has plagued this field for many years with no definitive answer in sight. On one hand, there are those who say it is art, as there is room for one to use innovation when approaching a problem. Conversely, there are those who say it is science, as there are many methodologies to choose from when deciding how to tackle a problem. Although systems engineering as a science has been the view of this author it has often been difficult to prove. However with the emergence of the Unified Modeling Language and the Unified Process, an industry standard has dawned – at least when it comes to modeling systems. Using Object Oriented Design and Analysis

as its base, UML has created a model which encompasses the best features of all of the currently available methods and put it into one package.

There lies only one problem; UML does not account for non-functional requirements. The often over-looked non-functional requirement has recently received a lot of attention especially from the University of Toronto. It is here that in 1993, Dr. Kenneth Chung and his colleagues first introduced the idea for an object-based method to map non-functional requirements. Once the systems engineering community got wind of this new concept, there was almost exponential growth in the number of object-based methods for non-functional requirements.

Similar to the unification process done for UML (see section 4), there is now an effort going on to unify these “non-functional” method wars.” In a paper published by the Association for Computing Machinery (ACM) Dr. Chung teamed up with Dr. Eric Yu and Dr. John Mylopoulos in writing a piece entitled *From Object Oriented Requirements Analysis to Goal Oriented Requirements Analysis*. This work will be talked about in great detail later on this paper, however it has formed the basis for a new and exciting notation and approach to requirements analysis.

Both of the previously mentioned methods strive for a universal notation and approach for requirements analysis. Taking those two features in tandem, it makes for an excellent candidate for a computer-automated process. Using semantic networks, neural networks (or possibly agents) and the common notations described above, a process can be developed which can make a system self-sufficient . . . well almost. Depending on the complexity desired in developing the neural network, the network has the ability to forecast events. Meaning, trained properly, the network can predict how the system will behave under changing conditions, whether it be time, a change in reliability (percentage-wise etc. The power of the neural net is left to the developers. However, even a simple net provides one extremely important quality – the ability to take the current system, modify requirements, and generate models and perform requirements analysis with great speed.

With the total development cycle of large systems being upwards of ten years (from initial concept to field readiness), a neural network can grow with the system and alert developers to potential system flaws, algorithm maximization ideas, or even warn of maintenance issues. Coupled with the functionality described above, SNM can really be an asset to companies who develop evolutionary systems or develop one system which is eventually modified to suit the customer (i.e. a credit card processing system). The remainder of this paper will be dedicated to describing this theory in detail.

### **3.0 Method Wars**

As stated before, today there are dozens of like-minded techniques that support development from requirements analysis to implementation. The following section will provide a brief timeline of a few of the more prominent techniques.

#### **3.1 Coad's Object Oriented Analysis**

Taken from Alan Davis' book *Software Requirements: Objects, States, Functions*, Peter Coad's technique is called: Object-Oriented Analysis (OOA) and consists of five activities: specifying objects, attributes, structures, services, and subjects (Davis 61). In specifying objects, Coad advises that one look toward the real world and look for similar roles that are being acted out, and map those to objects required for the system. With the choice of objects in place, attributes and structures for the object are then selected. Attributes express some important aspect of the object. For example, if the object were a pager, some attributes would be: display message, change time, erase message. In terms of structures, they come in two types: gen-specific and whole part. These help describe the cardinality between objects. For example, a pager has one number is an example of a whole part structure. Whereas a general specific structure allows you to define one class that captures common attributes and allow objects that are members of that class to inherit those attributes and services. Sticking

with the pager example, pager would be the superclass, and sub-classes, numeric, alphanumeric, and voice would inherit the general properties of pager and then have their own attributes to suit their individual functionality.

The object model is the essential part of Coad's model. By listing all objects using the prescribed methods above, a developer would have a complete model layout of the system.

### ***3.2 Jackson's System Development***

In 1983, Michael Jackson came out with an entity based software development methodology called the Jackson System Development (JSD). Fundamental to JSD is the construction of a model that mirrors real-world entities. Meaning, all entities, their relationships, and actions performed/received by the entities. Jackson defines an entity as: "[something] that must exist in the real world, outside of the system. Once the entity has been defined, they are given well-defined roles that allow them to respond predictably to any stimuli. When all responses have been modeled, the system specification diagram will show a "cradle to grave" view of the entity. These diagrams prove effective as they provide a snapshot of all of the functionality encompassed in the system.

### ***3.3 Chen's Entity-Relationship Model***

Peter Chen's Entity Relationship Model (1976) was designed to show relationships in large databases. Since then it has gained popularity and is now being used to model requirements in systems. The basic premise behind ER diagrams are that they model the logical structure of data \*1,69. Using cardinality and relationships, ER diagrams pictorially give an overview of a system.

### **3.4 Orr's Structured Requirements Definition**

Orr's method, introduced in 1981, calls for defining the application context. It is a multi-step method which calls for a developer to:

- Step 1. Define a user-level data flow diagram
- Step 2. Define a combined user-level data flow diagram
- Step 3. Define the application level data flow diagram
- Step 4. Define the application level functions.

Taken together Orr believes that the inputs and outputs for the final system can be determined once the above definitions are acquired by the developers.

### **3.5 Ross' Structured Analysis and Design Technique**

Developed by Doug Ross in the early 1970's, the Structured Analysis and Design Technique (SADT) attempts to express a problem in terms of natural language using a non-ambiguous graphical notation in which natural language is embedded. The model is composed of a hierarchy of diagrams, with a context diagram (most general) is drawn first and is then refined into smaller diagrams with a more specific functionality.

\*\* In 1979 Tom DeMarco introduced the Structured Analysis and System Specification (SASS). Many of its features are carry-overs from SADT and for that reason it will not be spoken about in great detail. His contribution lies in the procedure that diagrams are laid out in. DeMarco advocates a more piecemeal approach, where Ross backed an iterative approach

### **3.6 Teichroew's PSL/PSA<sup>TM</sup>**

In the early 1970's, Dan Teichroew at the University of Michigan developed The Problem Statement Language/Problem Statement Analyzer as a completely text language to describe processes, information flow between processes, and hierarchical decomposition of process and data. 1,999 PSL statements (which must be syntactically correct) can be input into PSA which in turn stores all the statements into a database. Once stored, commands can be given to the PSA to generate a wide array of reports such as data dictionaries, data flow diagrams and analysis reports to name a few.

## **4.0 A UML Primer**

The first item that must be addressed when speaking of UML is that it is only a notation. UML is not the savior for those looking for a common way to model systems. Section 5 will deal with the Unified Process which deals with the modeling process itself. However the power of UML is its flexibility and ability to be modified to fit almost any size project. As Doug Rosenberg of Software Development Online notes: "The UML notation is big (maybe too big) and is flexible enough to accommodate the needs of a very wide range of products (Rosenberg 1). He continues, "One of the most important things to remember when learning UML is that you don't need to use every construct just because it's there. With the large base UML offers, it can be applied to any domain and yield successful results. This section will briefly outline UML from its origins to present.

### **4.1 Background**

Identifiable object-oriented modeling languages began to appear between the mid-1970s and the late 1980s (see section two for a detailed look at OO methods) as various methodologies experimented with different approaches to

object oriented analysis and design. During this time period, the number of distinguishable modeling languages increased from less than ten to more than fifty. Some may argue the more options a person/organization has, the better off they are. However this is not the case. Developers had trouble finding complete satisfaction in any one method and thus the “method wars” began. By the mid-1990s, hybrid methods started sprouting up which incorporated the older techniques, and a few clearly prominent ones emerged.

#### **4.2 Development Effort**

One of the best efforts to date in terms of integrating methodologies has been the Unified Modeling Language (UML) and its subsequent modeling process the Unified Process. The development of UML began in late 1994, when Grady Booch and Jim Rumbaugh of Rational Software Corporation began their work on unifying the Booch and Object Modeling Technique (OMT) methods. Soon after, Ivar Jacobson (author of the Object-Oriented Software Engineering [OOSE] method) joined Rational and together they set forth to create a modeling language for the following three reasons:

1. The existing methods were already evolving toward each other independently.
2. By unifying semantics and notation, they could bring some stability to the object-oriented marketplace allowing projects to settle on one mature modeling language and letting tool builders focus on delivering more useful features
3. Expected their collaboration would yield improvements in all three (Booch, OMT, OOSE) earlier methods.

As their unification effort formalized, they set the following four goals to focus their efforts:

1. Enable the modeling of systems (and not just software) using object-oriented concepts
2. Establish an explicit coupling to conceptual as well as executable artifacts
3. Address the issues of scale inherent in complex, mission critical systems
4. Create a modeling language usable by both humans and machines

With the help of corporate partnerships of IBM, Microsoft, Digital Equipment Corp, Hewlett Packard and others, in 1996 UML 1.0 was released. UML 1.0 was a modeling language that was well defined, expressive, powerful and generally applicable. It was submitted to the Object Modeling Group (OMG – the governing body for Object-Oriented methods) in January of 1997 as an initial Request for Proposal (RFP). In an attempt to improve the clarity of UML 1.0 semantics and to incorporate contributions from new partners, UML 1.1 was released and eventually adopted in the fall of 1997.

Since that time, UML has been well received. Deb Melewski of Platinum Technology writes: “In many ways, 1998 has become the year of UML. Organizations have been gathering up for UML implementation, numerous books on the subject can be found in bookstores, software tools providers have been incorporating UML support into their products, and UML repositories are working towards creation. As recently as April of this year, Microsoft announced support for UML in their Office 2000 application.

Even with software support for UML, there are still companies who do not model. This lack of progression can impede the success of UML market-wide. According to Richard Soley, chairman and CEO of the Object Management Group, one of the first companies to adopt UML in 1997, UML will be embraced by these companies eventually. He comments: “It may be a lot easier to sit down and start writing, but the code suffers. Now is the time to take a look at object-oriented analysis and design. There’s no longer the excuse of there being too much to look at; everyone should be using it (Melewski 2). With statements like this coming from industry leaders, it is hard to see how UML will not become a mainstay in the object-oriented field.

### 4.3 UML Today

The UML is non-proprietary and open to all. It addresses the needs of user and scientific communities as established by experience with the underlying methods on which it is based. Rational Software, the company that employs the three main developers of UML (Rumbaugh, Grady, Jacobson) has done an excellent job of marketing their product and tool developers are rushing to support UML in their applications. As a result, UML has positioned itself to be the basis for many tools including those for modeling, simulation and development environments. But is there more for UML to achieve? It is the belief of this author that UML can be combined with some of the underlying themes of semantic networks and goal oriented analysis to create a nearly fully automated tool. Before this method is revealed, a brief introduction of the Unified Process is in order.

### 5.0 Inside the Unified Process

Although the crux of this paper will deal with the automation of UML, GORA, and the Unified Process, it is important that we complete the triumvirate of concepts by introducing the Unified Process. As taken for the latest UML book to hit the market, *The Unified Software Development Process: The Unified Process is balanced [much like UML] because it is the end product of three decades of development and practical use. Its development as a product follows a path from the Objectory Process (first released in 1987) via the Rational Objectory Process (released in 1997) to the Rational Unified Process (released in 1998) (Jacobson 4).*

The Unified Process is Use Case Driven. This makes it conducive to the semantic modeling approach we wish to employ, as use cases can be the driver for testing and verification of the inputs we plug into the system. Furthermore, as use cases mature, the system can 'learn' in parallel; therefore programmers would not have to miss a step. Secondly the Unified Process is Architecture-Centric, meaning the process cannot begin until the domain for the system has

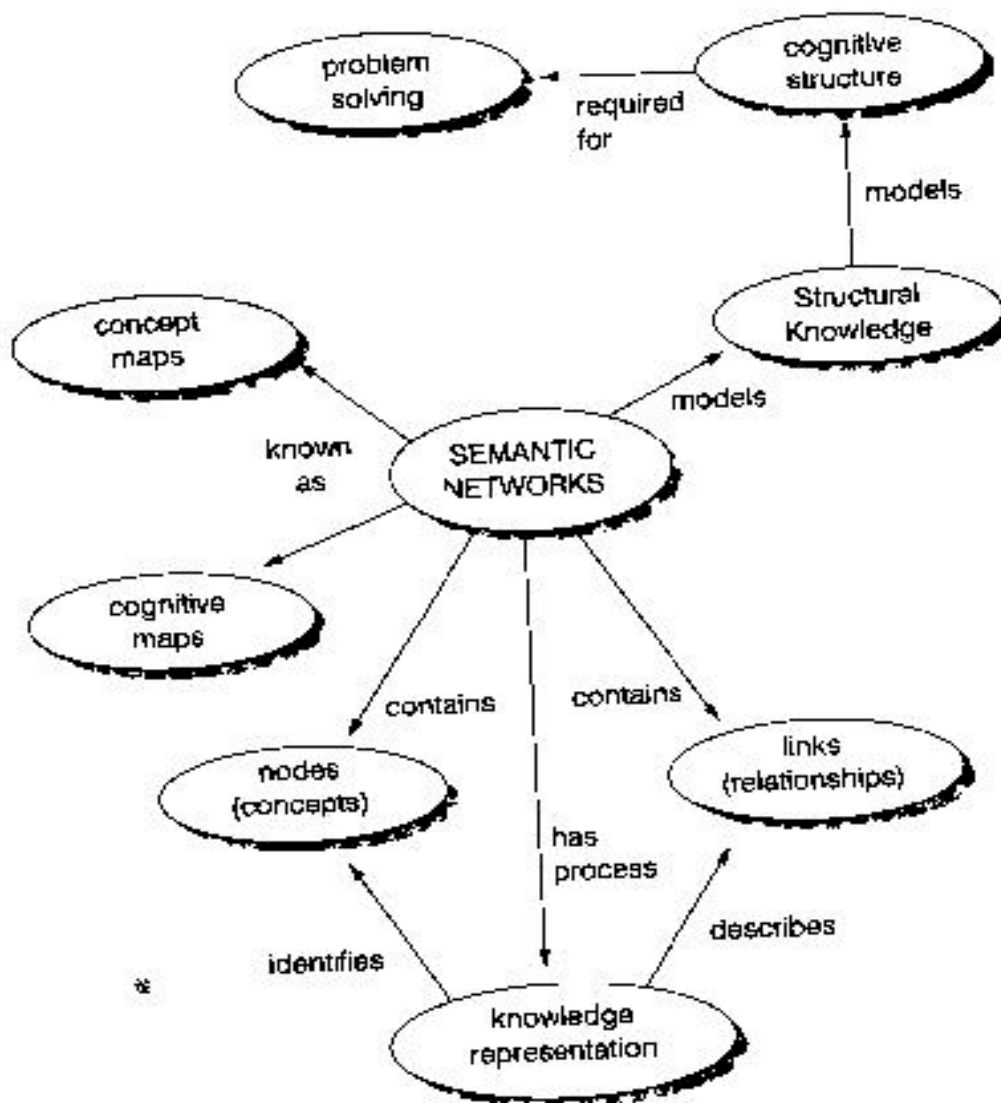
been defined. Once the domain has been established, the use cases will then be modified as they become more mature and this in turn leads to more of the architecture being discovered and this process continues until the architecture is deemed stable. This iterative and incremental nature of the Unified Process lends itself to teaching the network as it can be taught in stages and then adjusted and tested again (see appendix A for diagram).

As Appendix A shows, once use cases for the system are derived, each sequential model has a direct dependency from the Use Cases. Therefore, if the use cases can be captured in a form that is understandable by a computer, it would provide the backbone for a computer-driven process.

## 6.0 Introduction to Semantic Networks

Semantic networks (see figure 6.1) are representations of human memory structures. Current thinking is that these structures are organized semantically according to meaning that defines the relationships among the ideas in memory. Semantic networks in memory and the maps that represent them are composed of nodes (concepts or ideas) that are connected by links (statements are relationships). They aid learning by requiring learners to analyze the underlying structure of the ideas they are shaping. Their graphical nature allows for easy understanding and decreases the ambiguity found in a text-based model.

Although the use of semantic knowledge would be new to systems engineering, there is a project currently in use by the medical profession MetaMap Indexing (MMI), which uses this technique. The Semantic Knowledge Representation project is concerned with reliable and effective management of the knowledge encoded in natural texts. In the case of a new system, this text can be a design document such as a Software Requirements Specification or Interface Requirements Specification (IRS). Once this



**Figure 6.1--Example of Semantic Knowledge Tree**

Diagram courtesy of: [http://www.conroe.isd.tenet.edu.ed...Pub\\_hm/DOCS/MINDTOOL/SEMANTIC.HTM](http://www.conroe.isd.tenet.edu.ed...Pub_hm/DOCS/MINDTOOL/SEMANTIC.HTM)

information is captured via a semantic network, it can be transformed into a format usable by a neural network.

## **7.0 Neural Networks**

Neural networks are the middle piece of the Neural Semantic Method. They will process information contained in the neural network and have the ability to produce system outputs by crunching the information that it is fed. Therefore, a design team can test feed the network inputs, which simulate a wide array of conditions, and the network can forecast how the system will operate in each circumstance. The following section will give an overview of neural networks and partially describe the role they play in the Neural Semantic Method.

### **7.1 Introduction to Neural Networks**

Neural networks are composed of elements that perform in a manner that is analogous to the most elementary functions of the biological neuron (it must be noted that to suggest that a neural network will soon duplicate the functions of the human brain is still a far-fetched notion at best). Artificial neural networks contain the following characteristics

1. They can modify their behavior in response to their environment
2. Once trained, a network response can be insensitive to minor variations in input, meaning they can generalize input.
3. Are capable of abstracting input meaning they have the ability to extract an ideal from imperfect inputs

Although it is beyond the scope of this paper to describe 'how' neural networks work, it is important to understand how they are developed. By far, the most interesting part of neural networks is their ability to learn. As described by Zeidenburg in his book *Neural Networks in Artificial Intelligence*: " A neural

network is a computational model that is a directed graph composed of nodes and connections between the nodes. Each node's activation is based on the activations of the nodes that have connections directed at it (Zeidenburg15). This means, similar to how a system responds when various states are present, a neural network can be trained to respond similarly by programming each node (which represents an action) to act at certain times. However before these nodes can operate successfully, they must be trained. Training is accomplished by sequentially applying input and adjusting the network weights in an attempt to yield the desirable (predictable) result.

### ***7.2 Usage in Semantic-Neural Method***

Using the semantic network as a baseline, a neural network would be set in place to emulate the processes modeled in the semantic network. As a precondition to this happening, the semantic model must be complete and thoroughly examined as once nodes are taught to the system, eliminating one node can have a ripple effect as it may cause the restructuring of all nodes below them. Once this is done, training can begin. Neural networks can be trained two ways: Supervised training or Unsupervised training.

Although most experts argue the implausibility of supervised training stating: "it is difficult to conceive of a training mechanism in the brain that compares desired and actual outputs, feeding processed corrections back through the network (Zeidenburg 23). Setting all criticisms aside, it is the belief of this author that supervised training is the best approach for this type of system. Once all of the nodes are programmed, a developer can tweak the network (or better yet the learning rules of the neural network) to perform as desired. Being able to come up with a predictable outcome is one of the tenants of systems engineering and computer programmers are no strangers to tweaking code to produce desired results.

### **7.3 Feasibility**

There is a population in the systems engineering profession who might say that employing neural networks in a design process is far-fetched to say the least. In defense of the method, there are presently mature networks in existence that make use of neural networks. The Adaptive Resonance Theory Network by Carpenter and Grossberg and the Functional Link Network by Pao are two of the more mature networks in existence. These two networks are sixteen and eleven years old respectively and have received critical acclaim within the Artificial Intelligence community.

With the confidence that a functional neural network is plausible, the final piece of the puzzle, Goal Oriented Requirements Analysis (GORA), allows us to model non-functional requirements.

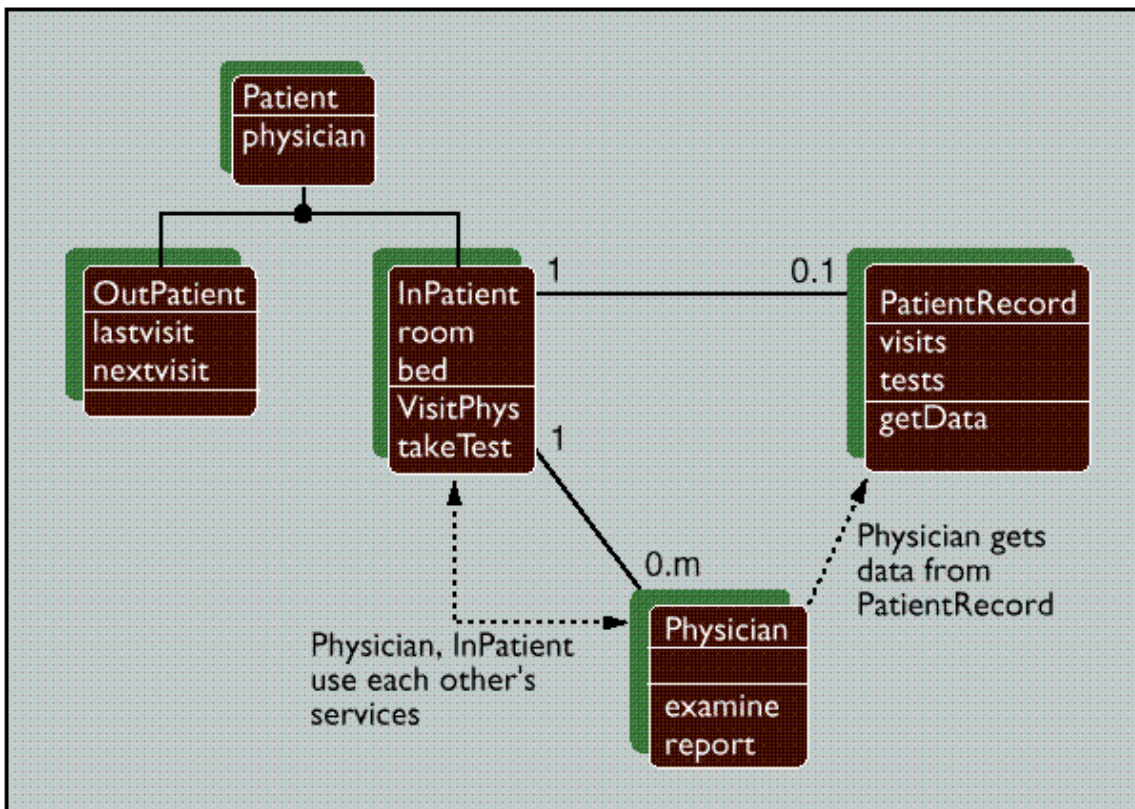
## **8.0 Goal Oriented Requirements Analysis**

Goal Oriented Requirements Analysis is a process that is the brainchild of Eric Yu, John Myopoulos and Lawrence Chung, all of the University of Toronto. Their goal was to come up with a requirements analysis model which can capture functional (sometimes called behavioral) and non-functional (also referred to as non-behavioral) requirements in unison. Whereas traditional requirements analysis techniques offer a way to map objects and their relationships to each other, they do not have the capability to incorporate non-functional requirements in their model.

As figure 8.1 shows, there is no room for non-functional requirements in the typical Object Oriented Design approach. It has been proven however that omission of non-functional requirements can be detrimental to the success of a system. Alan Davis states: "Knowing how to specify behavioral requirements for a software system is only half the battle. All applications, from the most trivial to the most complex, have additional requirements that define the overall qualities or attributes to be exhibited by the resulting software system (Davis 307). Even

if these non-functional requirements are not explicitly defined, it is still up to the developer to create a design which takes these into account.

Even though most development companies would agree that non-functional requirements are significant. To date there has been no effort to model both types of requirements together. In attempt to fill this void, GORA introduces the concept of the softgoal. The softgoal is the non-functional object which mirrors the function of an object in the traditional sense – that is one used for functional requirements (i.e. one having a name, attributes, and operations).



**Figure 8.1—Typical Object-Oriented Design**

### 8.1 Softgoals Defined

Before defining a softgoal, let's deal with the concept of a goal. In an Artificial Intelligence context, a goal is satisfied when its sub-goals are satisfied. Therefore, if a goal had five sub-goals, once those five have been achieved, the goal is also achieved (or satisfied). Softgoals have a slightly different verification

method. Softgoals are satisfied. The word satisfied comes from philosopher Herbert Simon who came up with the words meaning: ". . .to get a result that is good enough."(Mautner 1) His book The Models of Man (published in 1957) is an excellent reference for more information on this idea.

With this in mind, a softgoal is satisfied as defined by Yu: ". . . when there is sufficient positive and little negative evidence for this claim, and that they are unsatisficeable when there is sufficient negative and little positive evidence for their satisficeability.(Yu 8) Therefore, if a softgoal had five sub-goals,

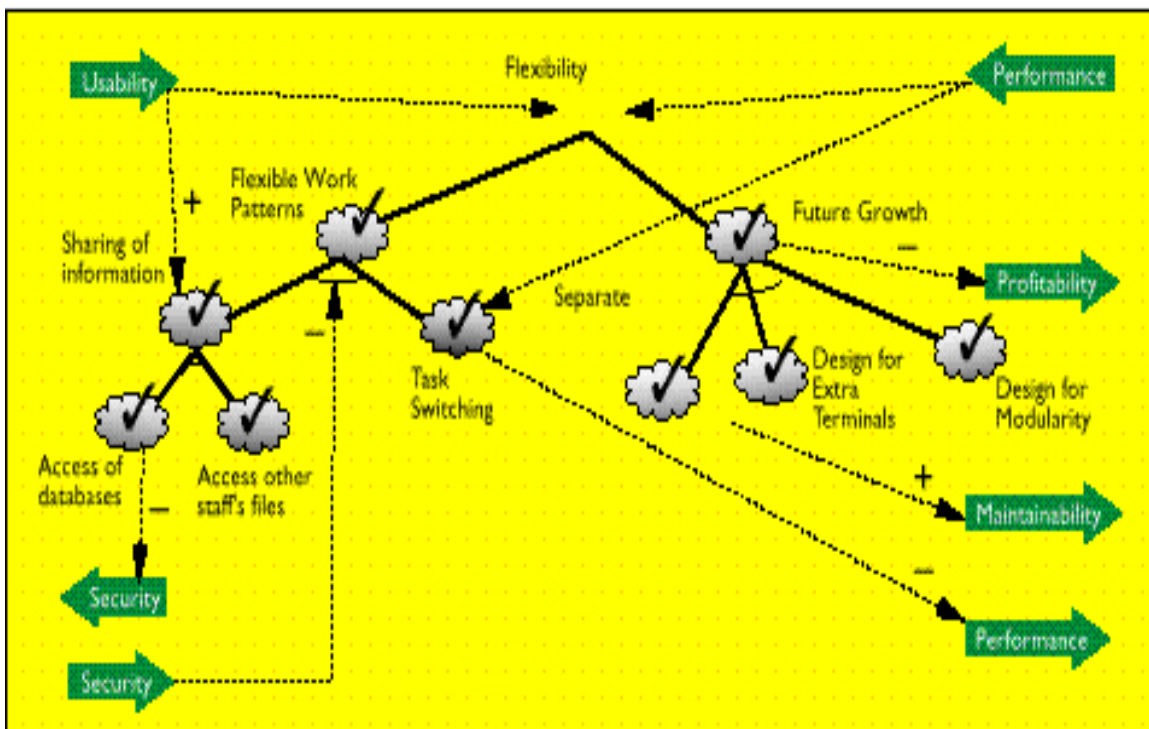


Figure 8.2—Softgoal Tree

### Legend:

**Arch** -> denotes an AND relationship between nodes (if arch is not present, then the relationship is OR). In an AND relationship, the goal (G) is satisfied when all of  $G_1, G_2 \dots G_n$  are satisfied and there is no negative evidence against it. Conversely in an OR relationship, the goal (G) is satisfied when one of  $G_1, G_2 \dots G_n$  are satisfied.

✓ -> means sub-goal has been chosen to be accommodated by the new system

<b>+(G<sub>1</sub>, G<sub>2</sub>)</b>	-> goal G <sub>1</sub> contributes positively to the satisficing of goal G <sub>2</sub>
<b>-(G<sub>1</sub>, G<sub>2</sub>)</b>	-> goal G <sub>1</sub> contributes negatively to the satisficing of goal G <sub>2</sub>
<b>➡</b>	-> softgoal tree which has receives contribution from the current tree (arrow pointing in opposite direction indicates softgoal tree exerts contribution to current softgoal tree)

if three of the five sub-softgoals were satisfied (as sub-goals can be satisfied too), then the main softgoal can then be said to be satisfied. Once softgoals have been

identified and elaborated, they are then analyzed in relation to one another. The two types of relationships used are AND and OR. Therefore, a softgoal can either be satisfied with sub-softgoal A AND sub-softgoal B or with sub-softgoal A OR sub-softgoal B.

In examining figure X above the softgoal tree for FLEXIBILITY is divided into two sub-goals which have an OR relationship. Therefore, flexibility can be satisfied by satisficing the softgoal FLEXIBLE WORK PATTERNS or FUTURE GROWTH. These two softgoals are further broken into other softgoals.

In addition to those relationships, arrows denote other non-functional requirements and their relationship to the softgoal tree in question. For example, SECURITY (in the bottom left corner) exerts a negative contribution on FLEXIBLE WORK PATTERNS, whereas satisficing the softgoal MAINTAINABILITY receives a positive contribution from the softgoal SEPARATE PERFORMANCE STANDARDS.

## ***8.2 Functional Requirements Analysis***

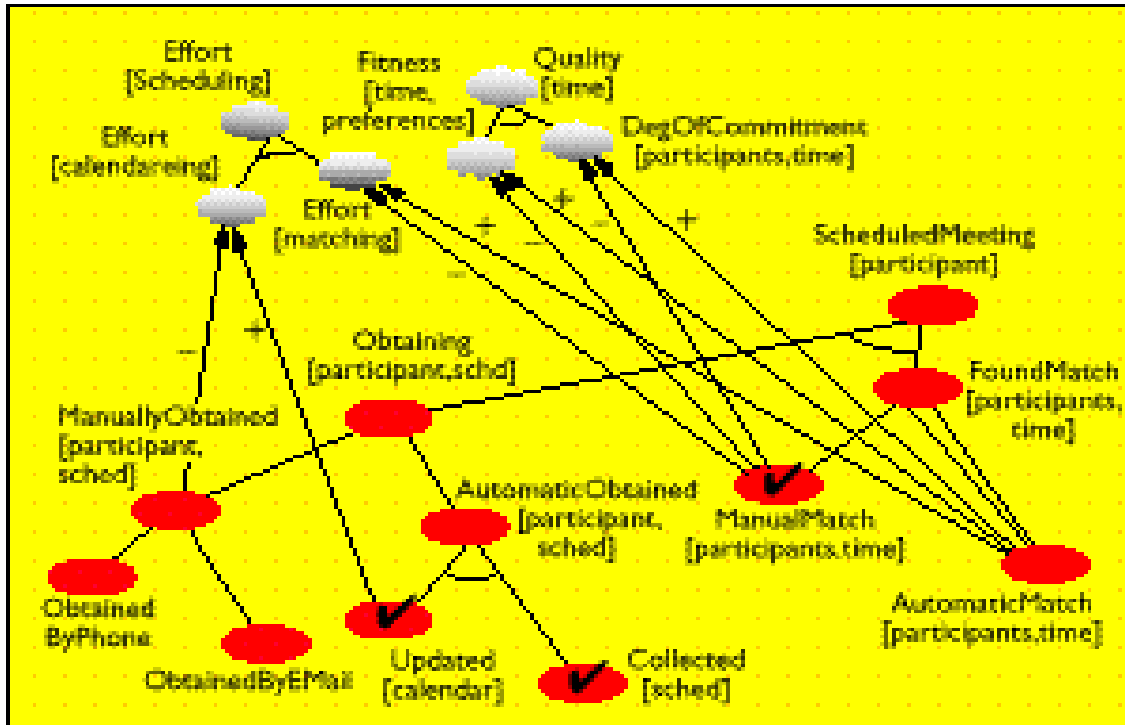


Figure 8.3—Goal-Oriented Analysis Diagram

**Legend (also see figure 8.2)**

- Ellipses -> denote functional requirements
- Clouds -> denote non-functional requirements

The great benefit of GORA is its ability to represent both functional and non-functional requirements in one diagram. The diagram can be read in two halves with the softgoal tree structure in the lower half and the functional goals shown on top. There is a multitude of information that can be gathered from figure 8.3. From this diagram, all requirements can be viewed and presented to a customer or design team for perusal. Although it would behoove the analysts to use a traceability matrix or another traditional form to track requirements, appendix B serve as an excellent overview for a system.

Now that we have elaborated on the various pieces that compose the Semantic-Neural Method, it is now fitting to describe how they interact to produce a dynamic way to model a system and assist in all systems engineering activities from idea conception to retirement.

## 9.0 The Semantic-Neural Method

The semantic-neural method is an attempt to use technology to further enhance reuse in terms of modeling and systems design. This section will explain, in detail, how it attempts to achieve this goal.

### 9.1 *Semantic-Neural Method in Theory*

Semantic networks are: A brain-like data structure for storing unstructured, 'common sense' knowledge by accessing a semantic network correctly, you should be able to deduce new knowledge not explicitly represented in the original network. Given the number of things you know your brain must use a parallel search to deduce stuff, so searching in parallel is used for the computer semantic network too. This definition is widely accepted for semantic networks. They aid learning by requiring learners to analyze the underlying structure of the ideas they are shaping.

For the process to be most effective, the method must be used from the early stages of the system onward. Preferably at the requirements analysis level, at the latest. The work put into coming up with a semantic network can be a direct result of any efforts endured during requirements analysis. The resulting semantic network will be design independent, while still laying down a knowledge structure that can be used throughout the project.

Once the semantic network has been completed, a Goal Oriented Requirements Analysis effort can begin. Vis-à-vis to this, the neural network can be created. As analysts and developers gain a firm handle on both functional and non-functional requirements, this can be programmed into the neural network. There is a definite link between the neural network and the semantic network as the neural network can be seen as a lower level model of the semantic network.

Where the semantic network is basically a layout/framework for representing concepts and ideas and the relationships between the two. The neural network can perform computational processes on this information when given an input which will provide a definite (predictable) output. The task of

building the neural network, although top-heavy in the requirements analysis and design phase, is a continuous process. As new information is acquired, it must be fed into the neural network for analysis. Once this is done, the neural network will be able to re-evaluate all of the nodes and determine what affect this has on the overall system, if any.

Upon completion of the preceding two steps, system production will be able to begin. With the concepts of UML and GORA already fed into the system, the neural network will be able to generate objects/diagrams that can be replicated in development. Even the abstract non-functional requirements can be placed in this environment and accounted for. About the only task the neural network cannot do is the actual programming involved in developing the product.

## **9.2 The Semantic-Neural Method – A Pedagogical Approach**

The Initial Stage of the Semantic-Neural Method is setting up the Semantic network. This requires the following seven steps:

1. Set boundaries for the system domain
2. Identify important concepts
3. Create, define and elaborate nodes
4. Construct links and link concepts
5. Evaluate the semantic net and if needed go back to step 1
6. Develop the Neural Network
7. Train the Neural Network

Once the semantic knowledge tree is setup the creation of the neural network can begin.

### 9.2.1 Setting the boundary for the system

This step is a pre-cursor for the neural network. Neural networks can only operate in a pre-defined environment/domain. The developers of the system must be able to fully describe the areas in which this system is to perform so that

once the knowledge structure is formed, it can account for all situations. So a system which monitors the air temperature outside of a high rise building must be able to operate in all types of weather. Its sensors must be able to understand the difference of a reading due to excessive frost on its sensor and a reading when the sensors are clear.

#### 9.2.2 Identify important concepts

This can be thought of as an early crack at requirements analysis. In attempting to understand the important tasks the system is to handle, developers can plot these into the knowledge representation (semantic network) and begin to work with the customer to further define these processes.

#### 9.2.3 Create, define and elaborate nodes

By using partitioning, break the system into functional parts/processes. These nodes can then be elaborated into lower level function.

#### 9.2.4 Construct links and link concepts

Knowing how objects (nodes) relate to one another is extremely important when designing a system. If X happens to Object A, how does it affect other objects? By properly understanding the relationship between concepts, it aids in the prioritization of tasks and further comprehension of the system.

#### 9.2.5 Create and Evaluate the semantic network

Before moving on to the next step, the semantic network must be thoroughly analyzed (by more than one person) to make sure that a correct representation of the system has been established. Even if some concepts are plotted but not fully elaborated, they must be part of the semantic network before proceeding. The reason for this is two fold: A. If a node is missing, it is not as simple as just adding it in to the network. Relationships must be defined, more than likely adjusting some of the pre-existing nodes. B. If you neglect a major

system function and proceed to constructing the neural network, the time that it would take to backtrack and insert the concept would require far more work than to verify the network beforehand.

### 9.2.6 Develop the Neural Network

This step is obviously easier said than done. It takes careful programming but if the semantic knowledge tree is correct and unambiguous, developing the neural network becomes easier. Before this step is initiated, all requirements should be frozen and the neural network that evolves can be assigned a version number. The process of adding functionality to a neural network is fairly straightforward, however when it comes to modifying existing functions, the task becomes more involved. The idea behind building a neural network is to reduce the amount of time that is spent modeling and testing the system, not to increase it.

Developing a neural network is an incremental process. Ideally, the initial neural network will have the latest UML syntax programmed in and also have a utility for non-functional requirements analysis. Programmers will be able to hard-code \*\* requirements into the network based on specific parameters (i.e. priority, time intensiveness, relationship to other requirements, etc). It must be noted that these variables will be addressed in neural network nomenclature, which have the ability to rate attributes and make suggestions based on input.

Once the first iteration of the network is completed, Step 5 should be revisited, using the most recent version of the SRS. Any changes in the SRS which take place during iteration should be ignored, unless they change functionality completely. Once system functionality is stable, Step 7 can be visited.

*\*\* Note: The Neural Network will not have to be built from ground zero. Initially, the program will ask the user to enter the names of the nodes (and sub-nodes) and assign quality ratings to each. The ratings assigned will aid in the construction of the Goal Oriented Object Tree (See Figure 8.3) which can then be tailored by the user. Developer will also be prompted to enter system objects which will be used for UML representation of system. The neural network will*

*maximize its ability when objects are added or functionality is modified. In this case, it will be able to update all the data it contains make suggestions on how to maintain current performance even with the new changes.*

### 9.2.7 Train the Neural Network

At this point, the neural network is complete. It is now up to the developers maximize its capabilities throughout the system's life.

Programmers who work on the actual system should also do development and training of the neural network. A rotation between two programmers, who would work on the network for two weeks and then work on the system for two weeks, would ensure that there is a definite consistency between the two projects.

## **9.3 Requirements Analysis**

Once the knowledge structure is in place, requirements analysis can begin. We will use Goal Oriented Requirements Analysis (GORA) for this. As we enter into this stage, it is assumed that an SOW has already been received and all requirements have been identified and stored preferably in a requirements traceability matrix. Now GORA can take place. As objects are identified (name, attribute, methods) and cardinality is established, they can be fed into the system. As they are entered in, knowledge from the semantic network is used to determine how each object plays a role in the overall functionality of the system. It is important that all requirement ambiguities are resolved with the customer before being programmed into the neural network. Even though the neural network is capable of identifying ambiguities between objects, it cannot resolve heuristics.

Therefore it makes sense logically, it will not be captured by the network whereas to a human observer might detect the mistake without hesitation. In Arthur Riel's book *Object Oriented Design Heuristics* he attempts to offer sixty or so guidelines for Object-Oriented design and development, which can also be

applied to requirements. But unfortunately his list of do's and don'ts are not suited for computation by a computer.

#### **9.4 The Maturation Process**

Jumping ahead to the implementation, it does not matter how the system is implemented. What does matter is that up-to-date information is kept in the semantic and neural networks. Many noted semantic networks experts warn that it does no good to simply add a fact [node] to the network, in order to see a change, you must re-evaluate the entire net. This means that as new information is learned about the system, varying from system performance results to a modification in an algorithm. Although this task could possibly consume at least one half full-time man-hours, it the benefits far outweigh the drawbacks.

## **10.0 The Semantic-Neural Method in Practice**

### **10.1 Developing the Networks: Semantic and Neural**

In order to show the SNM in practice, the following hypothetical sub-function will be modeled. Here is a brief problem statement. Design a system that calculates the wind chill in a temperature. We will overlook the hardware design of the system and instead concentrate on the computational features.

#### **1. Setting the boundary for the system**

The system will be designed for outdoor use and capable of functioning in temperatures ranging from  $-50^{\circ}$  Celsius to  $50^{\circ}$  C. System will be for land use only and be capable of recording values from 5280 feet below sea level to 5280 feet above sea level.

#### **2. Identify important concepts**

The system will be free standing and be able to monitor how hard wind is blowing.

The system will be able to transmit readings to central computer every thirty (30) minutes

The system must be able to adjust to weather conditions (i.e. rain, snow, sunshine)

The system will calculate a temperature once a minute, which will be archived as empirical data.

The system will use the formula provided by the National Weather Service to determine wind-chill factor.

### **3. Create, define and elaborate nodes**

For simplicity, we will only deal with the root of each node. Commonly, nodes will be elaborated into more specific functionality, hence the term knowledge tree. In most cases, each node can be thought of as a separate tree with the main tree (shown in Appendix B).

Node 001 – Wind determination

Inherent to the term “wind-chill temperature” is the measurement of the wind. This function will measure the wind in knots.

Node 002 – Extraneous weather sensors

When taking a measurement of the air, there are various factors, which can skew this reading. The unit must determine these factors (such as shade, rain, frost etc) and adjust temperature readings accordingly.

Node 003 – Transmit information

Measurements will be transmitted to a central computer every thirty (30) minutes.

Node 004 – Calculate temperature (wind-chill adjusted)

Taking the readings given from Node 001 and Node 002, the wind-chill temperature will be calculated.

Node 005--Store All data

All readings (each minute) will be archived. This intermediate information will also be sent to the main computer along with average wind-chill temperature.

#### **4. Construct links and concepts**

The system has two initial processes: *Determine Temperature* and *Determine Weather Conditions*. The readings from here are sent to a process which *Calculates Wind-chill*. Once calculated the temperature is archived by an *Internal Archive* and finally sent to the *Central Computer*, which acts as a monitor for the entire system.

#### **5. Create and evaluate Semantic Knowledge Network**

See Appendix B for semantic knowledge tree. This step is fairly difficult to quantify. The best way to approach is to conduct a meeting with the stakeholders of the system and walkthrough the system.

#### **6. Develop the Neural Network**

This process takes place as the system is being evolved. In most cases the net will be complete before the system is developed and ready for testing.

#### **7. Train the Neural Network**

This process is outside the scope of the paper. However, training is similar to debugging code, as a developer must tweak the network to achieve desired results. As explained earlier, the network can respond to various inputs (or states) and produce an outcome. The weights given to inputs must be exact so all results are uniform. Phillip Wasserman's book: *Neural Computing: Theory and Practice* provides excellent insight to this process.

### **11.0 Benefits of the Semantic-Neural Method**

The benefits of the Semantic-Neural Method are maximized by large systems, which are either evolutionary systems or product line systems. These

systems best lend themselves to the incremental approach to systems development as opposed to the one-time development process waterfall or spiral models provide.

The ability to adopt to change is one of the major features of this model. As noted in *Tried and True Object Development: The need for continuous development of new features originates from the changes on customer's business environment or technology*" (Jaaksi186). When these changes occur, functionality is usually added or enhanced, resulting in an upgraded system. However the development of the new functionality will be assisted and restricted by the architecture and components from the previous release(s).

Even though much of the code is present, all too often, the code is not properly commented or the design is not easily understood. When this occurs, an upgrade becomes difficult. Jaaksi comments in the aforementioned book:

Creating new functionality over an existing system by reusing the code of earlier releases and adding to it may sound like an easy way to make money, especially if the stem has established a strong market position. Unfortunately, developing an evolutionary system is not that easy. For one thing, the architecture does not improve automatically over time. Often it does the opposite. 188

In using the semantic neural method, a developer can reuse as much or as little of previous work in designing a subsequent system. Moreover, the neural network can identify inconsistencies on the fly as (once trained) it "knows" how objects/nodes/function relate to each other.

Secondly, SNM provides a global way to document and model systems. With the advent of standards within the modeling community, a method, which incorporates these standards and produces useful diagrams and models that can be archived is invaluable. Moreover, the neural network can determine how the system will act under varying conditions. Given an input, the network can predict the outcome.

Thirdly, the knowledge representation tree provides a simple pictorial of the system which can be viewed by stakeholders for evaluation. Also the subsequent, more detailed diagrams can provide insight into how specific requirements are satisfied.

Fourth, in terms of reuse, SNM can streamline changes and create new models on the fly, as opposed to leaving it to the developer to try and figure out inconsistencies.

Finally, it also serves as a tool to communicate with stakeholders. The semantic knowledge tree can explain system functionality in terms they are familiar with. Stakeholders can see how the system will function and make suggestions that can help solidify requirements.

## **12.0 Conclusions**

Exploring a new process has been a very exciting, yet daunting experience. The SNM may never see the light of day, but the research and critical thinking put in to give it some life has been a worthwhile effort. For the most part, much of the technology already exists to bring this idea into existence. The extra work comes in creating and training, application-specific neural networks. The development cost here might be higher than usual, however if a company is sure of its future (in terms of the type of product it wishes to produce in the future) the benefits are extraordinary.

**In terms of the future, if a narrower scope can be defined, and possibly a small prototype established, it would pave the way for more formal attempts to document and market this method.**

## 13.0 Bibliography

Author Unknown. 27 April 1999 "Wind Chill"

<http://www.gunstock.com/windchill.htm>

Author Unknown. 27 April 1999 "Wind Chill Factor"

[http://classes.atmos.uiuc.edu/120/wind\\_chill.html](http://classes.atmos.uiuc.edu/120/wind_chill.html).

Author Unknown. 6 March 1999 "Using semantic networks as a Mindtool"

[http://www.conroe.isd.tenet.edu.ed...Pub\\_htm/DOCS/MINDTOOL/SEMANTIC.HTM](http://www.conroe.isd.tenet.edu.ed...Pub_htm/DOCS/MINDTOOL/SEMANTIC.HTM)

Author Unknown. 24 February 1999 "Semantic Networks"

<http://webservices.comp.vuw.ac.nz/comp/Courses/473/1997/Lect2/tsld029.htm>.

Author Unknown. "Introduction to Semantic Networks" 3 March 1999

<http://infosys.king.ac.uk/Circle/Courses/AI/SemNet/TeachSemnet>.

Ambler, Scott W. "The Unified Modeling Language v1.1 and Beyond: The Techniques of Object-Oriented Modeling" Cambridge University Press

1998 2 March 1999 <http://www.ambyssoft.com/umlAndBeyond.pdf>.

Brill, James H. The Journal of the International Council on Systems Engineering

"Systems Engineering – A Retrospective View" Volume, 1 Number 4 1998.

Chung, K. L. Representing and Using Non-functional Requirements for Information System Development: A Process Oriented Approach Ph.D.

Thesis, also Tech. Rpt. DKBS-TR-93-1, Department of Computer Science, University of Toronto, June 1993.

Davis, Alan. Software Requirements: Objects, Functions, & States. New Jersey: Prentice Hall 1993

Fowler, Martin. "Why Use the UML?" Software Development Online 5 March 1999 <http://www.sdmagazine.com/uml/focus.fowler.htm>.

Franklin, Stan and Graesner. "Art Is it an Agent, or just a Program?: A taxonomy for Autonomous Agents" Institute for Intelligent Systems: University of Memphis (excerpted from Proceedings of the Third International Workshop on Agents Theories, Architectures, and Languages) 1 April 1999  
<http://ww.msci.memphis.edu/~franklin/AgentProg.html>.

Halladay Steve and Wiebel Michael. Object Oriented Software Engineering Brisbane: Prentice Hall, 1993.

Jaaksi, Ari; Aalto, Juha-Markus; Aalto, Ari; Vatto, Kimmo. Tried and True Object Development: Industry Proven Approaches with UML. New York: Cambridge University Press, 1999.

Jacobson, Ivar; Booch, Grady; and Rumbaugh, James. The Unified Software Development Process. Reading: Addison Wesley 1999.

Keller, Paul E. "Neural Networks: What are Neural Networks?" 11 April 1999 Pacific Northeast National Laboratories  
<http://www.emsl.pnl.gov:2080/proj/neuron/neural/neural.ann.html>.

Lee, Richard C and Tepfenhart, William M. UML and C++ A Practical Guide to Object-Oriented Development. New Jersey: Simon & Schuster 1997.

Mautner, Thomas. "The Penguin Dictionary of Philosophy" 2 May 1999  
<http://www.utilitarianism.com/satisfice.htm>

- Melewski, Deb. "UML Gains Ground" Platinum Technology 1 March 1999  
[http://www.platinum.com/products/reprint/uml\\_adt/htm](http://www.platinum.com/products/reprint/uml_adt/htm).
- Myopolous, John; Chung Lawrence; Yu, Erik From Object-Oriented to Goal Requirements Transactions of the ACM January 1999.
- Papurt David M, Inside the Object Model. New York: Sigs Books, 1995.
- Riel Arthur J. Object-Oriented Design Heuristics. Reading: Addison Wesley, 1996.
- Rosenberg, Doug. "UML Applied: Nine Tips to Incorporating UML into Your Project 21" February 1999  
<http://www.sdmagazine.com/uml/focus.rosenberg.html>.
- Russell, Ingrid F. "Neural Networks" 11 April 1999 Department of Computer Science  
[http://uhavax.hartford.edu/disk\\$userdata/faculty/compsci/www/neural-networks-tutorials.html](http://uhavax.hartford.edu/disk$userdata/faculty/compsci/www/neural-networks-tutorials.html).
- Sebesta, Robert W. Concepts of Programming Languages Third Edition. Reading: Addison Wesley, 1996.
- Sullo, Gary C. Object Engineering Designing Large Scale Object-Oriented Systems. New York John: Wiley & Sons, 1994.
- Wasserman, Philip Neural Computing: Theory and Practice. New York: Van Nostrand, Reinhold 1989.
- Yu, Erik and Mylopoulus John. University of Toronto "Why Goal-Oriented Requirements" 11 April 1999  
<ftp://ftp.cs.toronto.edu/pub/eric/REFSQ98.html>.

Zeidenberg, Metthew. Neural Networks in Artificial Intelligence. Great Britain:  
Simon & Schuster, 1990.

## **Appendix A – Models of the Unified Process**

**Appendix B – Semantic Model of Wind Chill System**

