A Hybrid Object-Oriented Environment Integrating Neural Networks and Expert Systems

Sukhdev Singh Khebbal

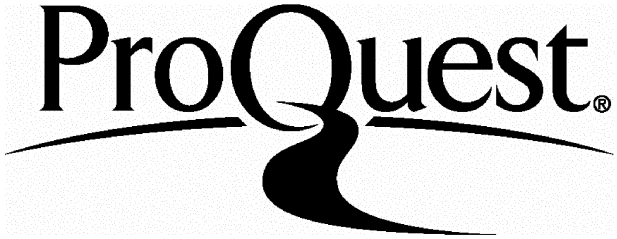
a thesis submitted for the degree of

Doctor of Philosophy in Computer Science

University of London

Department of Computer Science
University College London
Gower Street, London WC1E 6BT.

July 1995



Abstract

This thesis investigates the fundamental issues and methods for building hybrid systems. It presents an object-oriented hybrid environment (**ORBERON**) for integrating symbolic processing techniques, such as expert systems, and adaptive processing techniques, such as neural networks. The ORBERON environment was the inspiration for the European Community **Esprit III HANSA** project, which has produced a framework for the rapid integration of industry standard tools and novel artificial intelligence shells. The design and implementation of the HANSA framework is also detailed in this thesis.

This thesis is composed of five major parts: (i) the fundamental issues, arguments and methodologies for integrating symbolic and adaptive processing; (ii) the design and implementation of the ORBERON environment; (iii) the implementation of a complex real-world problem using the ORBERON environment; (iv) the design of the Esprit HANSA Framework and (v) the assessment of the hybrid approach, the ORBERON environment and the hybrid solution to the real-world problem.

To establish the arguments for integrating symbolic and adaptive processing, the structures, applications, strengths and weaknesses of expert systems and neural networks are examined. These techniques were chosen because they represent the most popular and well developed examples from the symbolic and adaptive disciplines. To capitalise on their complementary properties, various methods for integrating these techniques are reviewed and a development cycle for constructing hybrid systems is described. To support this hybrid approach, a classification scheme is proposed that categorises hybrid systems with regards to functionality, communication, and processing architecture.

The ORBERON environment is constructed around an object-oriented framework, so that each processing technique can be represented as an object and can communicate with other techniques via a message-passing mechanism. This approach allows *design, communication* and *execution* autonomy for techniques. The techniques inherit a standard communication protocol when encapsulated in the environment's ***Generic Interface Object.*** Techniques active within the environment are managed via a ***Dynamic Environment Manager.*** The design and implementation of these core components is examined in detail.

The functional validation of the ORBERON environment was undertaken by implementing solutions to a small proof of concept problem for Profit Trend Analysis and a complex real-world Cargo Consignment problem, from British Airways. The design and implementation of hybrid solutions to both problems are detailed. The hybrid solution to the Cargo Consignment problem and the viability of adopting a hybrid philosophy, was assessed by comparing the results of the hybrid approach with the results of the existing purely expert system approach. The ORBERON environment was assessed with reference to current object-oriented integration methods.

The main research contributions of this work have been: (i) the analysis and comparison of symbolic and adaptive techniques; (ii) the formulation of a novel classification scheme for hybrid systems; (iii) the design of a development cycle for hybrid systems; and (iv) the ORBERON environment for building hybrid systems. This research has also produced an edited book titled "Intelligent Hybrid Systems" and produced five publications.

Acknowledgements

This research has been successfully completed with the encouragement and guidance of many people. Firstly, I would like to thank my supervisor Professor Philip Treleaven for his assistance, advise and critique during the research and implementation of this work.

Secondly, I am indebted to colleagues in the Department of Computer Science at University College London, for their valuable advise, encouragement and friendship. Special thanks to Graham Roberts for technical advise during the implementation of the ORBERON environment. Also many thanks to Dr. Suran Goonatilake for invaluable discussions about hybrid systems during the course of this research.

I would also like to take this opportunity to especially thank Dr. Ugur Bilge, Dr. Paulo Rocha, and Anoop Mangat for their patience and constructive criticism while reading the earlier draft of the thesis.

I would like to acknowledge Steve Osbourn and Simon Cumming from British Airways, for supplying the Cargo Consignment problem to evaluate the hybrid environment.

Finally, I would like to thank the Science and Engineering Research Council for providing financial support for the first two years of this research. My gratitude also goes to the Thomas Witherden Batt Scholarship and the Momber Scholarship for partly funding the remaining year of this research.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents an introduction to the issues and arguments for integrating alternative models of computation such as symbolic processing and adaptive processing. It concludes by outlining the goals, contributions and organisation of this thesis.

1.1. Symbolic Processing versus Adaptive Processing

With the growth of information processing, a range of computational techniques are now available for solving complex problems. This thesis presents the issues, arguments and a methodology for building hybrid systems by "gluing" together different techniques, so that their individual computational strengths can be utilised in solving sub-tasks of a complex problem. By examining the current spectrum of information processing techniques, it is becoming clear that there exists two fundamental models of computation: *symbolic processing* and *adaptive processing*.

- **Symbolic processing** is based on the logical manipulation of symbols. It encompasses traditional computing techniques, such as procedural, functional and logic programming, as well as techniques that attempt to mimic human behaviour to create artificial intelligence, such as rule-based expert systems.
- **Adaptive processing** is based on dynamic, fuzzy forms of computation and has been popularised by recent techniques such as neural networks and genetic algorithms.

Figure 1.1 shows these models of computation spanning the spectrum of "intelligent" techniques ranging from expert systems at the symbolic end to neural networks at the adaptive end.

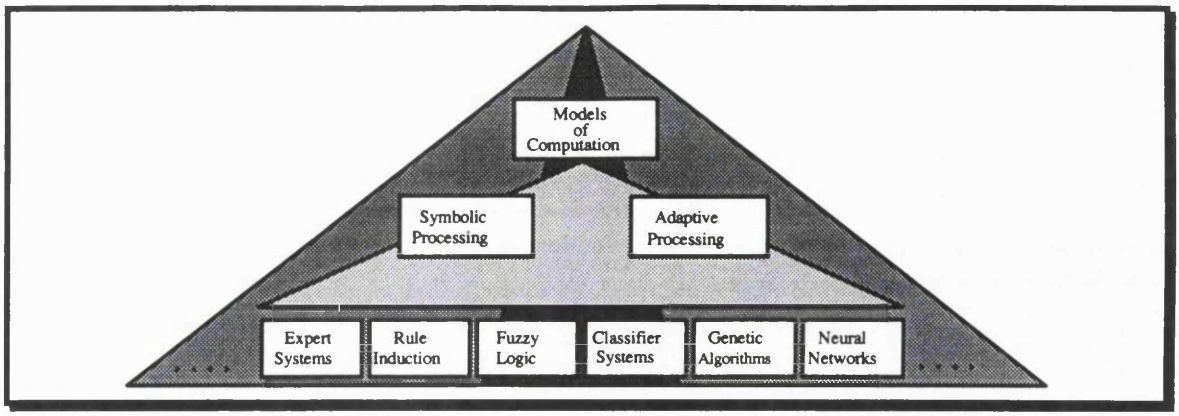


Figure 1.1 — Spectrum of information processing techniques.

The properties of these intelligent techniques vary from the rigid symbolic characteristics of expert systems to the more flexible nature of neural networks. For example, expert systems have to be explicitly programmed with the knowledge of a domain expert, whereas neural networks can "learn" to model the problem from the domain data. However, symbolic techniques such as expert systems and fuzzy logic can provide explanations of their reasoning process, where as neural networks cannot easily do so.

This research performs a detailed examination of symbolic processing and adaptive processing and reveals that both approaches have many complementary strengths and weaknesses that include the easy of knowledge acquisition, coping with brittleness in the domain knowledge and providing explanations for their results. By integrating techniques with matching complementary properties the developer can overcome a techniques individual limitations. In support of this hybrid strategy, there is a realisation that most solutions to complex real-world problems also require the synergy of these complementary approaches. This thesis specifically investigates the practical issues, arguments and methodologies, for integrating expert systems and neural networks which are the most prominent and well developed techniques from symbolic processing and adaptive processing. In particular, this thesis advocates constructing hybrid systems by adopting object-oriented software engineering techniques, in which different processing techniques can be represented as objects which communicate by sending messages. The application of these object-oriented techniques within a hybrid environment (**ORBERON**) for integrating symbolic and adaptive techniques is presented. The ORBERON environment incorporating neural networks and expert systems, was applied to the real-world cargo consignment problem from British Airways.

1.1.1. Individual Characteristics

From the inception of artificial intelligence, there has been a fundamental assumption that a physical symbol system has the necessary and sufficient means for general intelligence [78]. According to this view, any system that can process and manipulate symbols, such as a computer, can potentially be "intelligent". The process of logically manipulating symbols is often referred to as formal reasoning and enables one to infer sound conclusions from a collection of facts. The implementation of formal reasoning has created one of the prominent successes from symbolic artificial intelligence, namely *expert systems*. These computer programs embody the knowledge of human experts in the form of symbolic structures (e.g. rules) and apply the principles of logic to arrive at intelligent judgements about their application domain. Expert systems have been successfully applied to well-defined problem areas such as medical diagnosis, design, planning, monitoring and control [26].

Although adaptive processing has been recognised as a model of computation to rival symbolic processing, its foundations were laid down as early as those of symbolic processing. Adaptive techniques are often inspired by natural structures and processes, such as evolution in genetic algorithms, pattern recognition and the ability to "learn" in neural networks. Neural networks, the most popular of the current adaptive techniques, were developed in the early 1960's and were attempts to mimic the physical structure and learning mechanism of the brain [89].

The differences between symbolic and adaptive processing techniques can be seen by examining their theoretical and operational characteristics. Table 1.1 shows that symbolic processing is fundamentally a *rigid, serial and discrete* computing process, while adaptive processing is inherently *flexible, parallel and distributed* in it's knowledge representation and decision making capabilities.

These theoretical characteristics stem from fundamental differences between the knowledge representation and decision making processes within symbolic and adaptive techniques. Because symbolic techniques have to be programmed to represent domain knowledge and because they manipulate these symbolic concepts using rigid logic, they can produce precise repeatable results but are not very adaptable to situations where partial or uncertain information is present. Whereas, adaptive techniques because of their parallel distributed representation structures and their ability to learn from previous problem examples, can cope with partial, uncertain or new information but can produce results that are imprecise and unrepeatable.

Theoretical Characteristics		
Characteristic	***Symbolic***	***Adaptive***
Processing	Symbolic manipulation via rigid logic.	Inspired by natural adaptive principles.
Precision	Produces precise results	Produces imprecise results
Programming	Knowledge base must be explicitly programmed.	Self programming or trained from previous examples.
Representation	Sequential localised symbolic representation.	Parallel distributed representation.
Decision Process	Decisions based on logical inference through detailed rules	Decision process based on a collective view of data
Adaptability	Knowledge base generally fixed but must be maintained.	Knowledge is adaptive and extendable.
Brittleness	Brittle in presence of partial or uncertian information.	Can handle partial, uncertain and fuzzy information.

Table 1.1 — Theoretical characteristics of symbolic and adaptive processing.

Table 1.2 shows some of the operational characteristics of symbolic and adaptive techniques that stem from their theoretical foundations.

Operational Characteristics		
Characteristics	***Symbolic***	***Adaptive***
Operation	Explicit rule-following.	Operation hidden from user.
Justification	Good Explanation facilities.	Little or no explanation of results.
Domain Expertise	Domain expertise essential.	Minimum domain expertise required.
Speed	Slow as knowledge base grows.	Fast response due to parallel distributed processing.
Input Data	Inputs have to be accurate.	Inputs can be noisy (partially incorrect).
Development Time	Long development times.	Short development times.

Table 1.2 — Operational characteristics of symbolic and adaptive processing.

The symbolic structures and logical rule following in symbolic processing techniques gives them the facilities to justify the results that they produced. Unfortunately these rigid foundations often mean that a knowledge source (human or otherwise) must be available for the domain knowledge to be elicited and encoded into symbolic structures. This knowledge elicitation process can be time consuming and if the knowledge base is large, the actual operation of the technique can be slow. In most adaptive processing techniques a knowledge elicitation phase is not necessary because the techniques use either examples or raw data from the problem domain to build an internal model of the problem. This means that these techniques often have a very short development time in comparison

to experts systems [100]. The parallel distributed nature of adaptive representation techniques means that it is difficult for them to justify their answers, but does allow fast operation.

By examining the theoretical and operational characteristics of these fundamentally different techniques, it is evident that they have complementary strengths and weaknesses. Therefore, a major justification for creating hybrid systems that integrate symbolic and adaptive processing, is that the synergy of these different techniques will overcome the limitations of individual techniques.

1.2. Requirements of Real-World Problem Solving

The characteristics of symbolic and adaptive processing naturally effect the types of applications and domains that they are applied within. Symbolic processing systems are particularly suitable in domains that have well-defined technical expertise, such as configuring computer systems, chemical compound analysis and industrial planning [34]. In contrast, adaptive processing techniques have proved to be efficient when faced with problems that have unclear or questionable heuristics, such as pattern recognition, forecasting and optimisation. Examples of classic application areas for adaptive techniques are robot navigation and vision, speech processing and understanding, financial forecasting, and pattern recognition problems [51].

Although both symbolic and adaptive processing techniques perform well within their specific areas of competency, they both perform poorly outside these tight boundaries. This implies that these techniques can not be applied universally to any type of problem. Also, as complex real-world problems may have many different types of sub-tasks, one specific processing technique can not be applied to solve the entire problem. By examining the requirements of real-world problem solving, such as knowledge acquisition, robustness and explanation, it will become evident that real-world problems consist of some tasks that are best solved by symbolic processing and other sub-tasks that are best solved by adaptive processing. The following section highlights complementary strengths and weaknesses of symbolic and adaptive processing with respect to four key properties: knowledge acquisition, brittleness, high and low-level reasoning and explanation.

1.2.1. Knowledge Acquisition

Knowledge acquisition is a crucial stage in the development of real-world systems. As a process, it involves eliciting, interpreting and representing the knowledge from a given domain.

Symbolic techniques such as expert systems, normally gain their domain knowledge from a human expert and have been known to suffer from problems in knowledge elicitation and representation. The development stages involving elicitation and representation of domain knowledge are often time consuming, expensive, and potentially unreliable [25]. A key problem in eliciting knowledge from experts is the inability of experts to articulate their expertise. This is primarily due to certain types of expertise being "intuitive" [83].

Further problems include the possible existence of "gaps" in an expert's knowledge and in the correctness of an expert's knowledge. Once the appropriate domain knowledge has been acquired there is still the problem of selecting the best scheme to represent that knowledge (e.g. production rules, frames, semantic networks etc.).

Because of these problems, various learning systems have been applied to the automation of the knowledge acquisition process. There are several symbolic systems which have been used to learn rules and decision trees from "raw" domain data, e.g. rule induction [86, 71]. Although these have been used in the automated construction of knowledge bases, their popularity has been limited.

In contrast almost all adaptive techniques such as neural networks and genetic algorithms have the ability to "learn" from domain data, and form their own "internal model" of problem domain [34].

1.2.2. Brittleness

A system can be said to be *brittle* if it can only operate within a tightly constrained domain, and if it fails to function even if there is a slight change in the operating domain [47]. This *Brittleness Problem* stems from the inability of a reasoning system to cope with partial or uncertain information, inexact pattern matching and incomplete or inconsistent knowledge.

At present most symbolic systems are very susceptible to the brittleness problem because of their strict, logical reasoning structures. In expert systems this brittleness problem manifests itself in many ways, particularly in the operation and maintenance of large, complex knowledge bases [65].

Adaptive techniques offer partial solutions to the brittleness problem due to their distributed representation and reasoning. For example, in contrast to symbolic reasoning where only one operation is active at a given moment, neural networks have many computational units that process information and reason by numeric aggregation of

response from these units, across the whole network. These types of *distributed* representation and reasoning processes allow adaptive techniques to deal with incomplete and inconsistent data and also allows systems to "gracefully degrade" [91]. That is for example, if some part of a neural network become non-operational, the rest of the network will continue to function and attempt to give an answer.

1.2.3. High and Low-Level Reasoning

For a complete theory of cognition, there needs to be explanations both of how humans can do "low-level" parallel, pattern recognition type tasks as well as sequential, "high-level" cognitive tasks.

Although symbolic techniques have provided plausible models to describe "high-level" cognitive tasks such as language generation and comprehension, and goal directed reasoning, their ability to explain "low-level" pattern recognition type tasks has been limited.

In contrast, adaptive techniques such as neural networks offer the exact complementary ability. They are very good at modelling pattern recognition tasks such as visual processing and motor control but are not well equipped to model sequential, high-level cognitive tasks. Because of these limitations, some researchers have criticised adaptive techniques as inadequate models of cognition [28].

1.2.4. Explanation

The ability to provide users with explanations of the reasoning process is a desirable feature of intelligent real-world systems [15]. Explanation facilities are required both for user acceptance of the solutions generated by a real-world system, and for the purposes of understanding whether the reasoning procedure is sound [20]. Good examples of this requirement can be found in medical diagnosis, loan granting, and legal reasoning. In some domains such as medicine and finance, providing such explanations is now becoming a statutory requirement.

There have been fairly successful solutions to the *Explanation Problem* by symbolic systems such as expert systems and case based reasoning systems. In expert systems, explanations are typically provided by tracing the "chain of inference" during the reasoning process [102].

In contrast, most adaptive techniques such as neural networks find it difficult to provide adequate explanation facilities. This is due to neural networks not having explicit,

declarative knowledge representation structures but instead having knowledge encoded as "weights" distributed over the whole network [21]. It is therefore more difficult to generate a "trace of reasoning" which can be used for producing explanations. There is now a small but growing number of researchers who are interested in providing neural networks with explanation facilities by attempting to extract rules from their internal weight structures [31].

1.3. Arguments for Hybrid Systems

It is evident from the previous discussions, that symbolic and adaptive systems should be viewed not as competing models but as very ***complementary*** ones. The central arguments for constructing hybrid systems concern: *Technique Enhancement,* the *Multiplicity of Application Tasks* and for *Realising Multi-functionality* within one system.

By actually integrating these two fundamentally different techniques one can avoid many of the weaknesses inherent in each methodology, whilst capitalising on their individual strengths. This concept of *Technique Enhancement* has been the main motivation for combining symbolic and adaptive processing and would enable the production of superior reasoning systems.

In addition, most real-world problems are to complex for any single processing technique to solve in isolation. Most real-world domains have "logical" static components (that can be easily handled by symbolic techniques) as well as fuzzy, dynamic, poorly understood components (which can be handled by the adaptive processing techniques). Such *Multiplicity of Application Tasks* is well represented in real-world problem areas like financial applications, automated manufacturing and control, natural language processing and many other complex tasks. Therefore, the aim in these types of hybrids is to match techniques with particular strengths with tasks that require such properties.

The development of hybrid models also holds promise for improving our understanding of human cognition, by offering a mechanism to develop cognitive models. This "cognitive angle" of hybrid systems stems from the fact that neither current symbolic processing nor adaptive processing, appear to be sufficient for providing a basic explanation of human cognitive processing. As most cognitive tasks can be traced to regions of neuron activity, this implies that nature has a mechanism for unifying high level symbolic and low level adaptive tasks. Many researchers [45, 3] feel that *Realising this Multi-functionality* within a hybrid approach that unifies symbolic manipulation via an adaptive architecture, could provide a feasible model of robust human cognitive processing. The epilogue to Minsky and Paperts, 1987 revised Perceptrons book [73], promotes this "hybrid" view of the brain :

"Large numbers of relatively small distributed systems arranged by embryology into a complex society that is controlled in part (but only in part) by serial, symbolic systems that are added later"

As the need for hybrid systems is becoming accepted, what is required is a basic understanding of the fundamental issues, a coherent method to make qualitative assessment of different hybrid systems and methods to build real-world hybrid systems.

1.4. Research Motivations and Goals

The main goal of this research has been to investigate the basic integration issues behind hybrid systems and to construct an environment that allows the pragmatic investigation of hybrid systems. Therefore, the major motivations and goals of this research have been:

- To investigate the basic foundations, issues and pragmatic reasons for integrating symbolic and adaptive techniques. This was carried out by performing a detailed examination and comparison of expert systems and neural networks. These two techniques were chosen because they represented the most developed and popular forms of symbolic and adaptive processing.
- To highlight the complimentary strengths and weaknesses of symbolic and adaptive techniques, and to show how the integration of these processing techniques could overcome their individual limitations. These investigations concentrate particularly on expert systems and neural networks.
- To develop a fundamental framework for qualitative evaluation of hybrid systems by devising a classification scheme to clarify and categorise hybrid systems with regards to functionality, communication and processing architecture.
- To specify the interface, functionality communication and co-ordination requirements of a particular class of hybrid systems defined as *intercommunicating* hybrids. Intercommunicating hybrid systems not only offer immediate hybrid solutions, but also offer some insight into the interface and functional requirements of how different processing techniques may interact.
- To evaluate the software engineering methodology called object-oriented programming as a suitable mechanism for constructing different types of hybrid systems.
- To design and implement an object-oriented hybrid environment (ORBERON) for integrating independent symbolic and adaptive techniques for solving complex real-world problems.

- To apply the ORBERON environment to a real-world problem and show that the hybrid results can be more effective than the existing single processing technique approach. For this purpose the complex real-world Cargo Consignment problem from British Airways was used to test the environment and the hybrid approach.

The main assessment criteria to be met by the ORBERON environment are:

- **Flexibility**: The environment must allow for various ways to combine different processing techniques.
- **Extendibility**: The environment must allow the easy installation of new processing techniques.
- **Design Autonomy**: The environment must allow components of a hybrid system to be designed and implemented as independent entities.
- **Communication Autonomy**: The environment must allow components of a hybrid system to initiate and respond to communications independently.
- **Execution Autonomy**: The environment must allow components to be executed as independent entities.

1.5. Research Contributions

The main contributions of this thesis to the field of intelligent hybrid systems are:

- **The analysis and comparison of expert systems and neural networks.** These are the most well developed techniques from symbolic and adaptive processing. Their structures, strengths and weaknesses are compared and contrasted, and found to be complementary in nature.
- **The design of a classification scheme for categorising hybrid systems.** The scheme resolves the conflicting terminology that exists in the field of hybrid systems. The scheme takes into account the functionality, processing architecture and communication requirements of hybrid systems, and has been adopted by other researchers in hybrid systems [23, 63]. This work forms the basis for an edited book entitled "Intelligent Hybrid Systems" [35].
- **The identification of possible limitations of hybrid systems and the design of a development cycle for hybrid systems.** The development cycle helps to limit possible problems that may arise while building hybrid systems, such as communication, development and maintenance problems.
- **The analysis of object-oriented technology for building hybrid systems.** The use of object-oriented techniques offers many advantages including the

representation of techniques as objects, allowing structured communication via message passing and allowing the environment to be flexible and extendible.

- **The design and implementation of the ORBERON environment for integrating symbolic and adaptive processing.** Real-world problems require hybrid solutions that can handle well known, static, and precise components as well as poorly understood, dynamic, fuzzy and time critical components.
- **The design of the Esprit III HANSA (Heterogeneous Application geNerator Standard Architecture) framework.** This project was inspired by this thesis research and aims to produce a commercial object-oriented framework for rapid configuring of applications by integrating industry standard symbolic processing tools and adaptive techniques.

1.6. Thesis Organisation

The thesis work is organised into 9 main chapters:

Chapter Two - Symbolic Processing versus Adaptive Processing, surveys expert systems and neural networks because they represent the most prominent and well developed examples of symbolic and adaptive processing.

Chapter Three - Hybrid Systems: Issues and Classification, firstly examines the arguments for why hybrid systems are necessary. Secondly, a classification scheme is presented, that categorises hybrid systems with respect to functionality, processing architecture and communication requirements. Thirdly, the functional implications of different methods for integrating Expert Systems and Neural Networks are examined. This is followed by an investigation into possible problems that could be encountered with hybrid systems and concludes by presenting a development cycle for hybrid systems.

Chapter Four - Object-Oriented Integration, advocates object-oriented programming methods for building hybrid systems. The object-oriented approach to integration is examined and the basic concepts and main advantages of this approach are highlighted.

Chapter Five - The ORBERON Environment, describes the design and implementation details of the ORBERON environment. The fundamental elements that make up the environment are described, followed by the implementation details of the basic expert system and neural network components, namely the CLIPS expert system and the Back-Error Propagation simulator.

Chapter Six - Hybrid Applications, describes the design and implementation details of a proof of concept Profit Trend Analysis application and the real-world British Airways Cargo Consignment problem. The results from these hybrid solutions are also examined.

Chapter Seven - The Design and Implementation of the HANSA Framework, outlines the design and implementation of the core architecture within the Esprit III HANSA (Heterogeneous Application Generator Standard Architecture) project which was inspired by this thesis work.

Chapter Eight - Assessment, firstly makes an assessment of the hybrid approach, classification scheme and object-oriented methodology for building hybrid systems. Secondly the chapter assesses the design and implementation of the ORBERON environment and assesses the results of the hybrid solution to the British Airways Cargo Consignment problem.

Chapter Nine - Conclusions and Future Work, makes concluding comments about the hybrid approach, the ORBERON environment and future extensions to this work.

Chapter 2

Symbolic Processing versus Adaptive Processing

This chapter surveys the most prominent, well developed and representative techniques from symbolic processing and adaptive processing; namely expert systems and neural networks. Their internal structures, development cycles, application areas and programming environments are examined. From this analysis their computational strengths and weaknesses are highlighted.

2.1. Introduction

With the emergence of symbolic processing and adaptive processing as the two fundamental models of computation, there has been both industrial and academic interest in the integration of these fundamentally different approaches. To appreciate the contrasting but complementary nature of these approaches, the most prominent techniques from both processing approaches are examined. Firstly expert systems, and then neural networks, are studied with respect to their structure, development cycle, application areas and programming environments. This analysis is followed by the examination of the computational strengths and weaknesses of both processing techniques.

2.2. Symbolic Processing

The fundamental assumption of symbolic processing as stated by two of its founding theorists Newall and Simon, is that "*a physical symbol system has the necessary and sufficient means for general intelligent action*" [78]. This view of simulating human cognition was also referred to as "*language of thought*" by Fodor [29] and is commonly know as Artificial Intelligence. This symbolic framework suggests that human perception and cognition are tantamount to acquiring and manipulating symbolic representations. These symbolic structures are typically represented and manipulated within computers as serial stored programs running on a von Neumann [77] model of computation.

Since the 1960's, symbolic processing researchers have been trying to simulate human cognition by finding general methods for solving broad classes of problems within general-purpose programs. However, despite some progress, this strategy produced no

real break-through. Therefore the research emphasis shifted to concentrating on representation and search techniques. Not until the 1970's did symbolic artificial intelligence researchers realise that the problem-solving power of a program comes from the knowledge it possesses and not just from the formalisms and inference schemes it employs. This realisation led to the development of special-purpose systems that embodied the expertise of some narrow problem domain. These systems referred to as Expert Systems or Knowledge Based Systems, have been the most commercially successful implementations of symbolic artificial intelligence.

2.2.1. Expert Systems

A domain expert is a person who, because of training and experience, is

- able to be proficient and efficient within a problem domain
- good at sifting through irrelevant information to get at the basic issues of a problem
- good at recognising problems they face as instances of types which they have experienced before

It is this "operative" knowledge of a domain that is referred to as "expertise" of a person. It has been the belief of many symbolic artificial intelligence researchers that the encoding of this expertise along with specific knowledge about some problem domain, is the main ingredient to make programs behave intelligently. Therefore, considerable research has been conducted into the development of procedures and methods for elicitation, construction, and representation of knowledge within symbolic systems. An expert system is a computer program which embodies the "knowledge" of a specific area of human expertise, in such a form that the machine can offer intelligent advice or take an intelligent decision about a problem task. The process of acquiring the knowledge from an expert is known as knowledge elicitation. It typically involves a series of interviews with the domain expert, along with careful recording of observations when the expert is performing tasks [116].

2.2.2. Expert System Structure

An expert system is organised so that the knowledge about the problem domain is separate from other knowledge such as general knowledge about how to solve problems or knowledge about how to interact with the user. Having the domain knowledge separate makes it easier for the knowledge engineer to design procedures to manipulate it. The operational environment and the three main components of an expert system can be seen in Figure 2.1.

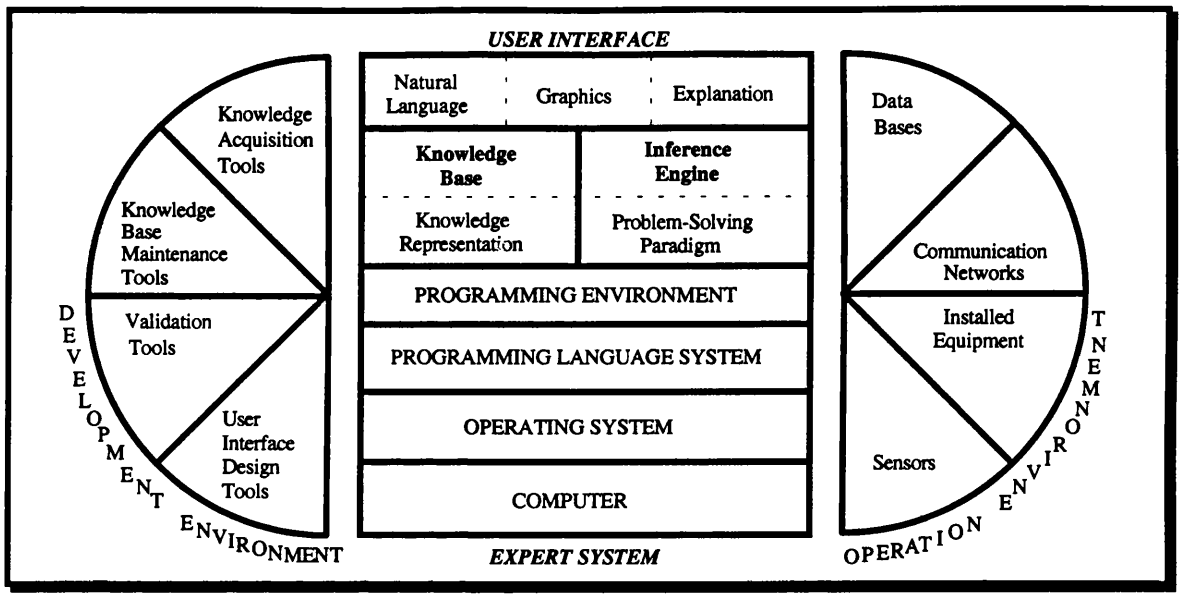


Figure 2.1— Components of an Expert System and Operational Environment.

The main components of an expert system are :

- The ***Knowledge Base*** : contains the domain knowledge as a collection of data-like facts and method-like knowledge representations.
- The ***Inference Engine*** : contains the problem-solving mechanisms that allow new knowledge to be inferred from existing information in the knowledge base.
- The ***User Interface*** : contains knowledge browsing/acquisition and inference tracing/explanation features that utilise modern graphics technology.

The inference engine contains an interpreter that decides how to infer new knowledge from existing knowledge and a scheduler that decides the order in which the knowledge should be applied. Many high-level expert system building languages such as EMYCIN [69], have the inference engine built in as part of the language. Other low-level languages, e.g. LISP [124], require the expert system builder to design and implement the inference engine. Both approaches have their advantages and disadvantages. A high level language with the inference engine built in means less work for the expert system builder. However, the builder also has fewer options regarding how knowledge can be organised and accessed. A lower-level language with no inference engine requires a greater development effort, but it provides some basic building blocks so that the system developer can tailor the control scheme to the needs of the problem domain.

The information within the knowledge base must be represented in a format that can be manipulated by the computer. There has been considerable research into finding good knowledge representation schemes that are efficient and easy to use. Each technique

provides the system with certain benefits, such as efficiency, comprehensibility and easy modification of knowledge. A comprehensive summary of the most popular techniques can be found in the *Handbook of Artificial Intelligence* [105]. Some of these popular representation and programming methods are reviewed in Table 2.1.

Method	Description	Tool
Rule-based	Uses IF-THEN rules to perform forward or backward chaining.	EMYCIN
Frame-based	Uses frame hierarchies for inheritance and procedural attachment.	SRL
Procedure-oriented	Uses nested subroutines to organise and control program execution.	LISP
Object-oriented	Uses *objects* that communicate with one another via messages.	SMALLTALK
Logic-based	Uses predicate calculus to structure the program and execution.	PROLOG
Access-oriented	Uses probes that trigger new computations when data is changed or read	LOOPS

Table 2.1—Representation and programming methods supported by expert system.

2.2.3. Development Cycle

The evolution of an expert system normally proceeds from simple to hard tasks by incrementally improving the organisation and representation of the systems knowledge. This incremental prototyping also means that the developers can profit from what they learn in implementing and testing during each cycle of the system development. One common result of incremental development is called a *paradigm shift* [44]. This occurs at some point during the development cycle, when the knowledge base may reach an unmanageable size, control becomes unwieldy and slow or the constructs within the knowledge base may seem unrelated. When the paradigm shift occurs it is normally time to redesign or re-implement the system. Long-term development efforts will be more likely to involve major paradigm shifts than will short-term projects.

Expert system development can be viewed as five highly interdependent and overlapping phases: identification, conceptualisation, formalisation, implementation, and testing. Figure 2.2 shows how these phases interact.

During *identification*, the knowledge engineer and expert determine the important features of the problem, which include the type, scope, resources and the goals of the expert system. During *conceptualisation*, the knowledge engineer and expert decide what concepts, relations and control mechanisms are needed to describe problem solving in the domain. *Formalisation* involves expressing the key concepts and relations in some formal

method (e.g. rules, frames or semantic networks). During *implementation*, the knowledge engineer turns the formalised knowledge into a working computer system. The final phase involves *testing* the performance of the prototype system and revising it as necessary.

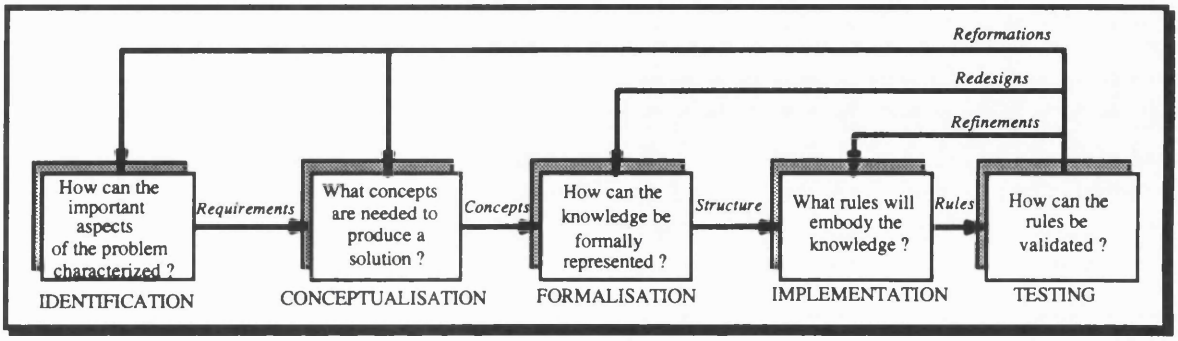


Figure 2.2 — Development phases for the expert system building.

The above development cycle is iterated through until a satisfactory system is constructed. Once this is completed the system must be field tested by the user on real-world problems. This can be often long and extensive to ensure that the system meets the user's commercial requirements.

2.2.4. Applications

The domains best suited for expert systems are ones where genuine experts exist who can articulate their problem solving knowledge and agree on solutions. The domain is best suited when the task is well understood. Within these domains, the basic activity can be grouped into a broad set of categories as shown in Table 2.2 below.

Category	*Problem Addressed*
Interpretation	Inferring situation description from sensor data
Prediction	Inferring likely consequences of given situtations
Diagnosis	Inferring system malfunctions from observables
Design	Configuring objects under constraints
Planning	Designing actions
Monitoring	Comparing observations to expected outcomes
Debugging	Prescribing remedies for malfunctions
Repair	Executing plans to administer prescribed remedies
Instruction	Diagnosing, debugging, and repairing student behavior
Control	Governing overall system behavior

Table 2.2 — Generic categories of expert system applications.

With the acceptance of expert systems in the late 1970's, a number of classic systems had been implemented in various problem domains. For a comprehensive table of application domains and successful expert systems refer to Appendix A. The systems described in Appendix A, have become important milestones in the application of expert systems to real-world problems. They include DENDRAL [12], which interprets mass spectrograms to identify chemical constituents, MYCIN [101] used to diagnose illness, DIPMETER [19] and PROSPECTOR [32] which analyse geological data for oil and mineral deposits respectively, and XCON\R1 [66] which configures computer systems.

2.2.5. Programming Environments

Most expert system programming environments are known as "Shells" and consist of three principal parts, the ***knowledge base***, the ***inference engine***, and the ***user interface***. The shell concept was originally developed by the MYCIN expert system, previously systems such as DENDRAL were one-of-a-kind systems in which the knowledge base was intermingled with the software that formed the inference engine. MYCIN explicitly separated the knowledge base from the inference engine. This was extremely important because it meant that the essential core of the expert system could be reused. The shell produced by removing the medical knowledge of MYCIN was called EMYCIN (Essential or Empty MYCIN) [70]. This allows the core inference engine, explanation, and user interface facilities to form a "shell", into which new knowledge bases about different problem domains can be loaded.

Several surveys of shells have appeared in recent years [33, 43]. Generally, shells can be classified as being either : ***Academic or Research Shells, Special Purpose Shells,*** or ***General Purpose Shells.*** Shell development began in academia, mainly as the offspring of MYCIN, EMYCIN [69] and OPS5 [30]. These research projects were quickly commercialised by many start-up companies that were set-up to develop and market these types of shells for the growing PC market.

Special purpose shells employ a base language (e.g. Lisp or Prolog or, for efficiency, a conventional language like "C") as an implementation language for the representation format, the inference engine and the user interface. The shell is restricted to accept domain knowledge in one particular representation format (normally rules). Special purpose shells enable expert systems to be built for the ***same*** *category or functional capabilities* [10] (diagnosis, configuration, planning etc.) but in ***different domains.*** Where as the general purpose shells enable expert systems to be built for different categories in different domains. The classic example of this procedure is MYCIN and the shell EMYCIN.

General purpose shells employ their base language to construct various components for knowledge representation, inferencing (forward, backward, inheritance), and the user interface. This environment can be used to build an expert system for an arbitrary category in an arbitrary domain. Commercial examples based on Lisp are ART [122] and KEE [56] .

There are a multitude of ***commercial shells*** available that range from special to general purpose shells, offering various facilities and running on various machines. Table 2.3 represent a small sample of some of the most popular systems.

Commercial Systems					
Category	*Tool*	*Use*	*Description*	*Hardware*	*Developer*
Programming Languages	LISP	General-purpose	Procedure-oriented	Lambda machines	LISP Machines Inc.
	PROLOG	General-prupose	Procedure-oriented	DEC 10's DEC 20's	Quintus Computer System
	SMALLTALK	General-prupose	Object-oriented	Tektronic 4404	Xerox Corporation
System Building Tools	EXPERT-EASE	Knowledge Acquition	Infers decision trees from examples	PC's DEC's	Expert Systems International
	RULE-MASTER	Knowledge Acquition	Infers decission trees from examples	VAX-11, Sun, HP	Radian Corporation
	TIMM	Knowledge Acquition	Infers rules from examples	Vax-11, Prime 400	General Research Corporation
Knowledge Engineering	KES	General-purpose	Logic-Based Frame-based	DEC VAX UNIX / VMS	SW Architecture & Engineering
	M.1	General-purpose	Rule-based English-like syntax	IBM PC	Teknowledge
	OPS5	General-purpose	Rule-based	VAX-11 systems	DEC

Table 2.3 — Selected Commercial Systems.

2.3. Adaptive Processing

Intelligent behaviour exhibited even by an infantile human brain can be characterised by basic processes of *learning, generalising, understanding, planning, problem solving and the ability to model and communicate with the external world.* These daily processes are performed by the human brain with an efficiency that no present generation of conventional *von Neumann architectured* computer can match. This shows that the conventional computer is very good at symbolic processing defined by explicit commands, but it remains poor at even simple pattern matching. This suggests that we have two fundamental domains of computation: the *adaptive* or *pattern processing* domain of the brain and the *symbolic processing* domain of the conventional computer.

The fact that biological computation is so effective at tasks such as recognising objects, storing/retrieving vast amounts of varied *knowledge,* suggests that it may be possible to attain similar capabilities in artificial devices based on the design principles of the brain or neural systems. These simple models have only a metaphorical resemblance to nature's computers but they offer an elegant and different way of thinking about information processing. This approach is now inspiring new computer designs and techniques.

2.3.1. Neural Networks

Current research in Artificial Neural Networks (ANN) builds on a long history of efforts to capture the principles of biological computation within mathematical models. The effort began with pioneering investigations of neurons as logical devices by Professor Warren McCulloch and Dr Walter Pitts in 1943 [93]. The field of Cybernetics was created by Norbert Wiener in 1947 and in 1957 Frank Rossenblat of Cornell University devised the Perceptron — a simple two layer vision system using a matrix of photocells as a retina. In 1969, Minsky and Papert examined the notion of building thinking machines using perceptrons to mimic human neurons. They showed major inadequacies of the Perceptron [73]. The glaring fault that Minsky and Papert had identified in the Perceptron was that there was no way to create an *Exclusive OR* function. Without an Exclusive OR function, complex pattern recognition was not possible. This virtually stopped research into neural networks. The return from the wilderness was heralded by a paper in 1982 by John Hopfield [49] which illustrated that the neural network approach contained interesting non-linear properties. This era also produced the solution to the Minsky and Papert critique of the Perceptron, i.e. an Exclusive OR could be performed by introducing at least one "Hidden" layer of neurons between the input and output layers. Several new learning rules were also established to cope with the hidden layers [54, 46, 55].

The 1980's and early 90's have seen an extraordinary growth of interest in neural models and their computational properties. Many individual factors have converged to bring about this growth of interest: neurobiologists were gaining more understanding of how information is processed in the brain; cheap computer power made it possible to analyse the various neural models and there was growing interest in parallel computation and analog VLSI, which lend themselves to implementation of neural networks.

Although neural networks (also referred to as *a type of Connectionist Model* [60]) may be different in detail they are all a variations of the Parallel Distributed Processing Framework [91, 93]. The aim of most neural networks is to find a distribution of weights,

over the connections, such that presenting a certain input or part of an input pattern will result in an output which is related to the input in some useful fashion.

2.3.2. Neural Network Structure

As is well-know, all neural networks are composed of a structure of interconnected primitive Processing Elements (PE). In most neural networks, processing elements are arranged into layers with each PE in one layer having connections to each PE in the next layer (sometimes PE's in the same layer are connected). Associated with each interconnection is a *weight* and with each processing element is a *state* (normally on or off but can have a value). The combination of weights and states represents the *knowledge* of the network. Figure 2.3 shows a processing element operating as a simple threshold device.

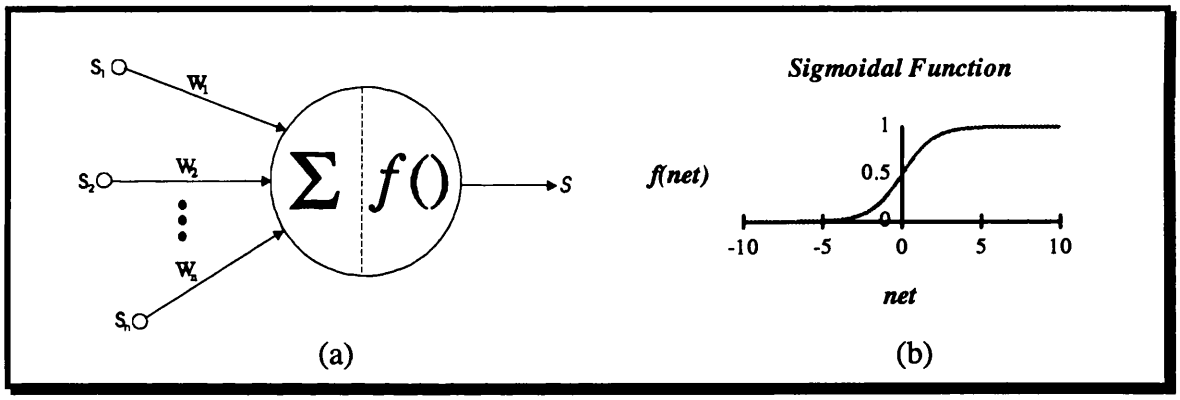


Figure 2.3 — A Neural Network Processing Element.

The output of the PE is dependent on the sum of the Inputs (S_i) and the input connection weights (W_i). A neural network "learns" information by modifying the connections between processing elements. This is analogous to synaptic weight changing in the brain. Neural network models have received a vast amount of research, which has spawned various classes of models, each with their own strengths and weaknesses [79]. The fundamental characteristics of various neural network models are shown in Table 2.4. These models differ on several aspects: *network topology*, *recall phase*, and *learning phase*.

Network topology ranges from single layer, feed-forward models, such as the Perceptron [73], through single layer with feedback connections, as the Hopfield model [49], to more complex interconnection patterns, found in the multi-layer networks with back propagation of errors [92], and also two-dimensional grid of neurons of the Self-Organising Map [59].

The *recall phase* f_1, formed by the *propagation rule* and *activation function*, presents some variations among these neural models. The propagation rule basically calculates the weighted sum of the input states. The activation function is usually restricted to either a threshold function, pseudo-linear function, or sigmoid families function.

The *learning phase* (f_2, f_3) differs considerably among the models. In algorithms such as Hopfield/Kohonen's associative memories [49, 59], the learning phase does not involve any *error calculation* (shown in Table 2.4 as ✖), but generally most models depend upon an error calculation. They can be as simple as the difference between a target value and the neuron's output state (Perceptron [73] and Delta Rule [54]), or can involve more complex computations, such as the Back Propagation model [92] in which the error calculation requires the derivative of the activation function ($T'(net)$).

Neural Network Model	Network Topology	Range of input values	Recall/Learning Phase			
			f_1 - Recall Phase		Learning Phase	
			Propagation Rule	Activation Function	f_2 - Weight Updating	f_3 - Error Calculation
Hopfield/ Kohonen	single-layer with feedback	binary	$net = \sum s.w$	hard limiter	$\Delta w_{ij} = s_i.s_j$	✖
Perceptron	single-layer feed-forward	binary or continuous	$net = \sum s.w$	hard limiter	$\Delta w_{ij} = \eta.s_i.\varepsilon_j$	$e_j = t_j - s_j$
Widrow-Hoff (Delta Rule)	single-layer feed-forward	continuous	$net = \sum s.w$	linear	$\Delta w_{ij} = \eta.s_i.\varepsilon_j$	$e_j = t_j - s_j$
Back Propagation	multi-layer bidirectional links	continuous	$net = \sum s.w$	sigmoid	$\Delta w_{ij} = \eta.s_i.\varepsilon_j$	$e_{j_o} = T'(net).(t_j - s_j)$ $e_{j_h} = T'(net).\sum E.W$
Boltzman Machine	multi-layer or randomly connected	binary	$net = \sum s.w$	sigmoid	$\Delta w_{ij} = \eta.\varepsilon_j$	$e_j = \eta(\langle p_{ij}\rangle - \langle p'_{ij}\rangle)$
Counter Propagation	multi-layer feed-forward	binary	$net = \sum s.w$	hard limiter	$\Delta w_{ij_1} = -\eta.e_{j_1}$ $\Delta w_{ij_2} = -\eta e_{j_2}$	$e_{j1} = s_i - w_{ij1}$ $e_{j2} = \eta_1.t_j - \eta_2.w_{ij2}$
Self Organising Map	2-dimensional grid of output PEs	continuous	$net = \sum s.w$	sigmoid	$\Delta w_{ij} = \eta.\varepsilon_j$	$e_j = s_i - w_{ij}$

Table 2.4 — Characteristics of Some Popular Neural Network Models.

2.3.3. Development Cycle

"Programming" a neural network consists of specifying the *Activation and Transfer* functions [117, 79] for the processing elements and their interconnections, followed by the *training* of the network with input patterns. The full neural network development cycle can be seen in Figure 2.4.

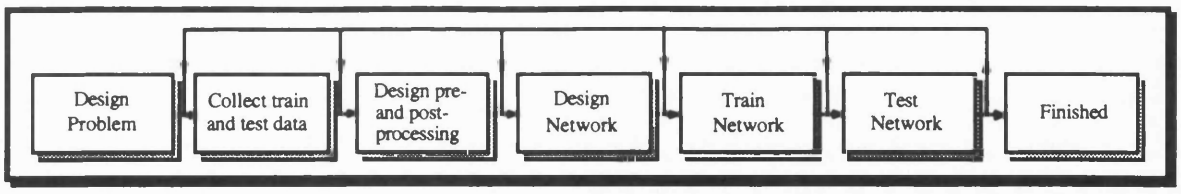


Figure 2.4 — Neural network development cycle.

Learning is the most important attribute of a neural network. During learning, input information is presented repeatedly and then the neural network produces an output and adjusts the pattern of weights to capture the underlying pattern of the input set. Learning can be classified into *supervised* and *unsupervised* learning. In *supervised* learning, input patterns plus the expected output patterns are repeatedly presented and an *error* is produced for the difference between the desired and actual pattern. This error is used to adjust the corresponding connection weights as in the formula above. In *unsupervised* learning there is no need for input and expected output pairs, all that is required is enough sample inputs that the network will self-organise to reflect higher-order regularities within the input set. There also is a middle ground of *graded learning* in which the network is not given desired output for a input pattern but instead a performance score is given for the closeness of actual and desired values.

2.3.4. Applications

Neural Networks at present offer numerous basic-research possibilities and have found commercial and industrial applications within narrowly constrained tasks (mainly pattern recognition and generalisation systems [51]). Current neural network packages [112] exist that use either all software (neural network simulation packages run on conventional hardware), all hardware (special hardware [4, 82, 81]) or a combination of both. Some of the current applications include :

- **Military/Aerospace:** Aircraft pilot/navigation systems, radar/sonar/image processing, expert air traffic control, and autonomous vehicles.
- **Communications:** Voice and image recognition/compression, real-time language translation, continuous speech recognition, and network management.
- **Manufacturing**: Robot vision, adaptive robot movement [42], VLSI chip design, and processing control.
- **Banking and Finance**: Loan evaluation, consumer credit scoring, cheque and signature scanning, and predictive analysis of stocks and bonds [24].
- **Medicine**: Expert diagnosis systems, DNA code prediction, and medical image interpretation.

- **Law Enforcement**: Finger print identification, and face recognition.

All the above applications illustrate that neural networks have proven their strengths at prediction, classification and pattern recognition tasks.

2.3.5. Programming Environments

Programming environments range from commercial products from both established and start-up companies, to public-domain software, available free from university research groups. However all of these programming environments share many common features such as:

- an ***algorithm library*** of common, parameterised neural network algorithms, such as Back Propagation, Hopfield, Boltzmann etc.;
- a ***high-level language***, often object-oriented, for programming or customising an algorithm or application;
- a ***network specification language***, a low-level, machine-independent, language (often based on C) defining the neural network simulation;
- a ***graphic interface*** with menus and a command language: for configuring a neural network, then controlling and monitoring its execution;
- ***translators*** for mapping the network specification language to various target machines.

As shown in Figure 2.5, neural programming environments can be categorised with respect to the target user group, the algorithm and the application requirements [113].

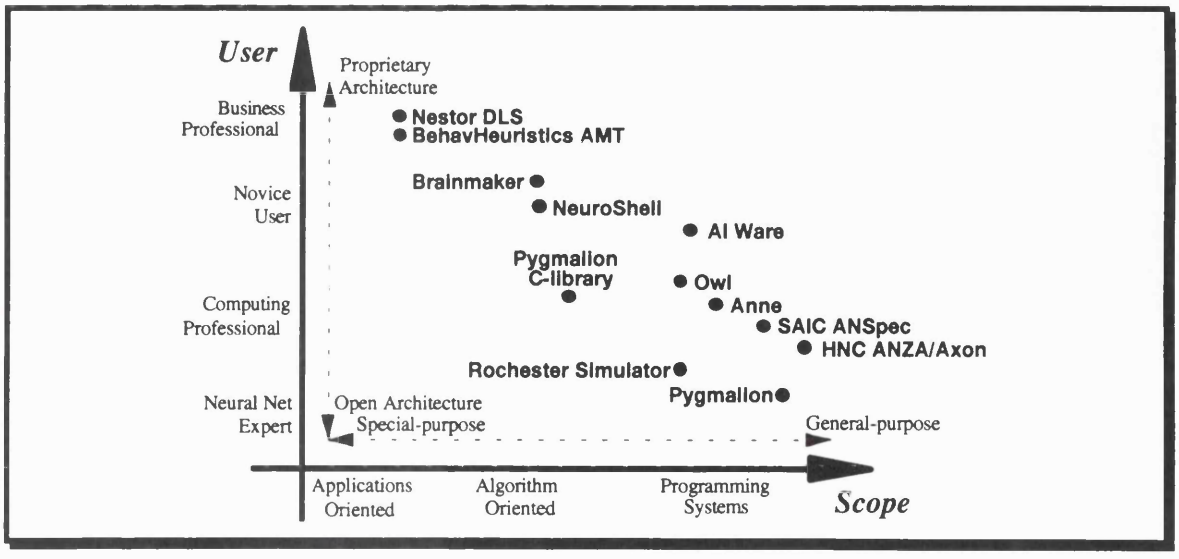


Figure 2.5 — Range of Neural Network Programming Environments.

Appendix B summaries some the most popular neural programming environments within these categories.

2.4. Computational Properties

From the above survey, it can be seen that expert systems are fundamentally rigid, serial and discrete computing process, while neural networks are inherently flexible, parallel and distributed in their knowledge representation and decision making strategies. The following section highlights the individual strengths and weaknesses of these techniques.

2.4.1. Strengths and Weakness of Expert Systems

Expert systems have a number of attractive features that stem directly from the ability to represent domain expertise in an explicit representation:

- **Permanence** - The construction of the expert system's knowledge base from key experts from a company can provide an additional feature of an *institutional memory.* This allows the companies expertise to be collected and therefore made permanent. Unlike human experts who may retire, resign or expire, the expert system's knowledge will last indefinitely.
- **Multiple Expertise** - the knowledge of multiple experts can be made available to work simultaneously and continuously on a problem at any time of the day.
- **Explanation** - the expert system can explicitly explain in detail the reasoning that led to the conclusion. This increases the confidence that the correct decision was made. A human may be unwilling or unable to do this all the time.
- **Steady, unemotional and complete response at all times** - this may be important in real-time and emergency situations when a human expert may not operate at peak efficiency because of stress or fatigue.
- **Increased Availability** - expertise is available on any suitable computer hardware i.e. mass produced expertise. The flexibility of the incremental development cycle of expert systems allows them to grow with the needs of the business or institution.
- **Reduced Cost** - the cost of providing expertise per user is greatly lowered.
- **Reduced Danger** - expert systems can be used in environments that might be hazardous for a human.

- **Intelligent tutor** - the expert system may act as an intelligent tutor by letting the student run sample programs and explaining the systems reasoning.

Current expert systems have various limitations, many of which will gradually disappear as researchers advance the state of the art. Some of these problems are inherent, brought about by the *rigidity* of symbolic processing techniques [84]. The following list highlights some of the limitations of current expert systems :

- **Representation of temporal or spatial knowledge is poor.** This type of knowledge requires large amounts of memory to keep track of the state of the system at various points in time or spatial relations of objects.
- In some areas, **experts have difficulty in translating their expertise to an explicit simple representation.** They describe their actions as being fuzzy and intuitive.
- **Expert systems have a very narrow domain of expertise and hence their operation is not robust.** When pushed beyond their limits or given problems different from those for which they were designed, expert systems can fail badly.
- **Expert systems have difficulty in dealing with erroneous or inconsistent knowledge.** This can occur if the knowledge base has errors or inconsistencies, perhaps from multiple experts who may disagree on a particular aspect of the domain knowledge or if the knowledge base becomes large and complex.
- Expert systems have a serious drawback in that they **can not automatically acquire knowledge from their domain** (i.e. have no mechanism to learn from domain data). Knowledge acquisition is a major bottleneck in expert system development. It can be tedious, time-consuming and often an expensive, but necessary activity. Despite some interesting research into Rule-induction and automatic knowledge acquisition techniques, this problem still exists.
- **It can be difficult to understand, refine and correct large knowledge bases.** For example, PUFF [2] an expert system that interprets data from pulmonary functions test, had to have its number of rules increased from 100 to 400 just to get 10 percent increase in performance.
- **There is no general technique for logically analysing the system's completeness and consistency.** This is why knowledge engineers often only use one or two sources of expertise, so that inconsistencies can be reduced.
- **Due to these problems, development times can be long.** The development time tends to increase significantly the more complex the problem and as the

size of the knowledge base increases. For example PUFF, the system that interprets lung function test data, required 5 person-years of effort; XCON, a system that configures computers [66], took about eight person-years to reach reasonable performance configuring VAX 11/780 computers.

2.4.2. Strengths and Weaknesses of Neural Networks

Due to the architecture and the kind of information processing that is performed within neural networks (i.e. *distributed, dynamic, and parallel*) [27], the attributes that arise from this architectural design have given neural networks their adaptive characteristics but have also contributed to some of their limitations.

Parallel distributed processing is probably the most important attribute of neural networks. The internal representation of information is stored in a *distributed* nature. In contrast, *local representation* such as semantic networks [11] each node represents a single concept. For example if a semantic network is accessing the concept of "cat" then the node for the cat concept is activated and all other nodes remain inactive. In a *distributed* neural network, nodes have no simple meaning, but instead the individual concepts are represented as a pattern of activity of nodes across the network. The implications of this Parallel distributed processing approach are significant. They include :-

- **Content-Addressability** in which the actual contents of memory, not the memory address is used to directly access the information. In the Parallel Distributed Processing approach, a pattern of activity can be retrieved quickly from a network trained on a large input set of information.
- **Association and Generalisation** acts upon the pattern of activity for a concept. Concepts that are closely related have similar internal representations, so properties of similar concepts may be associated together. Generalisation utilises association, by allowing multiple associated patterns to select a corresponding output, as with "best-fit" or "nearest neighbour" answers.
- **Pattern Completion and Pattern Discovery** are attributes resulting from generalisation and association. Just as the brain is very good at recognising a familiar face from a fuzzy photograph, so neural networks can grasp patterns from partial input data. Neural networks can be used to generate accurate predictive models for problems involving huge data sets and numerous variables via pattern discovery within the data (e.g. Financial Forecasting [120]).
- **Fault Tolerance and Graceful Degradation** is the ability to function when nodes have been lost. This ability is exhibited by adult brains which lose many cells every day with only a very gradual decline in capabilities. Even if nodes are lost a concept

does not disappear. It merely becomes less reliable. Neural networks have functioned when as many as 15 % of their units are damaged.

- **Self-organisation** in neural networks develops internal representations of the environment via the exposure to data. Similar to the brains own "cognitive codes", these *feature detectors* form unique interactions that can distinguish and identify input patterns.
- **Spontaneous Learning** is resultant of self-organisation. With conventional computers it is necessary to formulate the algorithms/rules to perform information processing. For many real-world problems, deriving an explicit formalisation is difficult and expensive, or even impossible. However, neural networks do not require explicit programming of conditions or rules. They learn via training on an input/output set and use the above mentioned attributes to solve problems that have been difficult to formulate or not solvable via other techniques e.g. Travelling Salesman Problem [14].
- **Autonomous Acquisition of Knowledge** can be obtained from a input data set and used to build the functional equivalents of an expert system. Neural networks can "discover" correlations between the input set and the actual outcome required, and so generate an "expert" opinion using hypothesis testing and probability estimation [94]. Neural Networks can also be trained incrementally — as more examples are provided, they can be simply added to the training set.

Unfortunately, also due to the parallel distributed nature of their construction, neural networks have difficulties:

- when dealing with problems that demand step-wise sequential processing.
- when precise solutions are required, for example when monitoring equipment behaviour.
- when an explanation is required to justify the conclusion that has been reached. There is no easy understanding path for neural networks. It is difficult to extract the internal knowledge of neural networks, for example in terms of rules, although some research is currently investigating possible strategies [21, 67].
- they also require a sufficient number of sample cases to train the neural network in the patterns relevant to the particular problem.
- when a precise, repeatable result is required. There is a an element of randomness in neural network results, due to the random weight initialisation process. There is no guarantee that an identical result will be obtained from two

neural networks even if trying to explain the identical concepts from identical training sets.

- when attempting to construct the best network architecture. A neural network's rate of convergence is controlled by several empirical parameters. The selection of these parameters, although the subject of much current research, is still a matter of trial and error. These parameters are critical to the speed with which training proceeds. Generally speaking, neural networks complexity grows exponentially with the size of the input vector (the importance of this disadvantage is being diminished with recent advances in VLSI technology and Optical Computing [112, 99]).

2.5. Summary

By examining the basic knowledge structures, development cycle and applications, for expert systems and neural networks, this chapter has highlighted the contrasting *but* complementary nature of these approaches.

Because experts systems manipulate symbolic information via logical inference they can produce precise and repeatable results, which can be justified by the system. This rigid background also means that expert systems cannot cope with partial or uncertain information and must be programmed by encoding knowledge from a domain expert. This knowledge elicitation process can be time consuming, and if the resulting knowledge base is large the actual operating speed of the system can be slow.

However, due to the parallel distributed processing structures found within neural networks, they are imprecise, and have difficulty in justifying their results. But because of this structure they are capable of learning from domain examples or raw data to form an internal model of the problem. The ability to learn about the domain means that neural networks have a short development time in comparison to expert systems and due to their parallel distributed structures they are faster to run.

As these techniques are being applied to real-world problems, there is a growing realisation that a single technique approach to complex problems is insufficient. Therefore, as both symbolic and adaptive techniques have complementary strengths and weaknesses, the synergy of the two approaches would help to overcome some of their individual limitations.

Chapter 3

Hybrid Systems: Issues and Classification

This chapter outlines the important arguments for hybrid systems and presents a new classification scheme that categorises hybrid systems with respect to functionality, processing architecture and communication requirements. This classification scheme is used to explore various hybrid methods for integrating expert systems and neural networks. Finally, possible problems with hybrid systems are outlined and a development cycle for their construction is proposed.

3.1. Introduction

Many proponents of symbolic processing and adaptive processing strongly advocate that *their* approach alone is best for the modelling of cognition and for solving complex real-world problems. As shown in chapter two, a close examination of this symbolic versus adaptive debate reveals that both approaches have complementary strengths and weaknesses. By integrating these techniques into hybrid systems, they can overcome their individual weaknesses and capitalise on their strengths. In support of this hybrid strategy, it is also becoming evident that most complex real-world problems are difficult to solve using either symbolic or adaptive processing in isolation, but rather they require the synergy of these two complementary approaches.

With this interest in hybrid systems, it is essential to establish the fundamental reasons for why hybrid systems are required, and to have a scheme for comparing different types of hybrid systems. To meet these goals, this chapter identifies the primary need for hybrid systems, introduces a new classification scheme to compare hybrid systems and presents a development cycle for building hybrid systems. This thesis research then builds upon this foundation by advocating an integration mechanism, implementing an environment (ORBERON) and finally by outlining the design for a commercial framework (Esprit III HANSA project) for building hybrid systems.

3.2. The Need for Hybrid Systems

From the previous discussions regarding the complementary strengths and weaknesses of symbolic and adaptive processing and the requirements of real-world problem-solving, it is evident that there are three main reasons for creating hybrid systems: *Technique Enhancement*, the *Multiplicity of Application Tasks* and *Realising Multi-functionality*.

(1) *Technique Enhancement*: This is the integration of different techniques to overcome the limitations of a particular technique. Here the aim is to take a technique that has weaknesses in a particular property and combine it with a technique that has strengths in that same property. For example, expert system's use a pattern matcher for matching variables in rules. This capability could be enhanced by using neural networks for the task because they are naturally efficient at pattern matching. Therefore, an enhanced system would be a hybrid that combined the expert system with a neural network that performed the pattern matching task. Recently there have been several examples of such expert systems enhanced with neural networks [108, 109].

(2) *Multiplicity of Application Tasks*: In this instance a hybrid system is created because no single technique is applicable to the many sub-problems that a given application may have. Most real-world domains have both "logical", static components (that can be easily handled by, say, expert systems) and "fuzzy", dynamic, poorly understood components (which can be handled by, say, neural networks).

(3) *Realising Multi-functionality*: The motivation in this instance is to create hybrids that can exhibit multiple information processing capabilities within one architecture. In other words these systems *functionally mimic* or emulate different processing techniques. The prime example of such systems is the use of a neural network for performing symbolic processing [3, 48]. In support of such systems, the human brain performs high-level cognitive tasks such as planning and navigation, on an adaptive distributed neural architecture. This would seem to suggest that nature has found a mechanism to combine the properties of symbolic processing and adaptive processing. Therefore, developing hybrid models could help to provide a better understanding of how to model human cognitive processing.

3.3. A Novel Hybrid Classification Scheme

The need for hybrid systems has led to many different interpretations of what exactly a hybrid system is within the context of symbolic and adaptive techniques. By examining the characteristics of existing hybrid systems, this confusion of terminology can

be resolved by categorising these systems with respect to important factors such as their functionality, processing architecture and communication requirements. A novel classification scheme for categorising hybrid systems has been developed that takes these factors into account and therefore helps clarify the confusion that exists in this emerging field. This scheme also allows qualitative evaluation of existing hybrid systems and provides a frame of reference to compare different hybrid systems. Since this work was first reported [36], several researchers have adopted it to describe their own research [63, 97, 96].

Figure 3.1. outlines the classification scheme which takes into account factors such as functionality, processing architecture and communication requirements. The classification scheme consists of three classes that characterise possible hybrid systems. The classes define hybrid systems as *Intercommunicating, Function-replacing* and *Polymorphic* hybrids. The next sections describe these different categories of hybrid systems and examines typical examples of these hybrid systems.

Within the next sections, discussions will be limited to hybrids of symbolic and adaptive techniques, although the hybrid classification scheme is also applicable to hybrids of other types of processing techniques (e.g. statistical clustering, regression techniques etc.).

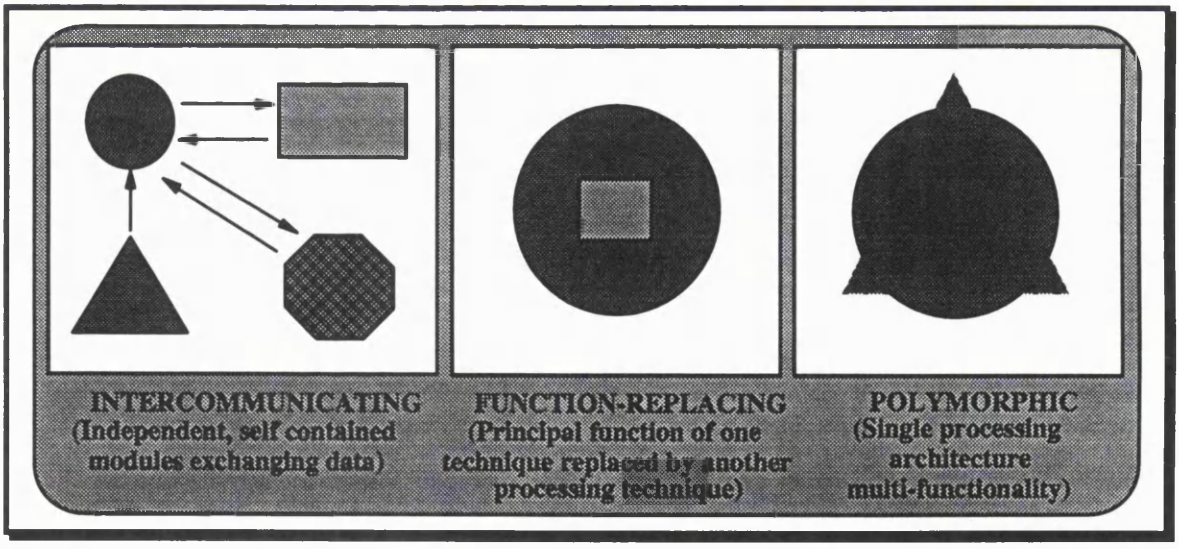


Figure 3.1 — Three Proposed Hybrid Classes.

3.3.1. Intercommunicating Hybrids

Intercommunicating Hybrids are *independent,* self contained, processing modules that exchange information and perform separate functions to generate intelligent solutions.

If a problem can be subdivided into distinct processing tasks, then these different independent modules can be used to the solve the parts of the problem that they are best at. These independent modules which collectively solve the given task are co-ordinated by a *control mechanism.*

For example, if a particular task has subtasks for pattern recognition, serial reasoning and optimisation, then a neural network, an expert system and an adaptive optimisation technique such as a genetic algorithm, can be made to perform these respective tasks. These independent modules can process subtasks either in a sequential manner (where the control can be provided in the rules of the expert system) or in a competitive-cooperative framework such as a blackboard problem solving architecture [17] (where the control is in a distinct control component).

A classic example of an *intercommunicating hybrid* is Schreinemakers and Touretzky's [98] system for the diagnosis of Mastitis. In their system, a rule-based expert system performs serial inferences, and calls neural networks to find patterns in the data as part of the diagnosis. The system uses a set of OPS5 data structures called Working Memory Elements for communication between the neural network and the expert system.

The expert system also acts as a "knowledge manager" to provide the neural network with new training examples in the cases where the diagnosis is inaccurate. The authors claim that the system archives a classification rate of 87%, which is comparable to the performance of an expert veterinarian.

Further examples of this type of hybrid are given by Tirri [108] who describes another neural network-expert system hybrid that is used for respiratory and anaesthesia monitoring and by Dunker et al [23] who describe a co-operative environment using blackboard design [17] for integrating neural networks and knowledge based systems.

3.3.2. Function-Replacing Hybrids

Function-Replacing Hybrids address the functional composition of a *single* processing technique. In this hybrid class, a *principal function* of the given technique is replaced by *another* processing technique. Typically, in these systems, the "higher" level technique directly calls the "lower" level replacement module when carrying out the original task.

The motivation for developing this type of hybrid system is for the *Improvement of individual techniques,* as was discussed earlier. The improvement gained by the technique that is replacing its *principal functions* could either be an increase in execution speed or

enhance reliability. Examples of *principal functions* include pattern matching in an expert system, weight changing in a neural network and crossover operations in a genetic algorithm.

Montana and Davis [75] provide an example of a *function-replacing hybrid.* They replace the back propagation weight-changing mechanism of a neural network with genetic algorithm operators. The genetic algorithm takes the existing weights of the neural network and then applies mutation and crossover operators on these weight values to obtain better set of weights. This hybrid has been able to outperform the commonly used back propagation neural network by taking a much smaller number of iterations to converge to a good solution.

3.3.3. Polymorphic Hybrids

Polymorphic Hybrids are systems that use a *single processing architecture* to achieve the functionality of *different* processing techniques. That is, these systems can *functionally mimic* or emulate different processing techniques. These might be viewed as being "chameleon-like" systems which can change their functional form.

Examples of polymorphic hybrids are neural networks that attempt to perform symbolic tasks such as step-wise inferencing and also neural networks which function as if they were doing genetic search.

Ajjanagadde and Shastri [3] demonstrate a *polymorphic hybrid system* where a neural network achieves the functionality of a symbolic reasoning system. This system addresses the symbolic variable binding problem (how to dynamically represent variables and their role bindings) by propagating *rhythmic* patterns of activity wherein dynamic bindings are represented as *in-phase* firings of appropriate nodes in the network. This allows a rule-like chain of inference and reasoning to be present within a neural architecture. The motivation for this system and many other *Connectionist Expert Systems* (described later in this chapter) is to demonstrate the cognitive principles of how symbolic processing can be archived on a connectionist architecture.

Another example of a *polymorphic hybrid* is Ackley's [1] connectionist network which is functionally equivalent to a genetic algorithm.

3.4. Neural Network and Expert System Hybrid Models

As previously discussed, neural networks and expert systems have limitations when they operate outside their areas of competency [27, 84] and that the integrators viewpoint perhaps represents the best utilisation of a neural network's bottom-up approach and an

expert system's top-down approach. Therefore this section uses the three reasons for creating hybrid system (*Technique Enhancement*, the *Multiplicity of Application Tasks* and *Realising Multi-functionality*) and the hybrid classification scheme to explore what functional characteristics a hybrid system could exhibit if neural networks and expert systems were integrated.

Intercommunicating Hybrids

Intercommunicating hybrids built from neural networks and expert systems represent the majority of systems that have been produced either in research or commercially. This is because they represent the easiest type of hybrid to build, and one which provides immediate hybrid systems within which:

1. ***Neural networks could pre-process application data*** into a form that could be effectively handled by the expert system (e.g. neural networks could generalise or classify input data). Figure 3.2 shows firstly, how a neural network conditions input data from an industrial plant via sensors and passes the results to an expert system which then sends control signals back to the industrial plant. A similar *intercommunicating* hybrid system is used to control temperature at in the Furnaces at the Japanese Kobe Steel Plant [125]. The second example shows a hybrid system that was proposed by the London Underground to check when platforms and escalators are overcrowded. Neural networks would classify video camera images of the platforms and escalators, and pass the results to an expert system that determines if an operator should be alerted to possible overcrowding.

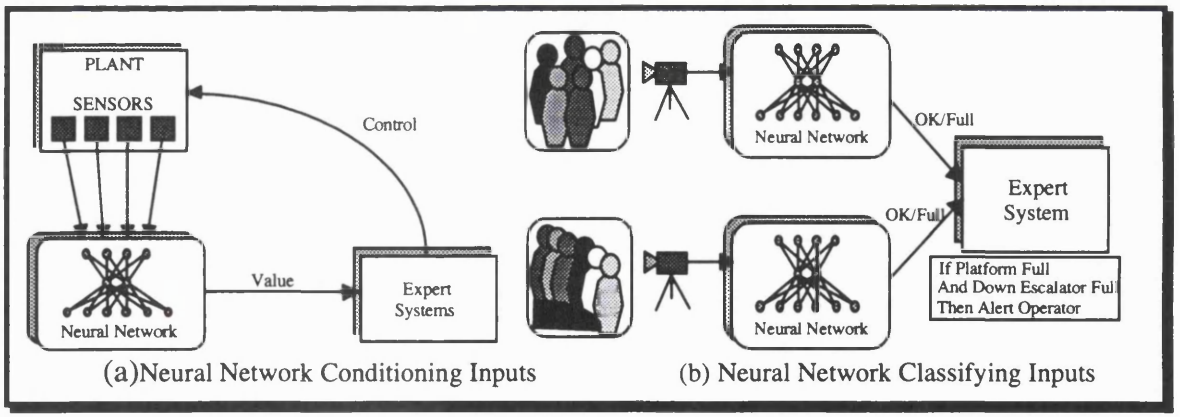


Figure 3.2 — Neural Network Conditioning input for Expert System.

2. ***An expert system controls information/knowledge flow through several networks.*** As Figure 3.3 shows, an expert system could dispatch inputs to trained networks and receive results back for processing and then dispatch processed information to another network. The expert system adds the sequentiality and control for multi-network

problems. It could also provide an explanation facility for conclusions that take into account the inputs and outputs of the neural networks [87].

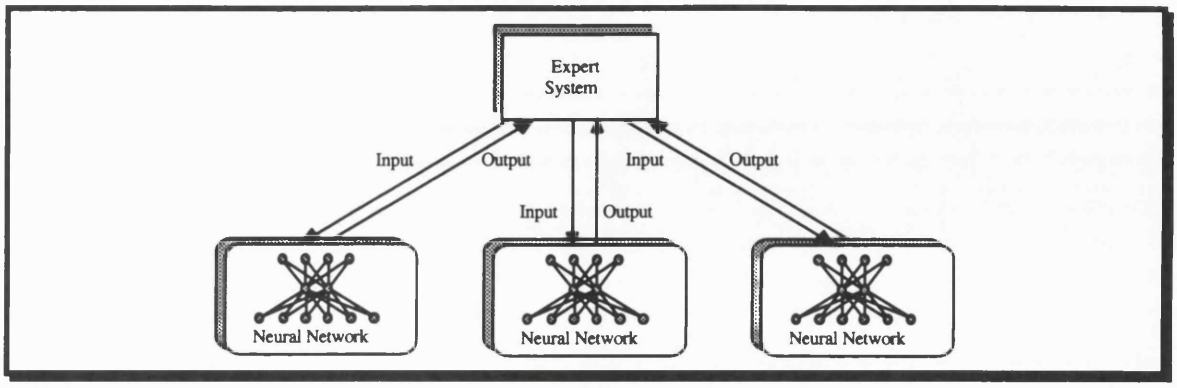


Figure 3.3 — Expert System controlling Neural Networks.

3. ***An expert system can be used to train the neural networks*** by supplying the training data from processed information.

4. ***An expert system can provide a convenient user interface*** for the particular application area.

5. ***The expert system and the neural network can work together to improve the results of the system*** [7]. As shown in Figure 3.4, this analysis coupled with newly inferred information, can result in new training patterns for the neural networks and so improve the overall performance of the system.

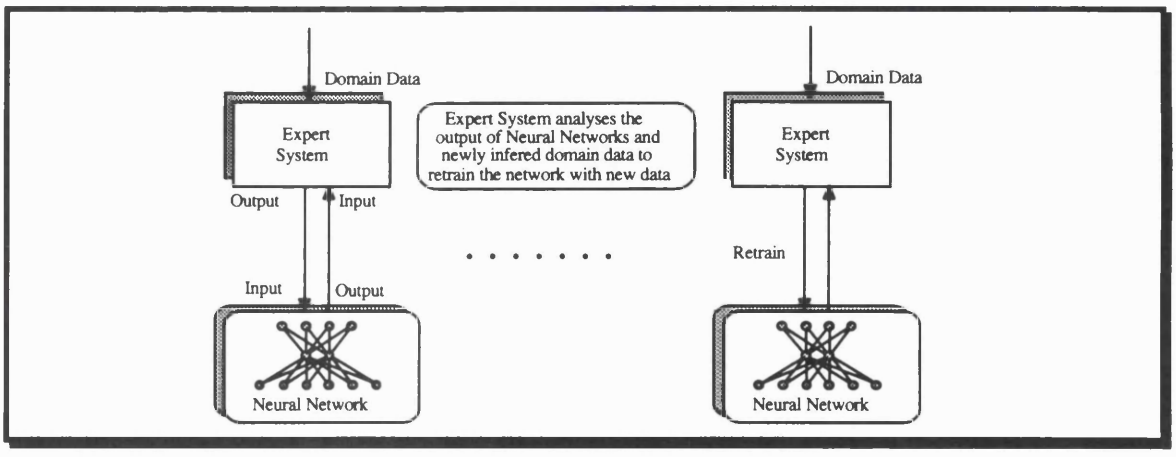


Figure 3.4 — Expert System Retraining a Neural Network.

Function-replacing Hybrids

This class of hybrid system could offer systems where neural networks can functionally replace principal components within an expert system. The basic cycle of an

expert system consists of pattern matching the state of the system with the rulebase, then selecting one of the matching rules and executing it. Figure 3.5, shows how this *function-replacing* type of hybrid could replace the pattern matcher and the selector of rules with two trained networks. This process would substantially improve execution time due to the inherent parallelism and pattern matching capabilities of neural networks.

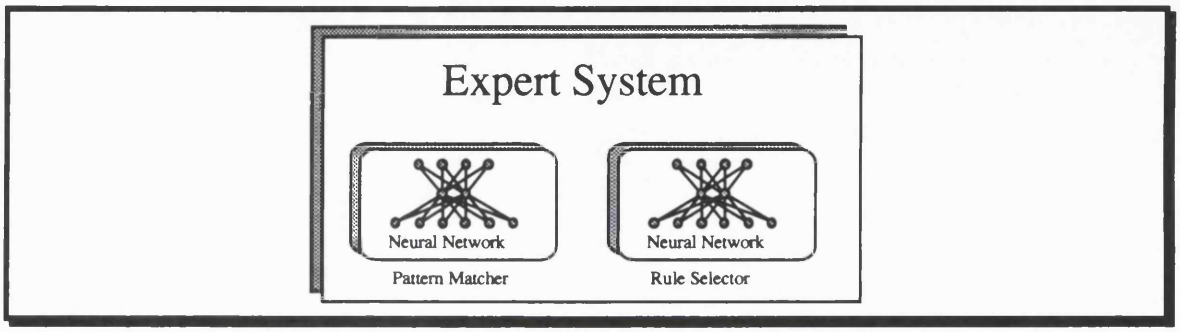


Figure 3.5 — Neural Networks Embedded in a Expert System.

Polymorphic Hybrids

Connectionist expert systems represent the *polymorphic* class of hybrid systems and possibly form the most interesting research area both from an engineering and cognitive science perspective. They attempt the theoretical assimilation of symbolic processing especially rule-based processing within connectionist networks. Connectionist expert systems have replicated in the structure and the functionality of connectionist networks, some of the components, issues and operations of expert systems. These include:

- Knowledge representation - rules, frames and other knowledge structures.
- A logical inference mechanism.
- Forward chained inference and backward chaining/question generation.
- Explanation of conclusions via **if-then** rules.
- Addition and deletion of rules from the current facts known by the system (i.e. the maintenance of the systems working memory).
- Negated expressions and partial matching of rules.
- Binding values to variables.

Table 3.1 shows recent systems that have performed these tasks with varying degrees of success. Many systems contain a fair percentage of the these objectives but no one system possesses the ability to perform all these desirable operations. Tariq Samad's RUBICON system [95], a connectionist rule-based system incorporated many of the

objectives including a variable number of expressions in the left and right hand sides of each rule, chained inference, working memory maintenance, negated expressions and partial matching via a multi-layered local and distributed architecture. RUBICON lacked variable binding and certainty measures. Other recent systems include the "Distributed Connectionist Production System" developed by Touretzky and Hinton [110], which is a complex system of many thousands of units arranged in five "spaces" and requires a limitation of exactly two expressions in rule antecedents. PLATO/ARISTOTLE [115] cannot perform chained inferences and has limitations in rule construct of one consequent per rule and no negation. The "Connectionist Expert System" [31] uses a connectionist knowledge base built from a Linear Discriminant Network but a separate inferencing procedure (utilising the Pocket Algorithm) called MACIE and has no concept of working memory. MACIE does possess a Backward chaining inference mechanisms that allows question generation when no conclusions can be successfully reached.

Connectionist Expert Systems								
Systems	Inference Mechanism	Rule Extraction/ Explanation	Variable Binding	Partial Match	No. of Expressions as Antecedent	Consequent	Local or Distributed	Node Complexity
Connectionist Expert System (Gallant) [31]	Forward & Backward Chaining	Extraction & Explanation		Yes	Mainly Used with Training Set		Local & Distributed	Simple
RUBICON (Samad) [95]	Forward Chaining			Yes	Variable	Variable	Local & Distributed	Simple
Distributed Production System (Touretzky, Hinton) [110]	Forward Chaining		One Variable possible in Rules		2	Variable	Distributed	Simple
Medical Diagnosis Expert System (Saito) [94]	Forward Chaining	Rule Extraction But No Explanation Facility			Mainly Used with Training Set		Distributed	Simple
PLATO / ARISTOTLE (Voevodsky) [115]	Forward Chaining			Yes	Mainly Used with Training Set		Distributed	Simple
FUZZNET (Hall, Romaniuk) [41]		Extraction & Explanation	Virtually Unlimited Variables	Yes	Variable	Variable		Simple Fuzzy Logic
Dynamic Variable binding (Ajjanagadde, Shastri) [3]	Forward Chaining		Virtually Unlimited Variables				Local & Distributed	Simple Phase Sensitive

Table 3.1 — Recent Connectionist Expert Systems.

An important feature of these networks is the ability to extract into an explicit symbolic form (If-then rules) the implicit rules that are learnt by the network. This process is required as a facility to explain the inference path of the systems final decision. The inference engine must have this ability to aid in the understanding of the system. Two recent systems by Saito [94] and Gallant [31] have attempted similar rule extraction systems, although Saito makes no attempt to incorporate the facility into the system as an on-line explanation utility as is done by Gallant.

As most real-world problems have "logical" static components that can be handled by expert systems, as well as "fuzzy", dynamic, poorly understood components, which can be handled by neural networks, there is plenty of scope for integrating these complementary techniques. Table 3.2 lists some recent real-world applications that use hybrid systems built from *intercommunicating* neural networks and expert systems.

Application Type	Application Description	Reference
Analysing financial health of a company	Examines four standard business ratios (quick ratio, debt-to-worth, sales-to-receivables & profit-to-worth). The expert system provides the user interface, obtains the company's financial data, calculates business ratios and searches a database for historic data. Neural networks are called by the expert system to estimate the companies borrowing ability (loan assessment).	Barker [7]
Control system for Underwater vehicle	Uses a blackboard architecture to diagnose and formulate plans for autonomously controlling an underwater vehicle.	Atkins and Deich [6]
Commodity Trading	Uses neural networks to learn patterns of trading, and an expert system is used to manage the investments according to trading rules.	Bergerson and Wunsch [9]
Natural Language	Three level system for analysis of noun phrases. Neural Network provides a learned semantic memory, the local connectionist network integrates semantic and syntactic constraints. Coupled together by an expert system.	Wermter and Lehnert [119]
Robotics	Development of robots capable of learning how to accomplish complex tasks using designer-supplied instructions and self induced practice. Expert system supervises the training of neural networks and controls the operation of the system during the learning.	Handelman and Lane [42]
Optimised Scheduling	SPIKE combines neural networks and evidential reasoning for operational scheduling of astronomical observations with the orbiting NASA/ESA Hubble Space Telescope. Normally creating an optimum long-term schedule for the Hubble Space Telescope is difficult due to the large number of activities, many relative and absolute time constraints, prevailing uncertainites and unusually wide timescales.	Johnston and Adorf [53]

Table 3.2 — Real-world Hybrid Applications.

3.5. Limitations of Hybrid Systems

With the inevitable move towards hybrid systems, a system developer may face a new set of implementation problems when integrating symbolic and adaptive processing techniques for complex problem solving. These problems vary in nature depending on whether the system is an intercommunicating, function replacing or polymorphic hybrid. By examining the operating characteristics of these different types of hybrid systems, one can foresee possible limitations with respect to *communication*, *input requirements*, *development and maintenance times*, and *operational efficiency*.

Intercommunicating Hybrid Systems

Due to the fact that Intercommunicating hybrids are composed of independent techniques that interchange data to generate solutions, they can suffer from operational efficiency problems. Firstly, having two or more independent techniques means having theoretical and operational knowledge about how these different techniques function. Coupled with this technical knowledge problem is the fact that each independent technique will have a different programming interface. Therefore, the developer must learn many programming interfaces. Both these "learning curves" must be overcome by the developer and this will add extra time to the development cycle.

Secondly, there is often a great deal of redundancy in the development and operation of the separate processing techniques. As the component processing techniques solve sub-problems via their own style of processing, and because they can only exchange data but lack direct access to each other's internal data structures, they must develop independent capabilities that may overlap in their data input and internal processing requirements.

As this class of hybrids relies on data exchange to communicate results to other processing techniques, there can be considerable communication overheads if there is a large number of components or high interaction between components. There may also be a need for each technique to develop data translation capabilities to transform data from one technique's representation into a format that it can understand (e.g. a numerical result from a neural network must be placed into an expert system knowledge base as a fact).

Function-replacing Hybrids

Although function-replacing hybrids have benefits of reduced communications overhead and improved runtime performance compared to intercommunicating hybrids, they still have limitations. Due to the fact that the techniques involved are still *independent* but are more tightly coupled means they still may suffer from redundant data gathering and

processing, similar to intercommunicating hybrids. The development and maintenance complexity of these systems may increase due to the *internal interface* between the "high-level" technique and the replacement technique that is substituting some principal function in the "high-level" technique. For example, an expert system that has its pattern matcher replaced by a neural network will need complex functions to translate its rules into a representation that the neural network will be able to understand.

For real-world applications, sufficient verification and validation of performance is extremely important in not only justifying the results but also to ensure the results are correct. These processes will be particularly difficult with these systems due to the tightness of coupling between the techniques being integrated.

Polymorphic Hybrids

Polymorphic systems can benefit from robustness, improved performance and operational capabilities if an adaptive distributed technique such as a neural network is used to achieve the functionality of different processing styles. With the full integration of techniques the complexity of specifying, designing and building systems, is difficult. There is also a distinct lack of experience and development tools available to build this class of hybrid systems.

3.6. A Development Cycle for Hybrid Systems

To make the design and implementation of an advanced computing application easier, it is recommended to adopt a set of guidelines or a development cycle. As described in chapter two, adaptive and symbolic techniques have their own development cycles, such as knowledge elicitation and representation in expert systems and training and evaluation in neural networks. What is required is a similar set of guidelines that developers can adopt and follow when constructing hybrid systems.

Figure 3.6 shows the development cycle for a hybrid system. The main phases of hybrid systems construction are problem analysis, property matching, hybrid category selection, implementation, validation and maintenance

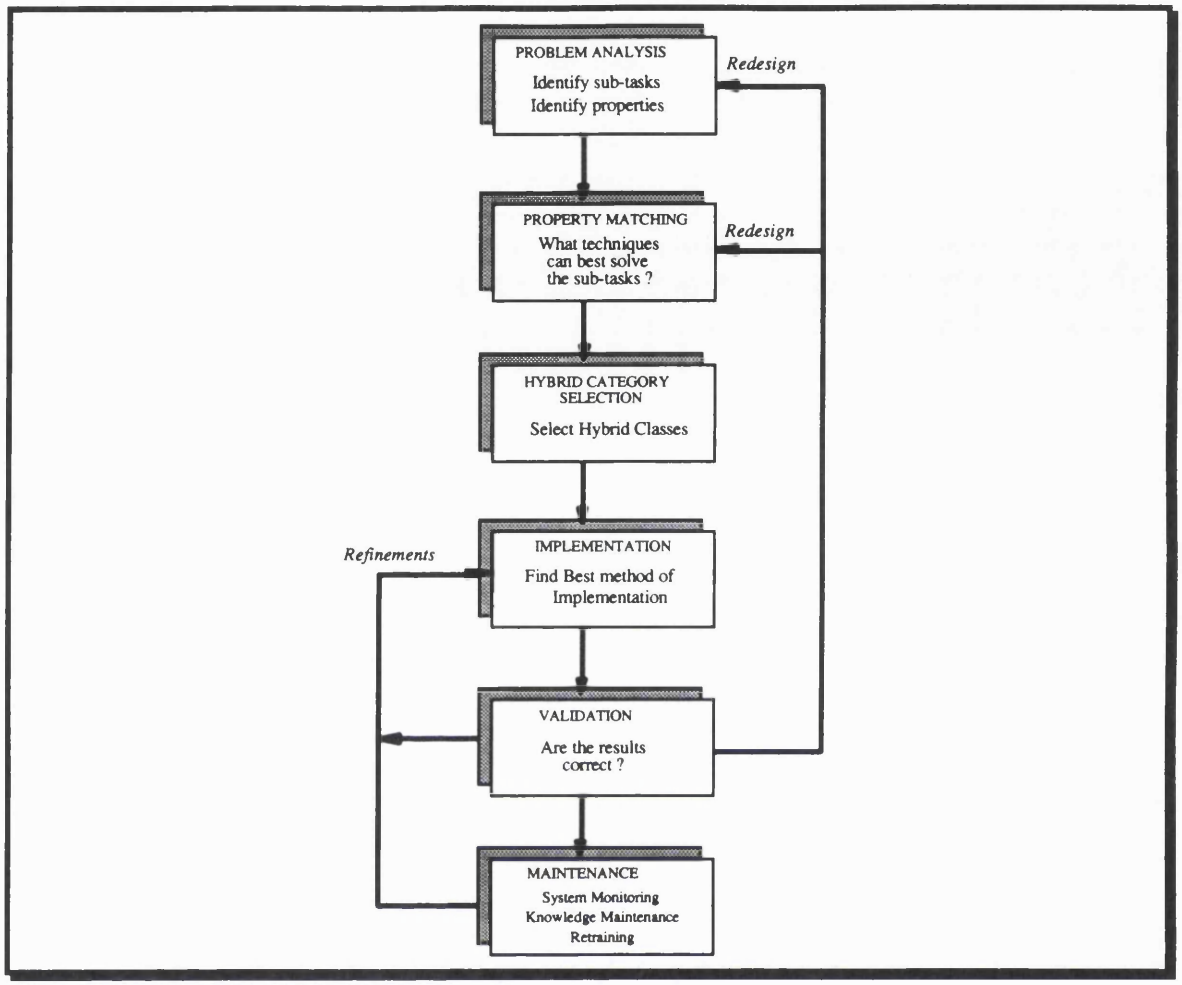


Figure 3.6 — Hybrid System Development Cycle.

The first step in developing intelligent hybrid systems is the ***problem analysis*** stage which involves two distinct steps.

- The first step is to identify any existing subtasks of the problem. A complex problem such as retail distribution, for example, may have both a product demand prediction subtask and a distribution optimisation subtask. However, there may also be problems which do not decompose into subtasks (e.g. a simple prediction task).

- The second step is to identify the properties of the problem. If the problem has subtasks, this involves identifying properties of the subtasks. Examples of the identified properties may include the need for automated knowledge acquisition, coping with brittleness and explanation, etc.

The next step, ***property matching***, involves the matching of properties of available techniques against the requirements of the identified tasks. To assess the computational properties of the available techniques, a *property assessment table* similar to Table 3.3

should be constructed. This type of table provides a rating scheme that grades the competence of each intelligent technique in performing tasks such as learning or providing explanations.

	Properties				
Technologies	**Automated Knowledge Acquisition**	**Coping with Brittleness**	**High-level Reasoning**	**Low-level Reasoning**	**Explanation**
Expert Systems	✓	✓	✓✓✓✓✓	✓	✓✓✓✓✓
Rule Induction	✓✓✓✓	✓✓	✓✓✓	✓✓	✓✓✓
Fuzzy Systems	✓	✓✓✓✓✓	✓✓✓	✓✓✓✓✓	✓✓✓✓
Neural Networks	✓✓✓✓✓	✓✓✓✓✓	✓	✓✓✓✓✓	✓
Genetic Algorithms	✓✓✓✓✓	✓✓✓	✓✓✓	✓✓✓	✓✓✓

Table 3.3 — Property Assessment of Different Intelligent Techniques.

The ***hybrid category selection*** phase selects the type of hybrid system required (Intercommunicating, Function-replacing, or Polymorphic) for solving the problem. This phase uses the results of the previous problem analysis and property matching stages. If the developer finds that there is one intelligent technique which has high ratings in all the desired properties, then there is obviously no need for a hybrid systems solution.

If the problem is composed of distinct, non-overlapping sub-tasks and if there are techniques which have been successfully matched in the previous stage, then an *intercommunicating hybrid* approach can be taken. For example, assume that there is a problem that consists of a prediction task (e.g. energy prediction) followed by a distribution task (e.g. optimisation of energy distribution). Here a neural network, which scores highly in its ability to predict, could be used for the prediction task. This can be followed by a genetic algorithm, which scores highly for its ability to optimise, for the distribution task.

If the problem does not contain distinct subtasks that can be solved using different intelligent techniques, then one has to use a *function-replacing hybrid* approach.

The first step is to list the *candidate techniques* which score highly on the different properties of the problem (obtained from the property matching stage).

Then the task is to find a mechanism to combine the different techniques in a manner so that the resulting hybrid has the desired high ratings. Usually such an improvement can be achieved by replacing a *principal function* of one technique by another technique. For example, the neural networks can be used to induce the decision-making knowledge for the fuzzy system, thus *replacing the function* of a domain expert manually specifying knowledge.

A *Polymorphic hybrid* is appropriate in situations where the desired functionality dynamically changes, this requires the ability to switch from one style of processing to another. For example, in a robot vision task, a Polymorphic hybrid system maybe used to dynamically switch to either a rule-based or pattern-recognition processing style depending on incoming information.

The developer will be now be in a position to start ***implementation*** of the system and will need to select the programming tools and environments needed to implement the hybrid systems. Object-oriented programming offers a natural model for implementing hybrid systems. Object-oriented applications represent the problem domain as a set of objects that interact by passing messages. These messages not only contain data but also the operation to be executed by the receiving object. Because objects have a well defined message interface, they are highly interchangeable. Also, as objects are independent entities that interact through message passing, they are ideally suited for distribution across a network of computers. Using object-oriented programming to represent different processing techniques enables one to mix several processing styles within the same application.

The ***validation*** phase is used to test and verify the functioning of the individual components of the application and the hybrid system as a whole. If any malfunctions are discovered, either at the component level or at the integrated system level, then the whole application should be redesigned or refined depending on the severity of the problem.

The purpose of the ***maintenance*** phase is to periodically evaluate the hybrid system's performance, and to refine it as the need arises. Maintenance is particularly important for adaptive components (e.g. neural networks and genetic algorithms) in rapidly changing environments where performance can easily degrade if they are not continually *re-trained* with recent domain data.

An aspect that is not covered in the development cycle is the involvement of domain expertise. A domain expert can be invaluable if available during the problem analysis and property matching stages. The expert's knowledge of the domain helps to identify the sub-tasks and also assess different intelligent techniques.

The above development cycle is believed to be general enough to be applied to most commercial and industrial hybrid systems. Although the development cycle has concentrated on hybrids of intelligent techniques it can be applied to hybrids of any set of computational techniques.

3.7. Summary

The current interest in hybrid systems has been brought about by the realisation that symbolic and adaptive systems have complementary strengths and weaknesses and because real-world problems are best solved by using purely symbolic or adaptive processing, but instead need the synergy of these approaches. On examining the complementary strengths and weaknesses of symbolic and adaptive techniques a novel classification scheme has been detailed that takes into account the functionality, architectural and communication issues of constructing hybrid systems. This classification provides a mechanism to make qualitative assessment of existing hybrid systems and also helps to guide the development of new hybrid systems. Expert systems and neural networks are used as an example of how these different hybrid approaches can be integrated to solve different types of problems.

In an attempt to devise a methodology for hybrid systems, the proposed hybrid classes are examined with respect to possible implementation problems. To help new designers of hybrid systems, a development cycle and a set of implementation guidelines have been constructed from the experience gained during this research.

Chapter 4

Object-Oriented Integration

This chapter advocates an object-oriented philosophy for building hybrid systems. A basic overview of object-oriented programming is given. This is followed by an investigation into the suitability and main advantages of this approach for constructing hybrid systems.

4.1. Introduction

For intelligent hybrid systems to be fully accepted in the commercial environment they must be able to integrate and communicate with conventional computing systems such as databases and spreadsheets. To "add value" to an organisation's existing decision support systems, intelligent hybrid systems must also be able to extract and use a wide variety of information, as well as disseminating their results to existing applications and systems. For these reasons it is vital that developers have a range of software tools, information exchange techniques and development environments for integrating intelligent systems with conventional computing systems.

To design and implement such hybrid systems, the perceptual features of adaptive processing and the high-level reasoning of symbolic processing must be captured in one system. This can be achieved in one of four ways [45] :

1. By designing a new methodology for getting traditional symbolic processing systems to handle low-level image and signal processing, to handle pattern recognition, and to be adaptive in its operation.
2. By designing a methodology for getting adaptive systems to handle "high-level" symbol-processing tasks in applied domains. This might involve adaptive systems which can manipulate data structures, handle variable binding in long inference chains, deal with the control of inferencing, etc.,
3. By inventing a new processing technique, to become yet another competitor to enter the set of possible models for delivering intelligent behaviour,

4. Or, by taking the current generation of symbolic and adaptive systems, and pragmatically integrating them to produce hybrid systems that exploit the strengths of each processing technique.

While the first three of these are certainly plausible approaches, and all three are currently driving many interesting research projects [74], they require major technological breakthroughs, and much rethinking of existing technologies. The fourth option of building intercommunicating hybrid systems, requires no major technological breakthroughs, but rather a novel mechanism to represent and link the current technologies. This approach, therefore appears to provide the path of least resistance for developing hybrid systems. This type of intercommunicating hybrid development is the most pragmatic way to succeed because it offers immediate hybrid solutions and still allows independent technological development of adaptive and symbolic techniques. This approach also allows existing and any new techniques developed in the future to be integrated easily into a hybrid system.

This class of intercommunicating hybrid system has a direct mapping with the emerging area of Distributed Computing Environments within conventional computing. Such computer systems will consist of heterogeneous, independent and distributed computing resources, such as computers (mainframes and PC's) running information-intensive applications (databases, spreadsheets, etc.). Intercommunicating hybrid systems and this new generation of open computer systems share similar properties in that they :

- are both heterogeneous multi-agent systems: where the agents are different information processing applications (such as neural networks, expert systems etc.) or computing resources.
- both need a mechanism to communicate and pass data between each other.
- both could be distributed over a network of different computers.

With the increased interest in such intercommunicating hybrid systems and heterogeneous computer systems, there is an increasing necessity for technology that would allow independent agents to be flexibly integrated, and their activities co-ordinated. In designing these multi-agent systems, researchers and developers must address three important issues relating to intercommunicating systems:

- ***Design Autonomy***: where components of the system are designed and implemented as independent entities.
- ***Communication Autonomy***: where components of the system initiate and respond to communication independently.

- ***Execution Autonomy***: where components are executing within the system as independent entities.

To meet the requirements of intercommunicating systems, for design, communication and execution autonomy, many researchers are developing modular mechanisms for configuring and co-ordinating agents. For example, the International Standards Organisation's (ISO) Open Systems Interconnection Reference Model [80] identified the use of layers containing services with well-defined *interfaces*, with the services being further divided into *modules* (possibly distributed), as key concepts in structuring complex intercommunicating hybrid/heterogeneous systems.

As intercommunicating systems require a structured modular approach that has a well-defined interface, it is becoming recognised that *Object-oriented programming* can provide this modular structure and the design, communication and execution requirements for these types of systems. This is illustrated by the increasing number of standards activities related to intercommunicating hybrid/heterogeneous systems. These include communications and distribution standards, database and programming language standards, and repository standards, that are all moving towards adopting, or have already adopted an object-oriented approach. These activates include not only those of the official standards bodies, such as ISO and ANSI, but also those of industry consortia, such as the Open Software Foundation and the Object Management Group.

Object-oriented programming is a software engineering methodology which forms a natural model for building intelligent hybrid systems. Object-oriented systems directly model entities from the problem domain as a set of distinct software units called objects, that interact by passing well-defined messages. For example, in an object-oriented banking application there may be Customer objects that send debit and credit messages to Account objects. These Account objects may co-operate to maintain Cash-on-Hand and Accounts-Payable objects.

Object-oriented programming is ideal for building hybrid systems (especially intercommunicating hybrids) because active objects can represent independent information processing techniques or computer resources, that can communicate with each other by using well-defined message interfaces. These messages not only contain data but also the name of the operation to be executed by the receiving object. As operations are named rather than called directly, objects are highly interchangeable and a given object can be used wherever an appropriate message protocol is specified. Also as objects are independent entities that interact through message passing, they are ideally suitable for distribution across a network of computers. This would enable power hungry applications to distribute the workload across a network of powerful machines. From these capabilities

it is evident that the object-oriented approach represents the exact definition and requirements of intercommunicating systems, be they hybrid systems or distributed computing environments.

Before examining object-oriented integration for hybrid problem-solving, the following sections will overview terminology, structures, and advantages of object-oriented programming.

4.2. Overview of Object-Oriented Programming

The concept of object-oriented programming is a relatively new methodology for designing and implementing software systems. Its major commercial goals are to improve programmer productivity by increasing software extendibility and reusability, and therefore to reduce the complexity and cost of software maintenance.

Object technology arose from a desire to change the way we design, develop and manage the complexity of software systems. A properly designed object-oriented application should contain objects that directly model the application domain, therefore making the application easily understood by the people involved in the development and maintenance process. Within a large application, objects abstract and distribute the dependencies that exist between data and functions into many smaller and simpler pieces. Breaking down an application into entities and relationships that are meaningful to end users is a common conventional programming-analysis technique. Unlike conventional programming, object-oriented programming preserves this same decomposition through the design and implementation phases. Figure 4.1 shows the abstraction of code and data into small separate objects that have a well-defined interface for manipulating the data.

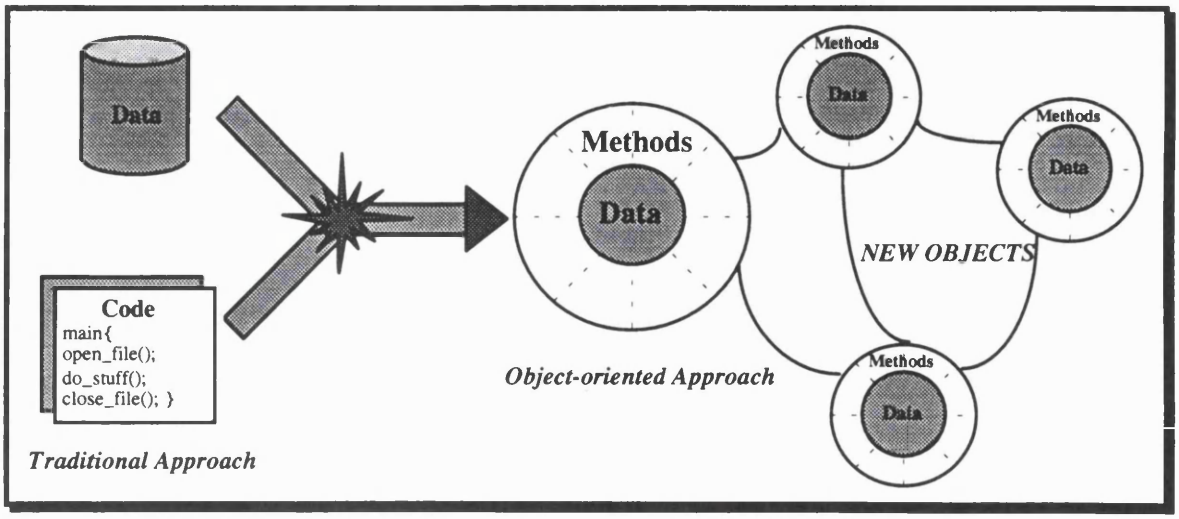


Figure 4.1 — Object-oriented Approach.

Traditionally in programming systems, data and procedures are separate entities, where the programmer is responsible for applying active procedures to passive data structures. Whereas, an object-oriented programming system doesn't view an object just as passive data but as the combination of the object's current private state and the methods that manipulate that state. In a typical object-oriented language, the definition of a type is often called a *class*. A class definition defines both the instance variables (the state or data) and the methods (operations) for objects of that class. These methods form a well defined interface to the functionality of objects of this class and forms the basis for allowing the user access to the data and functions within an object. This well defined modularity means that objects can be reused in similar applications in the same area. Reusing, rather than reinventing software speeds the development and maintenance of large applications.

Reusability is enhanced in object-oriented programming because the concepts encapsulated in a class are accessed via a clearly defined interface. The user needs to only understand the behaviour of the objects as specified in the interface, without concern about the implementation. The user can treat the implementation of an object as a "black box" that is hidden from view. This functional approach and the ability to have many object instances of the same class, enables objects to be reused in the design and implementation of an application.

Maintainability is enhanced in this approach because changes in the implementation of a data structure or an operation (i.e. code within the class implementation) can be localised to the region of code that implements that particular functionality within the class. This localisation makes detection and correction or problems much easier and faster.

Object-oriented programming centres around several concepts: abstract data types and classes, type hierarchies (subclasses), inheritance, and polymorphism.

4.3. Objects, Classes and Encapsulation

The architecture of an object-oriented software system is built around a set of *classes* that characterise the behaviour of all the underlying data in the system. Objects from each class are manipulated by invoking methods of the class; that is by sending messages to these objects. These messages represent the actions that are taken on the sets of objects.

Object-oriented programming focuses on the data to be manipulated rather than on the procedures that do the manipulating. The main challenge of object-oriented software design is the decomposition of a software system into the underlying data types or classes

and subclasses. The objects or class variables correspond to the physical or logical entities in the domain of the actual problem.

An *abstract data type* is a model that encompasses a type and associated set of operations. These operations are defined for and characterise the behaviour of the underlying type. In most object-oriented languages, a *class definition* describes the behaviour of the underlying abstract data type by defining the interface to all the operations that can be performed on the underlying type. The class definition also specifies the implementation details or data structure of the type. When all or parts of the data types are accessible outside the scope of the class, we call such portions of the type *public*. Usually these implementation details are accessible only within the scope of the class and are known as *private*.

The private section of a class definition usually defines the data structures of the underlying data type. It also specifies the interface to the methods that are accessible only within the scope of the class. These methods are often used to support the implementation of the public methods. The private section of a class may sometimes include objects from other classes that can be manipulated only within the scope of the given class. The public section of a class usually specifies the interface to the methods that form the basis for the reusability of the class across many application areas. The user can only operate on an object using this public interface provided by the implementor.

The operations that are defined for a class are called *methods*. These methods are analogous to procedures and functions in non object-oriented languages. If the public operations of a class are general enough to be applicable in many applications areas, the class can form the basis for a reusable software component. This also has the added advantage that the implementation of the encapsulated object can be changed without effecting the applications that use it.

An *object* is an instance of a specific class and will represent a state instance of that class by containing a copy of all the data (both private and public) that was defined in the class definition. Actions may be performed on such an object by invoking one or more methods defined in the class definition. The process of invoking a method is called sending a *message* to the object. Such a message typically contains the name and parameters of the method to be invoked, just as in a function call invocation in a non-object-oriented language. The invocation of a method (sending a message to an object) typically accesses or modifies the data in the particular object.

Encapsulation is the backbone of object-oriented programming and is the process by which individual software objects are defined. Encapsulation involves the definition of :

1. A clear boundary that encompasses the scope of all the object's internal software.
2. An interface that describes how the object interacts with the other objects.
3. A protected internal implementation that gives the details of the functionality provided by the software object. These implementation details are not accessible outside the scope of the class that defines the object.

The concept of encapsulation is related to class descriptions; but it also provides a refinement on how the various components of a problem solution are grouped. The unit of encapsulation is the object, which has properties described by its class description. These properties are shared with other objects of the same class.

An object-oriented language is said to be *extensible* because the programmer can create new types that may be endowed with specific properties and whose behaviour is characterised in a class definition. Objects from these new classes can be manipulated in much the same way as the predefined types provided in the programming language itself.

4.4. Subclasses – Inheritance and Polymorphism

The object-oriented approach to problem-solving uses objects for encapsulation and defines objects to be instances of classes. These class can be structured into hierarchies where some classes are subordinate to others and are called *subclasses* or *derived classes*. Subclasses are considered to be special cases of the class under which they are grouped in the hierarchy. The lower levels in a class hierarchy usually represent an increased specialisation, and higher levels usually represent generalisation.

A subclass definition characterises the behaviour of a set of objects that *inherit* some of the characteristics of the parent class. *Inheritance* is utilised in these class hierarchies so that new objects (children) can be derived from the older ones (parents). The new objects can have their own data or operations and modify the existing ones derived from their parents. Inheritance serves as the basis of code reuse and sharing in an object hierarchy.

The cost and complexity of software development can be reduced by allowing for the creation of subclasses. Inheritance and subclasses can lead to incremental problem solving where instead of modifying existing software components, or rewriting them, new subclasses are created from a set of baseline classes. Objects derived from these new subclasses form the foundation of the software architecture.

In most object-oriented languages, if class P is a parent of subclass S, then an object s, of subclass S, can be used wherever an object p, of parent class P, can be used.

This implies that a common set of messages (i.e. operations) can be sent to objects of class ***P*** and class ***S***. When the same message can be sent to objects of a parent class and objects of its subclasses, this is defined as *polymorphism.*

Polymorphism allows each object to respond to a common message format in a manner appropriate to the subclass from which the object was taken. Because of polymorphism, it is very easy to plug new objects into a system if they respond to the same set of messages as existing objects. For example, if a Window object expects all graphical objects to be able to draw themselves. Then it is very easy to add a Rectangle object to this system that previously had only Lines and Circles. This is possible because polymorphism would allow the new object to respond to a draw message in its own manner.

Different object-oriented languages provide a range of capability relating to the extent to which objects of a subclass may inherit, extend, or override the characteristics of the parent class. Some object-oriented languages support multiple inheritance, in which a subclass has more than one parent class.

4.5. Advantages of Object-Oriented Integration

Due to the many modular characteristics (i.e. encapsulation, inheritance, polymorphism) and the well-defined message interface, object-oriented programming naturally accommodates many possibilities for representing intelligent hybrid systems. These properties would enable different information processing techniques to be represented as objects and integrated with other processing components via a well-defined message protocol. These properties would also allow the easy exchange or addition of new processing techniques.

The use of object-oriented message passing as a well defined communications protocol allows links between different components to be established at runtime. Message passing provides the basis of the most flexible form of dynamic linking in a object-oriented hybrid system. At the lowest level, a message is simply data passed from one computational unit to the next. Combined with queues and the ability to send messages asynchronously, each object in a hybrid application could process a stream of data relatively independently of other units.

Employing object-oriented techniques allows the ability to manage concurrency because message passing systems are adaptable to many kinds of hardware environments, including parallel multi-processor machines and distributed systems. In the case of distributed systems an intercommunicating application may have components running on a variety of different machines, making use of specialised architecture's of these machines.

Object-oriented programming can be exploited as a means of creating a system for parallel architecture's without having to explicitly deal with processes and inter-process communication. The use of objects provides an abstraction that hides processes and inter-process communications within the self contained object and the message protocol used by the processing components.

There are other added advantages with respect to code reusability when adopting an object-oriented approach to building hybrid systems. Code reusability allows for multiple copies of the same processing technique to be active at the same time. Therefore these objects can be linked together for chained processing. For example, several expert systems or neural networks could be developed for sub-tasks of a problem and chained together to form the final solution.

The use of object-oriented programming for integration can also provide a platform for rapid prototyping of applications (see Chapter 8 concerning the Esprit HANSA Project). This rapid prototyping is possible because objects can represent different processing techniques that all communicate via a standard message passing protocol. This means that different processing techniques can be "plugged" together quickly and the prototyped solution to the problem tested. This rapid application generation would allow developers to experiment with different configuration of processing techniques to obtain the best solution for a problem. This would save time and would make for better applications because the best techniques could be used to solve the sub-tasks that make up the application. This approach allows the best tool for the job to be selected and the best tools to be combined to form the optimum application solution.

The main challenge for hybrid systems developers is in setting up a framework in which processing techniques can be represented, executed and allowed to communicate using a standard message protocol. This type of framework would allow newly available processing techniques to be integrated if they conformed to the message protocol. This "plug and play" capability requires that information processing techniques are represented as objects within the operating environment. To meet these requirements many commercial object-oriented integration methods are being developed that will allow the seamless interoperability of distributed applications [57].

4.6. Object-Oriented Integration For Hybrid Systems

With the many advantages that object-oriented programming offers, there are clearly many possibilities for implementing hybrid systems using object-oriented techniques. The three types of hybrid systems (function-replacing, intercommunicating and polymorphic) [36], can use object-oriented programming at various levels, ranging from

the actual programming of the processing technique using an object-oriented language such as C++ [121], to the use of an object-oriented programming environment and an information exchange protocol.

For example, function-replacing hybrids (i.e. hybrids where a principle function in one technique is replaced by another processing technique) can be represented at the lowest level. Both of these processing techniques could be programmed as a collection of objects, where internally they represent their principle functions as component objects. This object-oriented representation means that a principle function object such as the pattern matcher in an expert system, could be easily replaced by a another object that encodes a processing technique that is more efficient at pattern matching, such as a neural network. This replacement would therefore enhance the abilities of the host technique.

An intercommunicating hybrid (independent techniques that communicate by exchanging information) is naturally suited to object-oriented programming because techniques can be represented as objects and communication can be conducted via object-oriented message passing or via an industry standard linking and communication protocol, such as Object Linking and Embedding (OLE) from Microsoft [18]. Components within intercommunicating hybrids can be easily distributed across a network of computers where they communicate using one of the many emerging platform independent communication protocols, such as the Object Request Broker (ORB) from the Object Management Group [38].

Lastly, polymorphic hybrids (systems that use a single processing architecture to achieve the functionality of different processing techniques) can be programmed at the lowest level using object-oriented methods. From the definition, the lowest level is the single architecture of a technique, for example the neurons and connections of a neural network. Object-oriented programming could be used to build the low-level structures of the architecture, in the neural network example, neurons could be easily modelled as objects and connections stored in a list or matrix of pointers to these neuron objects. This object representation allows symbolic processing mechanisms such as variable binding or logical inferencing, to be built into the low-level neural representation by simply extending the functionality of the neuron objects [48].

The use of object orientation in integrating diverse processing components within hybrid systems is also characteristic of recent developments in application integration software, such as Hewlett-Packard's NewWave, Object Linking and Embedding (OLE) in Microsoft Windows, and in many planned Distributed Computing Environments (DCE). The requirements of building intelligent hybrid systems and distributed computing environments are extremely similar in that they require a representation scheme and a well-

defined communication protocol. The goals of using object technology in all of these systems is to provide the "glue" which allows applications to share information with one another in an easy and consistent manner.

4.7. Summary

By combining current adaptive and symbolic processing systems to exploit their individual strengths within intercommunicating hybrid systems, requires no major developments, but rather a mechanism to represent and link the current technologies. This approach, therefore appears to provide the path of least resistance in the short term. This intercommunicating hybrid approach offers immediate hybrid solutions and still allows independent development of adaptive and symbolic techniques. This also allows other existing and any new techniques developed in the future, to be integrated easily into a hybrid system.

To build such a class of hybrid system requires a methodology that allows techniques to be independently represented, that offers a well-defined interface and communication protocol. With respect to this implementation viewpoint, this chapter shows that intercommunicating hybrids and future distributed computing systems have a direct mapping onto object-oriented programming. The object-oriented software engineering approach is examined and its main advantages for constructing hybrid systems are highlighted. This analysis shows that object-oriented programming allows processing techniques to be represented as independent objects that communicate via a standard message passing mechanism. Also this software approach allows the *design*, *communication* and *execution* autonomy required by intercommunicating hybrid systems.

Chapter 5

The ORBERON Environment

This chapter describes the design and implementation of key components in the object-oriented hybrid environment called ORBERON. The fundamental components of this environment, such as the Generic Interface Object, Dynamic Environment Manager, the communication and windowing system, are detailed. The information processing techniques that were integrated within the environment were an Expert System and a Neural Network shell. The design and implementation of these information processing shells, are examined in detail.

5.1. Introduction

With the growing awareness and need for hybrid systems, there is also a necessity for environments that allow hybrid systems to be built both for research and for commercial usage. The ORBERON hybrid environment was primarily designed to investigate the interface, functionality, communication and co-ordination mechanisms required for hybrid systems. This design was used to demonstrate the advantages of object-oriented technology as an integration methodology. The initial techniques that were integrated into the environment were expert systems and neural networks because they represented the most popular and most developed techniques from symbolic and adaptive processing approaches. To show that hybrid solutions are feasible and better than single processing technique solutions, the ORBERON environment was applied to the real-world Cargo Consignment problem from British Airways.

The main design requirements for the ORBERON environment are :

- **Flexibility**: The environment must allow for various ways to combine different processing techniques.
- **Extendibility**: The environment must allow the easy installation of new processing techniques.
- **Design Autonomy**: The environment must allow components of a hybrid system to be designed and implemented as independent entities.

- **Communication Autonomy**: The environment must allow components of a hybrid system to initiate and respond to communications independently.
- **Execution Autonomy**: The environment must allow components to be executed as independent entities.

To meet these requirements the environment and the core message passing mechanism for communication between processing techniques, are constructed using the C++ programming language, so as to apply an object-oriented approach to hybrid systems. Unlike traditional programming methods that are based on concepts such as data flow or logic, object-oriented programming directly models the application by performing computations using message passing between functions in active objects [106]. These active objects represent actual entities in the problem domain. Figure 5.1 shows that in the ORBERON environment, objects represent separate information processing techniques. In the previous chapter the suitability and advantages of using an object-oriented approach for integration have been made explicit. The object-oriented design of the ORBERON environment allows the integration of techniques from the whole spectrum of information processing. Currently a modified public domain expert system shell called CLIPS and a Back-Propagation neural network shell (developed for ORBERON) have been integrated.

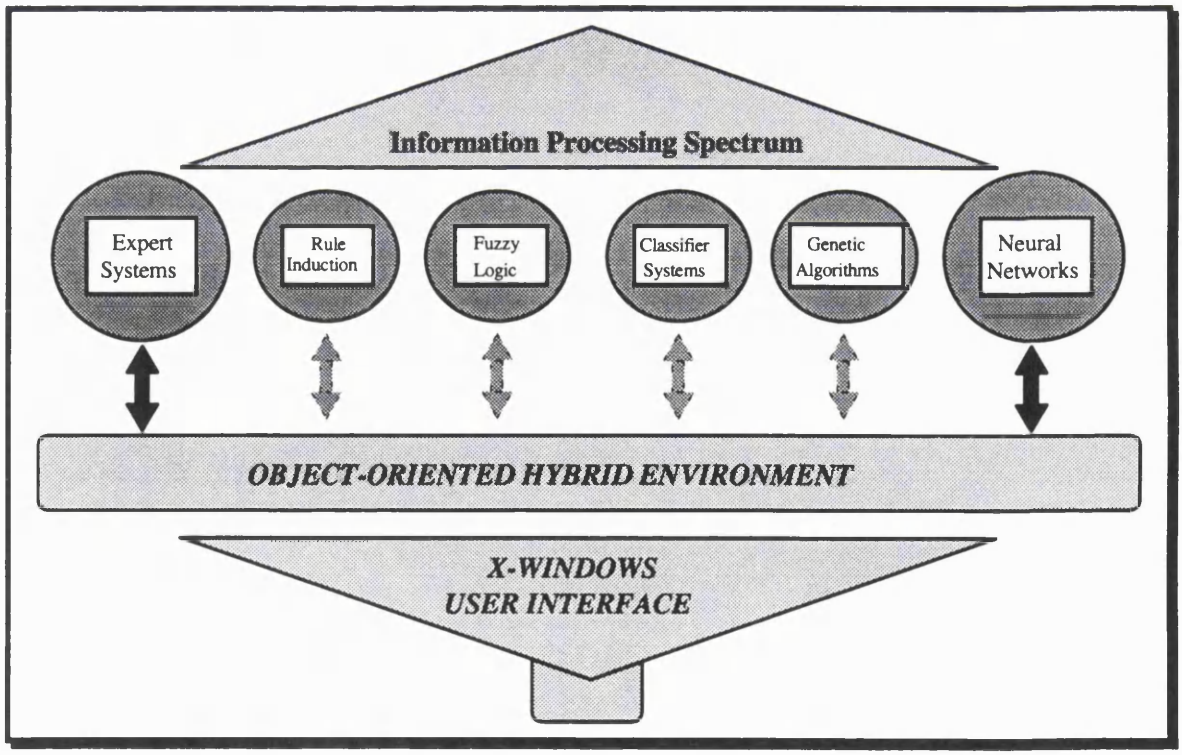


Figure 5.1 — System overview of the ORBERON Environment.

The object-oriented approach shown in Figure 5.1, represents an *intercommunicating* hybrid [36] with expert systems and neural networks as the primary

processing techniques available for integration. This intercommunicating approach differs considerably from earlier research into the use of neural networks *as* expert systems [31, 102, 95], where neural networks tried to simulate rules and symbolic operations. These systems tried to unify symbolic processing within the architecture of neural networks [88]. Previous intercommunicating expert system and neural network hybrid systems only allowed communication via intermediate files and required the data to be pre-processed for use by existing commercially available packages [7]. Most of these systems only have the ability to display and use one processing technique at a time during the development and execution of a problem solution. These hybrid systems also offer no integrated environment for combining different techniques. These problems have been a major barrier to developers creating hybrid systems. With the advent of object-oriented techniques, these limitations can be overcome. The ORBERON environment presented in this chapter offers the communication and co-ordination to "glue" together different information processing techniques. Because the environment allows the use of existing programming "shells" for different information processing techniques, it allows hybrid solutions to benefit from the advanced programming and monitoring facilities found in most of these shells.

The basic design concepts and principles found within the fundamental components of the ORBERON environment are described in the next section. These can now be seen to correspond (although rather rudimentary in implementation) to basic components within current object-oriented integration methods such as the Object Request Broker [38] from the Object Management Group and Object Linking and Embedding [18] from Microsoft.

5.2. Fundamental Elements of the ORBERON Environment

The ORBERON hybrid environment was built within an object-oriented framework, so that each processing technique appears externally to be an object despite its actual internal implementation. Figure 5.2. shows that within the system all information processing techniques are "wrapped up" within the Generic Interface Object. This allows the standard communication protocol to be inherited from the Generic Interface Object. When one of these processing techniques is activated, it's name is inserted into the Dynamic Environment Manager. This structure maintains and manipulates references to the active objects within the environment. Active objects communicate via the Dynamic Environment Manager by sending set messages to this structure, which decodes and routes the message to the appropriate object. In the following sections this structure along with the communication protocol and the windowing system, will be examined.

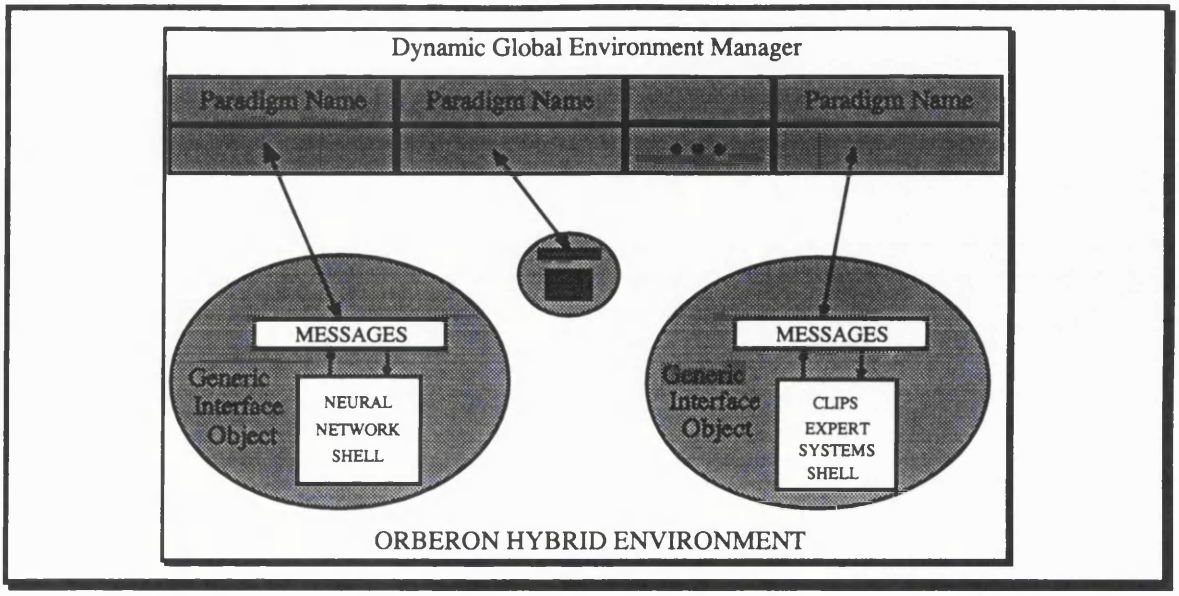


Figure 5.2 — The ORBERON Environment Structure and Components.

5.2.1. Design of Generic Interface Object

Many companies, although committed to Object Technology, have a legacy of conventional code that they must use within any new system. These legacy systems often have an existing large development investment and the cost of rewriting with object technology will be unfeasible. What is required is a mechanism to built on the existing investment by allowing the immediate integration of old code with object-oriented systems.

Object Wrappers can be used to migrate legacy code to object-oriented systems while protecting investments in conventional systems. The terminology was originally coined by Wally Dietrich of IBM [22] because with object wrappers it is possible to create objects by wrapping up the bulk of existing conventional code within a wrapper object and therefore protecting the investment in older systems. An Object wrapper enables an object-oriented part of a new system to interact with conventional code via message passing to the wrapper. The wrapper forms an interface for the objects in the new system to access the legacy system within the object wrapper. The wrapper can offer all or a restricted set of functions that the conventional system can perform. Figure 5.3 shows that when services of the old system are needed, these are requested by passing a message to the object wrapper which calls the appropriate function from the internal code and the output returned to the requesting object. As entire functional areas of the old system are replaced by objects, all that needs to be done is to disable the functions in the wrapper's interface, thereby never touching the old code again.

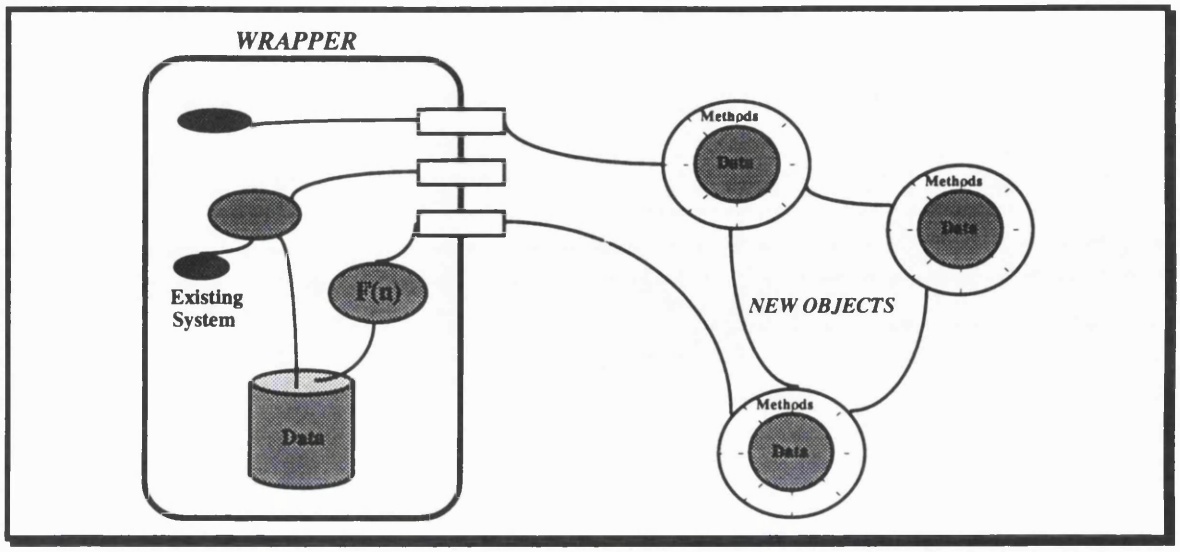


Figure 5.3 — Object Wrapper interfacing new objects with an existing system.

The Generic Interface Object is essentially similar to an Object Wrapper in that it is used to encapsulate existing information processing techniques (whether the technique is object-oriented or not). Although it also has the added functionality of containing the interface functions required for the information processing technique to communicate with the existing objects in the hybrid environment.

To permit the required heterogeneity, the ORBERON environment is based around this ***Generic Interface Object*** wrapper, within which, there is defined a standard set of messages and functions for communicating to other information processing objects. Figure 5.4 shows that each new technique that is added to the environment, must be derived from ("wrapped up inside") this Generic Interface Object, so that the pre-defined fundamental communication protocol can be inherited.

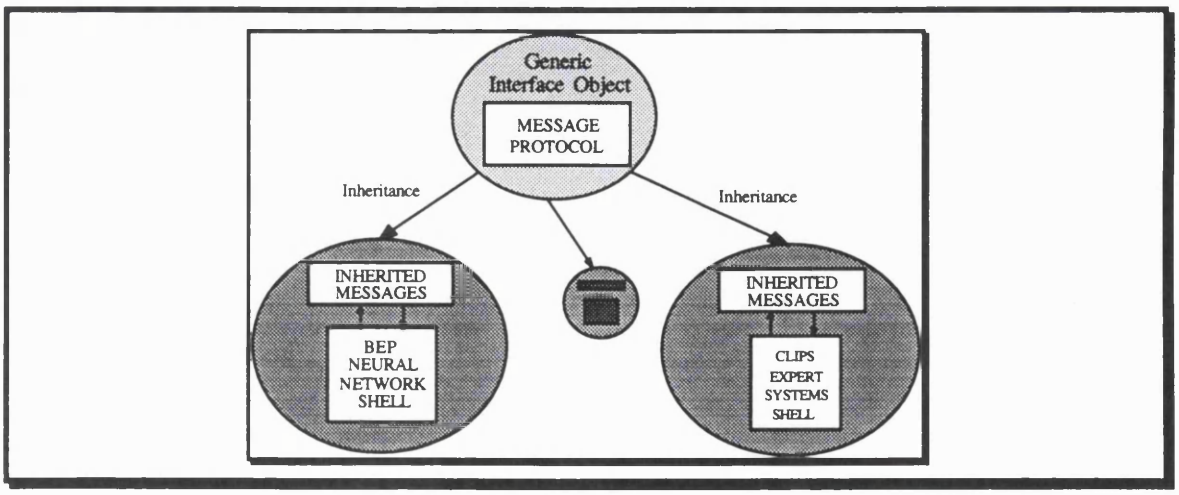


Figure 5.4 — Generic Interface Object is inherited to form the wrapper for new information processing objects.

This process of "wrapping up inside" the interface object utilises two properties of object-oriented programming called *polymorphism* (which allows multiple classes of an object to respond differently to the same message) and *inheritance* (the ability to define a new object that inherits properties of the parent object) to obtain the functionality that is required by the new object to communicate with and receive messages from the other information processing objects. This allows easy insertion of new information processing objects into the system, so that they respond to the same messages as existing objects. Once this is complete, the information processing object can then be used within the environment.

5.2.2. Design of Dynamic Environment Manager

Each new processing technique which is added to the environment is manipulated and accessed via the ***dynamic environment manager***. The environment manager maintains a structured list containing the name, the type of information processing system (i.e. expert system, neural network etc.), and a reference to the information processing object. This structure is maintained and manipulated as information processing objects are activated and de-activated. The Dynamic Environment Manager has three major functions:

1. To present a simple front end to the user of the environment, that allows the selection of as many information processing objects as required to solve the problem at hand.
2. To register and de-register information processing objects as they are activated and de-activated.
3. To route messages between active processing objects.

Figure 5.5 shows the simple front end to the ORBERON environment that allows the user to select different information processing systems. At present only neural networks and expert systems are available within the environment.

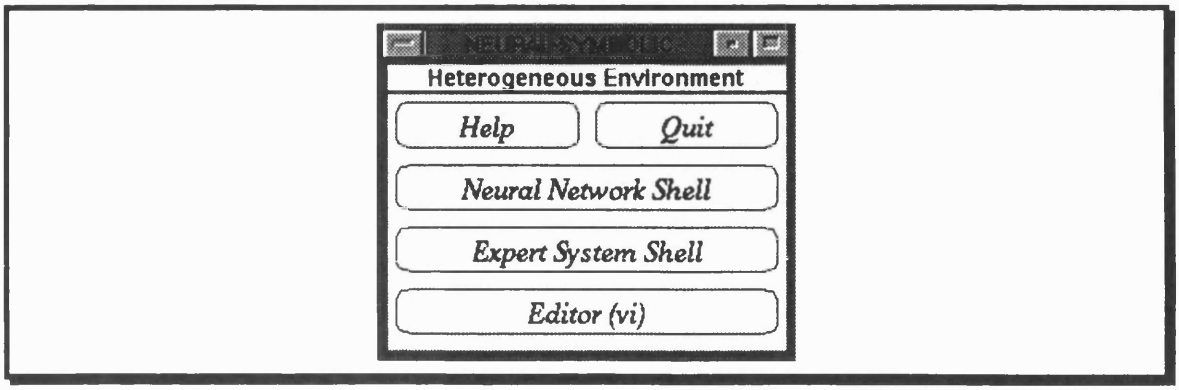


Figure 5.5 — Front End to the ORBERON Environment.

The front end also offers an editor, some help text and an option to exit from the environment. An information processing object (such as a neural network or an expert system) can be activated by either selecting it from the environment front end or automatically by another processing technique. On activation the environment requests a name for the processing object window. This name is passed to the dynamic environment manager to check if a processing object of the same type and same name is already open. If so, the system notifies the user via an error screen. Otherwise, a window and the appropriate information processing system is opened and registered with the environment manager. On de-activation the information processing object is removed from the dynamic environment manager structure and also physically removed from the environment.

Information processing objects within the environment communicate by sending predefined messages via the dynamic environment manager. These messages not only contain data but also the name of an operation to be executed and the name of the intended recipient processing object. The dynamic environment manager receives all messages, decomposes them, checks that the named processing object is active (reports an error if not active) and then routes the original message to the appropriate processing object.

5.2.3. The Communication Protocol

In object-oriented systems, objects communicate by sending messages that not only contain data but also the name of an operation to be executed by the receiving object. As operations are named rather than called directly, objects are interchangeable and a given object can be used wherever an appropriate message protocol is specified. There are certain issues that need to be considered when designing a communication protocol for intercommunicating hybrid systems. These issues concern *data transformation*, *throughput* and *flow control.*

Data passed from one processing technique to the next must be in the correct format if the receiving technique is to make use of it. Data transformation can take place either at the sender or receiver, or may be performed by an intermediate third party. For instance, data passed from one component to the next can be transformed by a specialised translator object. In the ORBERON environment the transformation process is the responsibility of each processing object and data must be transformed before it is transmitted.

Throughput has to be considered so that the flow of data from one component, to the next, is kept balanced. Bottlenecks can form where components with a high data

throughput are mixed with those with a lower throughput. In such situations, many of the advantages of speed and efficiency can be lost.

Flow control is important if data needs to be queued or processed in a particular sequence. Without flow control there is the danger of uncontrolled pools of data collecting at various points in the application. Such problems of throughput and flow control are overcome in ORBERON by using one expert system rulebase as the main control module for managing throughput and flow control through the various information processing objects.

The message passing communication protocol was devised to allow each processing system to access the main functions inherent in each processing system. For example, the message set below is sufficient to allow access to the prime functions that are performed by neural networks and expert systems (net_name is the name of the neural network object):

- **Net_Check (char* net_name, char* input)** - This is a call to a neural network object specified by *net_name*. The message calls the Net_Check operation to interrogate the network that is loaded, with the input data specified in the character array *input*, and returns the output of the neural network.

- **Net_Train (char* net_name, char* train_file)** - This is a call to a neural network object specified by *net_name*. The message calls the Net_Train operation to train itself on the input-output data set which is found in the file named *train_file*.

- **Net_Load (char* net_name, char* net_file)** - This is a call to the shell within the neural network object specified by *net_name*. The message calls the Net_load operation to load a neural network that has been saved in the file *net_file*.

- **Net_Win (char* net_name, char* icon_window)** - This is a call to the environment manager to create a neural network object with name specified in *net_name*, register it within the environment manager and display it within the hybrid environment as an icon or a window depending on the contents of the string *icon_window*.

- **ES_Load (char* es_name, char *filename)** - This is a call to an expert system object specified by *es_name*. The message calls the ES_Load operation to load a rulebase file specified by *filename*.

- **ES_Run (char* es_name, int run_limit)** - This is a call to an expert system object specified by *es_name*. The message calls the ES_Run operation which will execute the current rulebase. If a number is specified in *run_limit* then the

execution will stop after the specified number of rules have fired. Otherwise, the execution will run until complete.

- **ES_Assert (char* es_name, char *fact)** - This is a call to an expert system object specified by *es_name*. The message calls the ES_Assert operation to add the fact found in the string *fact* to the factbase.

- **ES_Slow_Assert (char* es_name, char *fact)** - This is a call to an expert system object specified by *es_name*. The message calls the ES_Slow_Assert operation to add the fact found in the string *fact* to the factbase.

These messages have been defined within the Generic Interface Object and inherited by the Back-Propagation neural network simulator object and the CLIPS expert system object both of which will be described in the next section. The structure of each message is defined when the new information processing object is added to the environment and corresponds to the main functions that are available within the processing system. Typically the message structure contains the name of the function to be executed, the name of the particular information processing object that the function is to be executed within and a variable number of data fields. The message is passed to the ***dynamic environment manager*** which checks the named processing object actually exists and the function required is appropriate, and then routes the message to the named object. The receiving object will extract the data, perform the appropriate function and return the results in a predefined data format. For example the expert system rulebase can send these messages via predefined "interface predicates" [108] of the same name e.g. when a rule needs to check the output of a neural network, it can send a Net_Check message with the appropriate arguments to the environment, which will pass the message to the correct neural network object which will return the results back to the rule. The following rule illustrates how an expert system sends the Net_check message and processes the results returned from the neural network.

```
(defrule GetNetworkOuput
     (Test Network "PTA" Output)
     (Network "PTA" Input ?input)
     =>
     (printout t ">>>==>>> Interrogating Neural Network"crlf)
     (bind ?string (Net_check "PTA" ?input))
     (bind $?output (str-explode ?string))
        (assert (NNOutput "PTA" $?output)) )
```

The final component of the ORBERON environment is the windowing system that allows the user to interact with the information processing objects found within the hybrid environment.

5.2.4. Design of the ORBERON Windowing System

The implementation of the expert system and neural network objects, allowed a suitable user interface and communication protocol to be developed for the ORBERON environment. The user interface consists of a X-Windows front end, with each information processing object having an associated window in the environment. The user can interact with the processing objects via these active windows during development or execution of a solution for a particular problem. The environment allows as many active neural networks or expert systems as the problem requires.

The user interface was constructed using an object-oriented package called InterViews (Interactive Views) from Stanford University [61]. InterViews is a library is of C++ Classes that simplify the design and implementation of a graphical user interface. InterViews makes building applications for workstations easier by providing abstractions that hide the details of the underlying X-Windows system. Figure 5.6 shows the InterViews class hierarchy, which contains common objects such as scroll bars, menus, buttons and a mechanism to combine these objects.

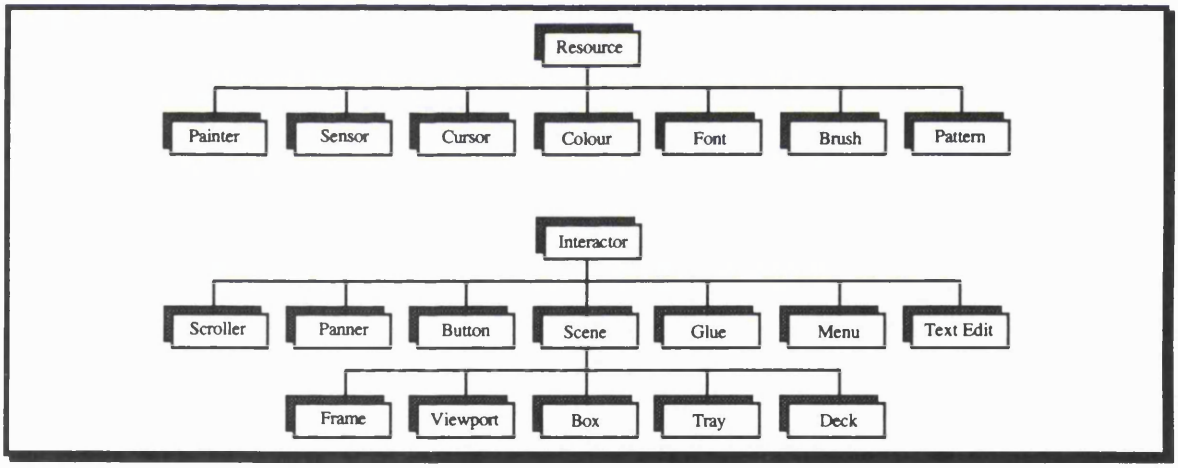


Figure 5.6 — InterViews Class Hierarchy.

Two types of windows were designed to cater for the user interface requirements of neural networks and expert systems shells (these requirements being representative interactions for other possible processing systems). The window for the neural network shell was designed so that it could understand commands for the C-Curses package [5]. This package allows access to the screen via x and y co-ordinate geometry, so that text can be placed at specific locations on the screen (see Figure 5.8). This Curses style window also allows access to a main control menu for the neural network shell object. The expert system object required a window that could accept terminal style command line input and display the output from the expert system (see Figure 5.7). These basic window

types are available for any future information processing objects to use, by simply defining in the new processing object, the style of window that the object will use as its output screen.

Now that the fundamental components of the ORBERON environment have been examined, the first information processing objects to be incorporated into the environment are detailed. Expert systems and neural networks were chosen because they represent the most popular systems from symbolic processing and adaptive processing.

5.3. The ORBERON Neural Network and Expert System Objects

The neural network and expert system object implementations form the test bed for the development of the fundamental components, user interface and communication protocol to be found in the ORBERON environment.

The expert system and neural network programming shells are represented as objects within the environment and are derived from the Generic Interface Object, which is the building block object wrapper for the ORBERON environment. These information processing objects consist of a Back-Propagation Neural Network Simulator and a C-Language Integrated Production System (CLIPS expert system) [34]. It was discovered that the best utilisation of information processing techniques within the environment, was to use simple "Shells" for representing and programming each technique.

These neural network and expert system objects can be combined to produce hybrids that can be applied towards solving "black box" type problems, where small neural networks that can be trained via example sets, coupled together through a conventional rule based system [103]. This means that implicit knowledge can be contained within the structure of the networks, as well as explicit knowledge in the rules of the expert system. (see Chapter three for other methods of how expert systems and neural networks can be combined).

Representing processing techniques as objects has advantages for code reusability in hybrid systems. This approach allows for multiple copies of the same technique to be linked together for chained processing. For example, with neural networks this means that they can perform chained inference or a hierarchy of connected networks (i.e. pass the results from one network to another in a chain). Not many neural network systems can perform this task at present. The same multiple processing technique concept can be applied to expert systems (or any processing technique in the ORBERON environment) and will allow small separately prepared rulebases to be chained together. This would enable multiple knowledge sources in one hybrid system to operate collectively and would mean

an end to the maintenance and comprehension problems associated with large monolithic knowledge sources.

5.3.1. CLIPS Expert System

The C Language Integrated Production System, CLIPS, is a shell for developing expert systems. It was designed at NASA/Johnson Space Centre to allow artificial intelligence research, development, and delivery on conventional computers. The primary design goals for CLIPS are portability, efficiency, and functionality. For these reasons the system has been written in the C Language. Because of its high portability, CLIPS has been installed on a wide variety of computers ranging from PCs to CRAY supercomputers. CLIPS is used by many research and commercial establishments for a variety of applications such as agricultural research (Texas A&M University), diagnosing plant diseases (University of Arizona) and scheduling and fault management of space communications and tracking network (NASA).

CLIPS is a forward chaining rule-base language that has inferencing and representation capabilities similar to those of OPS5. The system contains an inference engine and a language syntax that provides a framework for the construction of an expert system. It also includes tools for debugging an application. CLIPS is based on the RETE algorithm [30], which enables very efficient pattern matching. The basic elements of CLIPS are:

1. Fact-list - a global memory of data recording the current state of the system
2. Knowledge-base - contains all domain knowledge as rules
3. Inference Engine - controls overall execution

The CLIPS systems was chosen primarily because it was written in C, but also because of its popularity and easy of use. The fact that CLIPS is written in C, enables the expert system to be encapsulated within the C++ Generic Interface Object and allows easy modification. As evidence of this easy of modification there is already a Parallel version of CLIPS called PCLIPS [39].

The CLIPS syntax is very similar to LISP and is straightforward to learn. It uses the familiar LISP parentheses and prefix notation. Rules are divided into a left-hand side (LHS) and a right-hand side (RHS) with an equal sign/greater than symbol construct (=>) used as a separator. A typical rule looks like this:

```
(defrule dont-walk
     (status walking)
        (or
           (red-light)
           (walk-sign dont walk)
           (police say dont walk))
=>
     (fprintout t "Don't walk" crlg) )
```

When the facts necessary to match all the patterns on the LHS are present, the rule fires and the RHS is evaluated or executed. If several rules are able to fire on a given set of facts, they are placed on an agenda and fired sequentially. Usually the most specific, pertinent rule is placed at the head of the agenda so it executes first. A rule's priority can be altered through assignment of a *salience* value, which can range from -10,000 to 10,000.

Pattern matching occurs on the LHS, and CLIPS allows for literal pattern matching on single and multiple-field variables. Variables are proceeded by a question mark (**?**) and multiple-field variables by a dollar sign and question mark together(**$?**). Multiple-field variables add a simple list-processing capability to CLIPS, and several functions are available for manipulating them. In addition to the use of variables on the LHS, rules can be constrained through the use of included functions and predicates or through user-defined functions.

The RHS describes the actions taken after a rule fires. These actions can include asserting new facts into the fact list, call user defined functions, handle input and output, perform mathematical functions using the basic or extended math package, or make systems calls. Standard programming constructs such as **IF..THEN..ELSE** and **WHILE..END** structures can be used.

The CLIPS system provides a convenient and efficient development environment. Rules and facts can be entered interactively from the prompt or entered into a file via the built in **EMACS**-style editor and load into the system later. The development environment is very similar to many LISP systems. One of CLIPS's strong points is its easy extendibility through the user-definable functions which can be easily added in the C language. The function below called *userfuncs* is the repository for all user-defined functions.

```
/*********************************************************************/
/* USRFUNCS:  The function which informs CLIPS of any user           */
/*            defined functions. To define functions, either         */
/*            this function must be replaced by a function with the  */
/*            same name within this file, or this function can be    */
/*            deleted from this file and included in another file.   */
/*            User defined functions may be included in this file    */
/*            or other files.                                        */
/*            Example of redefined usrfuncs:                         */
/*              usrfuncs()                                           */
/*                {                                                  */
/*                   define_function("fun1",'i',fun1,"fun1");        */
/*                   define_function("other",'f',other,"other");     */
/*                }                                                  */
/*********************************************************************/
exp_sys::usrfuncs()
{
 define_function("Net_check",'s',(FP)net_check,"net_check");
 define_function("Net_load" ,'i',(FP)net_load, "net_load");
 define_function("Net_train",'i',(FP)net_train,"net_train");
 define_function("Net_win",'i',(FP)net_win,"net_win");        // Icon Change
 define_function("ES_load",'i',(FP)es_load,"es_load");        // Chaining
 define_function("ES_run",'i',(FP)es_run,"es_run");           // Chaining
 define_function("ES_assert",'v',(FP)es_assert,"es_assert"); // Chaining
 define_function("ES_slow_assert",'v',(FP)es_slow_assert,"es_slow_assert"); // Chaining
}
```

The user-definable function in CLIPS was used to specify the "interface predicates" that were added into the CLIPS language syntax. These interface predicates allow the expert system to access other information processing objects. For example to interrogate a neural network, the expert system rulebase would use the Net_check interface predicate in one of its rules.

The CLIPS expert system was converted into an object for use within the ORBERON environment by firstly ensuring that it conforms to C++ standards such as prototyping. This is required to ensure that the CLIPS code will compile using C++. The next stage is to inherit from the Generic Interface Object the messages that the expert system is going to respond to from other processing techniques.

The expert system object must now be constructed. This is done by deriving the expert system object from the Generic Interface Object. The expert system object will have inherited the interface functions required to access the other processing objects, namely the neural network object. The expert system object was completed by defining all the functions and the global data from the "wrapped" C system that will be used privately inside the expert system and the functions that will be made public to the other objects i.e. the interface functions. The last task was to include the type of output window required for the expert system. There are two types of windows available at present, one is a command line shell window and other is a CURSES window that understands commands from the C Curses package [5]. The expert system object uses a command line shell window to allow the user to type instructions at a command line. Figure 5.7 shows several expert systems in operation within the ORBERON environment. Once these simple steps are

completed the expert system object is ready to be compiled and linked into the ORBERON environment.

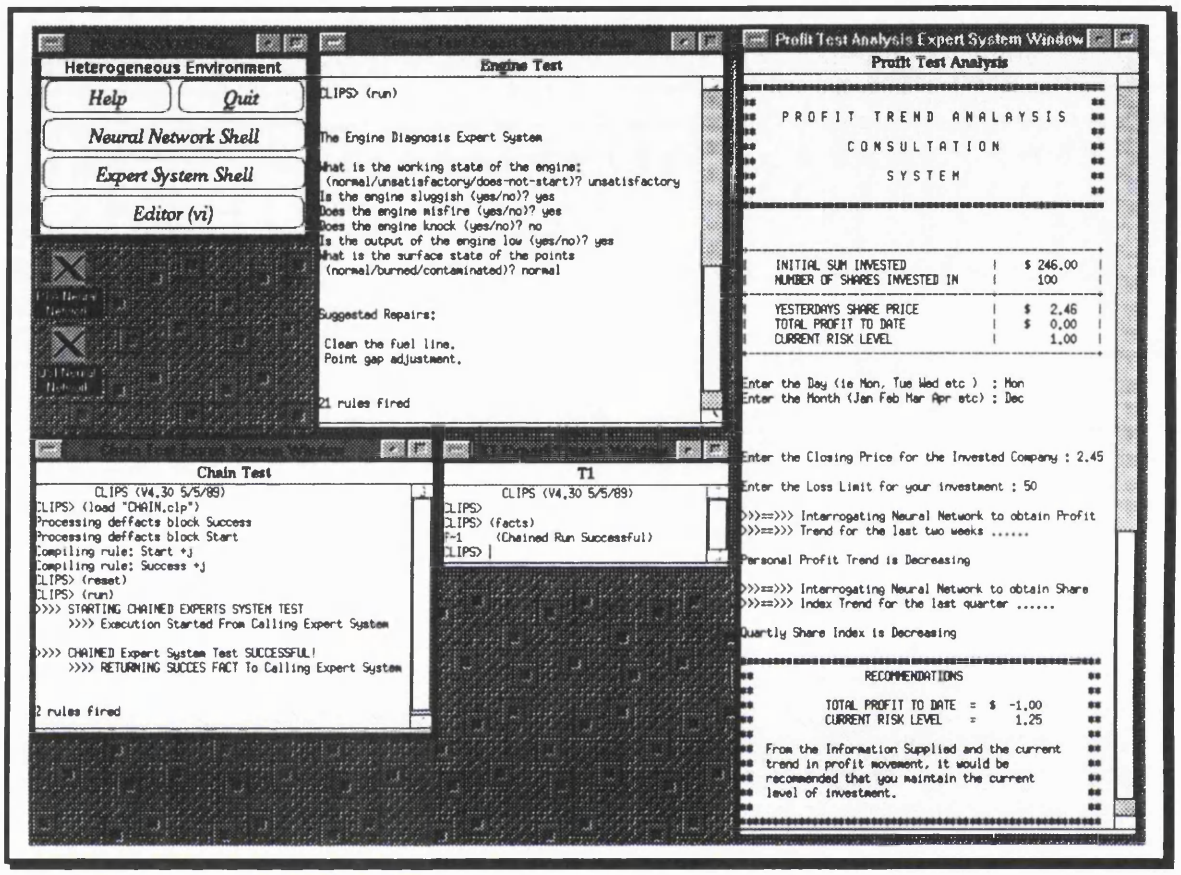


Figure 5.7 — Expert system windows displaying different functions.

5.3.2. Back-Propagation Simulator

The neural network object in the ORBERON environment is a Back-Propagation Simulator. The simulator was designed and implemented to allow the parameters involved in the back-propagation (BP) algorithm [92] to be experimented with. The simulator allows the user to build feed forward networks, train and interrogate the networks using the back-propagation algorithm. It also allows the user to view weights, output values, bias and error values associated with the network, and finally to load and save networks. Figure 5.8 shows two neural network shells within the ORBERON environment.

The structural and functional nature of neural networks make them ideal for construction using object-oriented techniques. This is because the definition of a neural network is modular and therefore suited to object-oriented representation. A neural network can be defined as a collection of neurons that are often formed into layers (input, hidden and output), and these layers themselves are interconnected. These neurons often

all perform the same simple operations on the weight values that feed into them, to calculate the output for that neuron. Any particular learning algorithm can be applied/implemented ontop of this general structure.

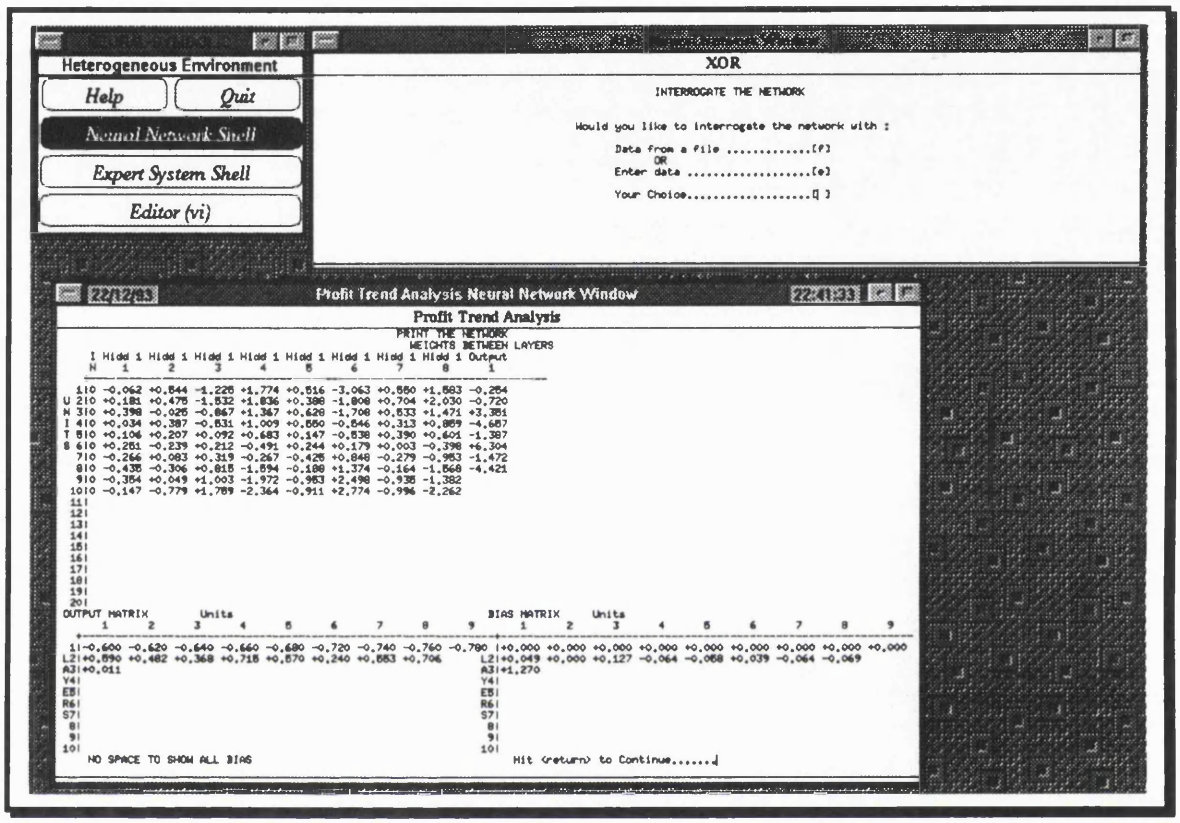


Figure 5.8 — BP Windows showing different Functions of the BP Neural Shell.

This structure suggests a simple object-oriented representation for feed-forward neural networks (i.e. networks where each node in a layer is connected to each node in the next layer), where the basic component is a neuron or processing element object. Each processing element stores values, such as its bias, and output and error values that represent its current state. The processing element also contains a list of weights that feed into that element. A layer object is then simply an array of nodes and a network object is a array of layer objects.

The creation of this structure within an object-oriented language not only allows the implementation of the back-propagation to be modularised into operations for networks, layers and nodes, but also allows these operations to be placed within the objects that they operate on. This also means that once an object and its operations have been defined, e.g. a processing element, then that object and associated operations can be simply reused. Figure 5.9 shows the object-oriented structure of the BP neural network used within the BP Neural Shell.

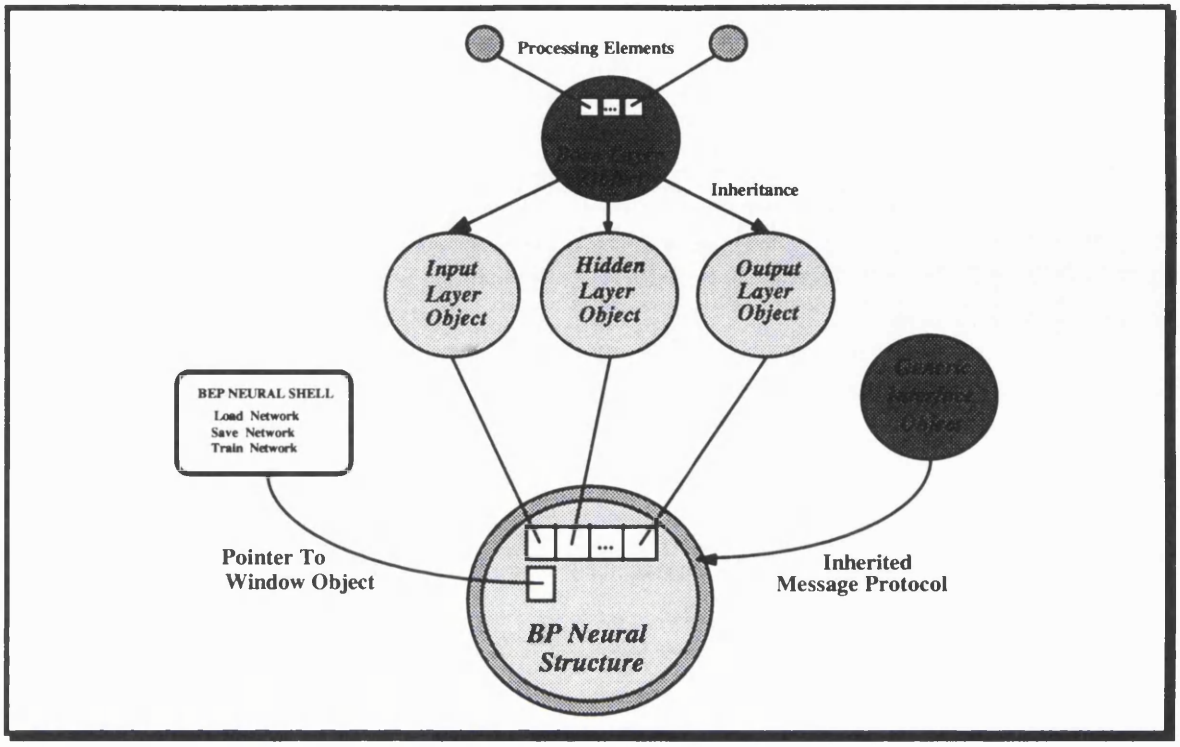


Figure 5.9 — BP Neural Network Structure.

The BP structure is derived from the Generic Interface Object so that it can inherit the standard message protocol. Within the network the layers have some operations that are universal to all layers but others that vary in their functionality depending on the type of layer. This situation is catered for by defining a generic base layer, with generic functions and then deriving each layer from this generic base layer class and overriding the methods that have different functionality. Using inheritance in this manner, different types of BP networks could be constructed for different types of problems, e.g. BP networks used for forecasting [120, 118] and others for pattern classification, as each have slightly different operational details.

The implementation details of the back-propagation neural network are based on the information found in Rumelhart and McClelland's Parallel Distributed Processing, Volume 3 [90]. The main difference being that the algorithm is applied to the above object-oriented structures.

5.4. Summary

This chapter has described in detail the design and implementation of the object-oriented ORBERON environment. The ORBERON hybrid environment takes into consideration the control, message and data passing requirements of different information processing techniques. The fundamental elements of the environment that address these issues have been described.

The integration of a neural network and expert system shell into the ORBERON environment formed the test bed for the development of the fundamental components, communication protocol and the user interface. The design and implementation of the neural network and expert system processing objects, illustrated the object-oriented philosophy and also the procedure required to introduce new information processing techniques into the ORBERON environment. The communication protocol for the neural network and expert system processing objects was outlined along with the windowing system used by these two information processing techniques.

It has been shown that the object-oriented design of the ORBERON environment helps to meet the design, communication, execution and expandability requirements of integrated hybrid applications.

Chapter 6

Hybrid Applications

This chapter describes two hybrid applications that were built using the ORBERON environment containing the neural network and expert system components. Firstly, the design and implementation of a proof of concept application in the financial area of Profit Trend Analysis, is described. This application was used to test the operation of the ORBERON environment. Secondly, the design and implementation details of a real-world Cargo Consignment problem from British Airways is examined. The results of this hybrid Cargo Consignment application are also detailed.

6.1. Introduction

To demonstrate the effectiveness of object-oriented integration and of adopting a hybrid approach, two real-world applications have been implemented using the ORBERON environment. The first was a small proof of concept application in the financial domain of Profit Trend analysis and the second, was a large real-world application from British Airways, for Cargo Consignment.

6.2. Design and Implementation of the Profit Trend Analysis Application

The aim of the profit trend analysis hybrid application was to aid investment analysts in deciding whether to increase, decrease, or maintain investment interest in a particular company. This was achieved by analysing the trends in the company's past performance and in the personal profit for an investor in that company. The application was designed to monitor investment of an initial sum and advise on the best course of action depending on the current risk involved with this investment. The decision process is based on a simple risk scoring system [12], that calculates levels of risk according to pre-defined factors such as those identified by an expert, and the analysis by two neural networks of the movement trends in personal profit for the previous two weeks and the company's quarterly share index.

The decision support and analysis sub-systems were implemented via a CLIPS expert system rulebase. Appendix C gives the full listing of the Profit Trend Analysis application rulebase. The rulebase consists of approximately 60 rules to analyse the results of the two neural networks that were used to classify movement trends (increasing, decreasing or remaining stable) for personal profit over the previous 10 working days and the company's share indices for the last quarter. This pattern recognition task is particularly difficult with conventional symbolic methods due to the erratic movements of shares and profits. Within the application, the expert system provided the control and high level decision making on the low level pattern classification provided by the back-propagation networks.

Figure 6.1 shows the operation of the Profit Analysis application, where the main expert system (long window on the right) was used as the main point of user interaction and application control. When the application rulebase is loaded into the expert system, it automatically starts the Profit Trend Analysis (PTA) and Quarterly Share Index (QSI) neural networks.

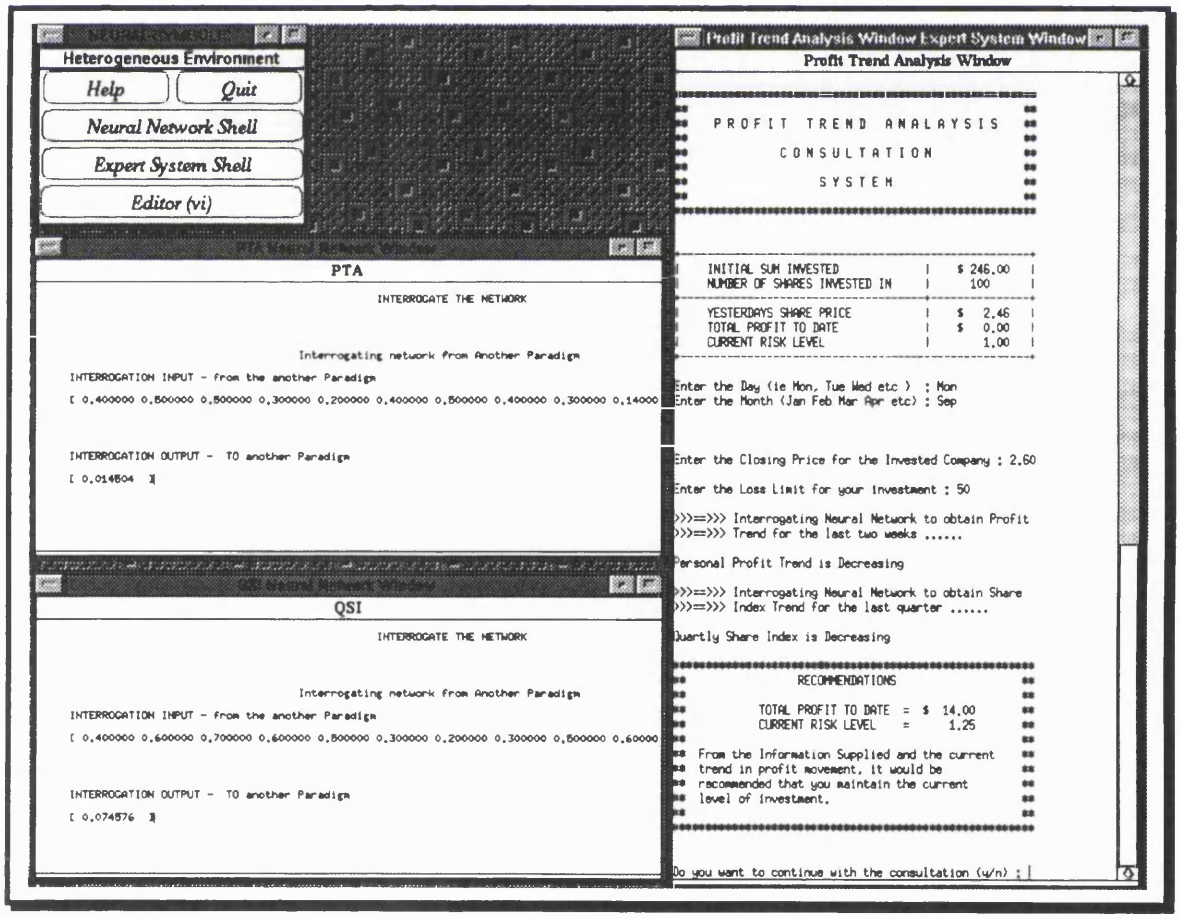


Figure 6.1 — Profit trend analysis application.

These neural networks are pre-trained to classify the direction of movement of trends and are opened as icons, which means that they are used in the background as black box processing elements. These iconised windows have been opened (shown on the left) in Figure 6.1 to show the communication which occurs between controlling expert system and the neural networks.

The flow of control and operation of the application begins by initialising and displaying the default setting for the initial sum of money available for investment, the number of shares brought, yesterdays share price, total profit to date and the current risk level in the investment. The system then prompts for the day (Monday, Tuesday etc.) and the month. This information is important because there is inherent risk on certain days of the week (e.g. Monday or Friday) and in certain months (e.g. January or December) due to the fluctuation in stock prices at the opening and closing of markets. The system then prompts for the closing share price of the company and the loss limit (the amount that you are prepared to lose from your investment).

The system then processes this information and interrogates firstly, the PTA neural network with the previous two weeks data concerning profits from trades (an initial set of previous two weeks data is set up at the start). Secondly, the QSI neural network is interrogated with the last quarter's share index for the company.

The PTA and QSI networks were trained with different patterns that represented different movement trends. Figure 6.2 represents the training data for the PTA and QSI. Both the networks have been trained to output values between 0 and 1. These values represent whether the general trend of the input pattern is decreasing, denoted by values close to zero, remaining stable denoted by values around 0.5 and increasing denoted by values close to one.

Once the networks have been interrogated their trend values are returned to the expert system which then performs some post-processing to interpret the numeric values into linguistic categories (decreasing, stable or decreasing). This post processing is important because it allows the expert system to represent numeric information from neural networks in a form that can be understood and incorporated into the explicit rules of the expert system (i.e. via a fact). The results of both interrogations are shown on main system window.

The system then calculates the current risk, using factors such as the current profit trend, yesterdays profit trend, the current share index trend, the loss limit, today's profit, the day and the current month. Depending on this calculated risk and the accumulated total profit the system evaluates the position that the investor should take, that is, whether the

investor should increase, maintain, be cautious or withdraw his investment (and other variations of these). The system then displays the total profit to date and the current risk level. This is followed by some text recommending that the investor adopt the position which was calculated using the current risk and profit.

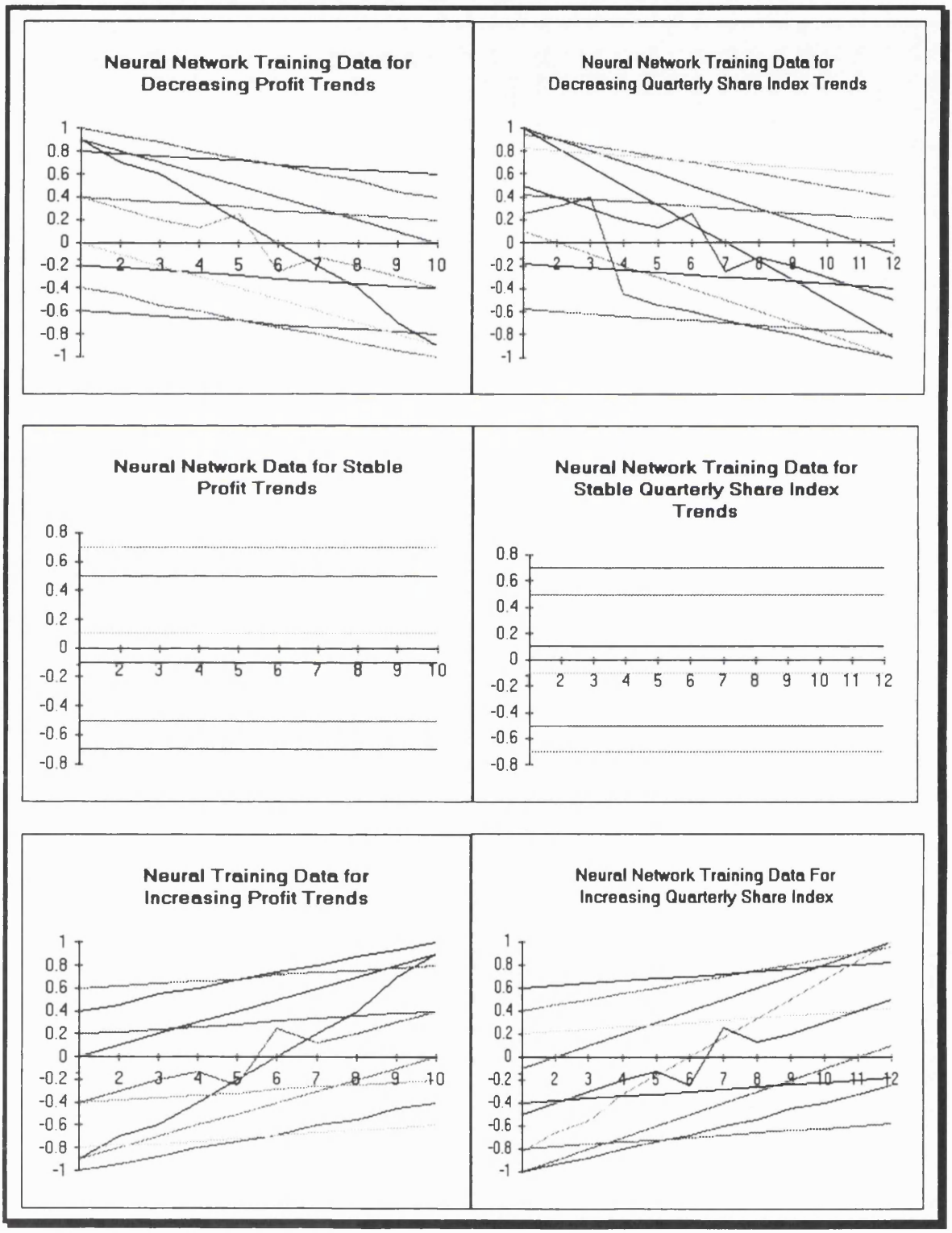


Figure 6.2 — Training data for PTA and QSI neural networks.

If this system was to be used in a real-world scenario then all the required inputs could be automated, the day and month could be retrieved from the host system, the closing price from a data feed and the loss limit calculated as a proportion of the profit to date. This would allow the system to be active and available for interrogation by an investment analyst. An important point about this relatively simple application is the fact that the domain expert who was consulted when constructing the knowledge base, had attempted (about five years ago) to construct a real-world version of this application with purely an expert system approach and had failed due to the difficulty in performing the pattern classification tasks via an expert system.

The object-oriented interface and communication methods of the ORBERON environment, performed efficiently during the implementation of this proof of concept application. This application served its purpose by validating the flexibility and extendibility of the ORBERON environment and showing that it allows the design, communication and execution requirements for building hybrid systems.

6.3. British Airways Cargo Consignment Application

The air freight market like the passenger market is rapidly expanding. It is estimated that the freight market will grow at a rate of 4.5 percent per annum. There is a trend towards deregulation within the air cargo business and this combined with the projected air freight expansion will result in additional optimisation and prediction needs in the 1990s.

To meet these growing requirements, British Airways is investing £150 million on new facilities at Heathrow airport. British Airways handles nearly 500,000 tonnes of freight ranging from gold bullion to tropical fish. This new Cargo Terminal will be 600,000 square feet and be complete at the turn of the century. The new Cargo operations will be using the latest technology to achieve maximum automation of cargo consignment and handling.

The importance of cargo loading and consignment in passenger aircraft is increasing. In 1986 the percentage of air freight volume which was transported in passenger aircraft totalled 67 percent. If the ratio of sold to offered capacity can be increased, then British Airways can increase operating profits. The problem that British Airways has to solve is the decision of whether to accept or refuse potential cargo. To achieve this, an air freight sales agent has to maximise net profit by accepting the ideal types of loads for any particular aircraft prior to take off.

The acceptance or rejection of a cargo requests, depends on quantitative and qualitative information about the particular cargo. There are also various technical air regulations (regulations on dangerous goods, maximum weight, maximum width and tariffs), along with general information about the cargo (whether it is livestock or perishable, it's weight to volume ratio etc.) that must be considered when decisions to accept or reject loads are taken. These factors may be interrelated and some even conflicting.

All these factors can cause real decision problems for the human sales agent, partly because acceptance is also dependent on qualitative factors such as the expectations of future air freight requirements prior to take off, certain trends that exist within certain flights or the importance of a particular customer. Although operations research methodologies have a long tradition within British Airways cargo business, these techniques cannot offer solutions to break the barriers of complexity involving both qualitative and quantitative decisions. The quality of the decision thus depends on the sales agent's own knowledge and experience, the time within which the decision has to be made, and the information available at the time.

A decision support system for cargo consignment would have to take into account the quantitative parameters such as density, volume, routing, flight time and the qualitative parameters like loss of choice for a customer and a customer's priority. These fundamental components of this problems lend themselves to a hybrid solution where an expert system could be used to handle the quantitative aspects and a neural network could handle the qualitative aspects of the problem domain.

Before examining the hybrid solution to this application, the Cargo Consignment problem is examined in detail and various existing systems highlighted.

6.4. Comparison of Cargo and Passenger Booking

The standard reservation profile for passenger flights is something similar to the graph shown in Figure 6.3, and the standard Cargo reservations profile is substantially different as shown in Figure 6.4:

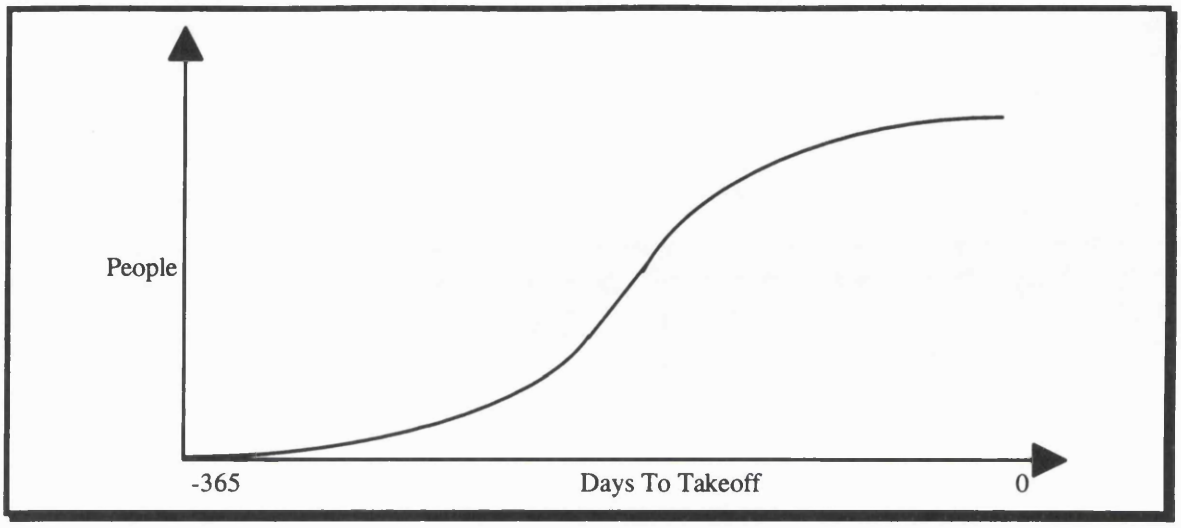


Figure 6.3 — A standard reservations profile for a passenger flight.

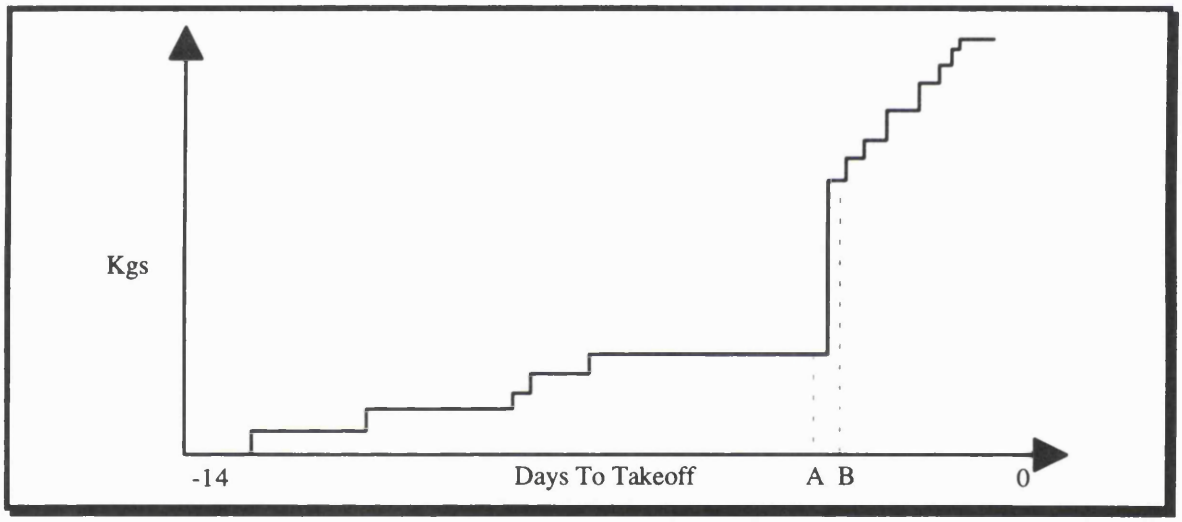


Figure 6.4 — A standard reservation profile for Cargo.

There are several differences and associated effects between the reservation characteristics or passengers and cargo:

1. The passenger profile covers 365 days where as the Cargo profile only covers 14 days. This means that there is far less time to react to the individual selling characteristics of a specific flight as they become apparent.

2. This effect is exaggerated by the shape of the Cargo profile (around 60% of cargo consignments are booked in the last two days).

3. The cargo profile is not smooth. Passengers, in general, book in ones and twos, where as cargo reservations vary from 10 kg's or less, to several tonnes and so the profile actually changes in a number of discrete steps. This also implies that

cancellation have a considerable effect on the cargo profile than cancellations do on the passenger profile.

4. On the passenger side, current booking levels are used to forecast final loads. The previous factors means that adopting a similar approach for cargo would result in wildly differing forecasts made just a few moments apart, such as at point A and B in Figure 6.4.

For a particular aircraft, it is known with some certainty how many seats will be available to carry passengers (although those seats can be arbitrarily divided into classes). Cargo management unit do not know until around an hour before departure, how much capacity will be available to them. Only lower-bounds figures are available with certainty.

The main variables with passengers are their numbers and the rates they pay. Cargo has extra dimensions in that weight, volume, monetary rate, and availability of appropriate containers all need to be considered. Also on constrained routes (where volume or weight are at premium), consignment density may be as important as the rate.

Changing carriers after a consignment is booked ("interlining") is not as prevalent for cargo (although this may be to Cargo's advantage) but is quite common for passengers to be flown by another airline if the booked flight is full.

6.5. Operational Background to Cargo Consignment

The operational goals of cargo consignment are to maximise profit by carrying the optimum mix of cargo products on a flight. There are many important operations that must be performed and many issues considered before a piece of cargo can be accepted or rejected. These operations concern Capacity Planning, Space Management and Yield Management.

Capacity Planning

Capacity planning involves the creation of Route Capacity Control Advice (RCCA) sheets for each cargo carrying flight. This forms the foundations for all the business conducted by the Cargo section, by providing the information as to the available space on the flight for the various Cargo products. RCCA provides information regarding the plane type, layout configuration and sector payload (i.e. the total weight and volume that can be carried on the plane). To calculate the available weight and volume for cargo, the passenger baggage, crew baggage, catering, mail and courier weights are subtracted from the current sector payload.

Block Bookings

A large proportion of freight carried in BA holds falls under the category of block bookings. These are bookings into reserved space on the aircraft with fixed rates. All the main agents and freight-forwarders make use of block space to guarantee that there will be space for their cargo. Block Bookings are divided into Mail, Courier and Permanent Bookings (PBs). Agents put in their requests for space based on their current level of freight and what they expect to sell for the following season. Decisions are made on basis of rate offered, weight carried and possibly other related factors such as customer service, through shipping etc. The main goal of Cargo section is to maximise profit by carrying the optimum mix of cargo products on a flight. PBs are generally reliable and place regular business with BA, but the rates for PB space are less than that obtained from free-sale.

The major problem is the poor utilisation rate of PB space. It was found that in 1994 the non-take up of PB space over all flights was 4 million cubic metres. Agents do not pay for the space, only for the cargo that is carried. No penalty is imposed for non-take up of space. Therefore, an agent is likely to put bids in to several carriers and take the lowest bid offered. Therefore it is important to monitor usage throughout the season to pull back space for free-sale.

Space Management

The overall objectives of space management may be divided into two parts:

- Maximise the total amount of space available for Cargo Business
- Distribute the mix amongst the Cargo products to ensure the maximum utilisation and yield from the available space, i.e. minimise the total amount of un-utilised space given the demand for Cargo on the flight.

Unused PB space is automatically released into freesale 18 hours from departure when it passes to Capacity Control. Space can be released earlier if outstations inform that the space will not be required.

Most space reclamation methods used now are reactive, i.e. they claw back space after it has been found that it will not be utilised. This is usually too late to make any significant sales into the freed-up space. Forecasting methods based on historical data and current trends could give an indication of the likely available space on departure and by taking the most pessimistic case it should be reasonably safe to take this space into free sale at a much earlier date.

Yield Management

The objective of Yield Management is to control space on key capacity constrained flights in order to maximise the revenue from the cargo carried. This is done by giving preference to high yield bookings over those offering lower yield.

Capacity Management Operation

Capacity management staff will look at all cargo which requires pallet or unit protection for weights over 300k. An average of 700 to 900 queue messages a day are processed. Their task is to check the weight and volume, and then reserve space for the cargo by writing down the consignment booked (using a program called PACKMAN to work out how many units are required) and then confirm the booking to the agent.

All bookings for yield controlled flights coming through a queue are initially processed by the automatic queue monitoring system ASP (Acceptable Shipment Package). ASP [37] looks at the queue every 5 minutes and processes any booking that it finds. ASP shows the target rate required to carry the cargo. It calculates using a weighting system whether or not to accept the booking. The Duty Officer will then make a decision based on the result from ASP and his own judgement and experience.

There are many factors that are not currently considered by the yield analysts when making a decision as to whether to accept a particular consignment onto a flight. This may be because either the information is unavailable or is only available through considerable time and effort on the part of the analyst. Information that would be important in the yield control process include :

- **Cost** - Cost information has an important bearing on the value to BA of a consignment. The costs include those for transport, handling, and trucking. A consignment with an apparently high yield may also have a mitigating high cost which could result in a greatly reduced contribution or even a loss.
- **Status Of Free Space** - Status of PB allotments usage, profiles on customer and agents.
- **Flight Trends** - Historical information as to how the flight filled over the last week, month or the same time last year.

The yield analysts develop their knowledge of booking acceptance through experience. This knowledge does not exist in any detailed written format. This may lead to individual ways of controlling bookings which are not consistent with those of other analysts. It also does not allow direct monitoring of a particular strategy for yield control.

If an experienced analyst becomes unavailable then so does his knowledge of how to do yield management.

6.6. Existing Cargo Consignment Systems

Although there has been many studies and systems [8, 50] relating to airline seat consignment there are only a few systems designed for the cargo consignment problem. A cargo consignment system needs to take into account a broader range of factors in order to reach a decision whether to accept or reject any freight. Some of these factors are outlined below in Table 6.1.

Important Factors	Characteristics
Physical Aspects	Weight, Volume, Dimensions, Nature Of the Goods, Destination Preferred Flight / Alternatives
Flight	Load factors last year, Load factors last few weeks, Rate trends, Booking Profiles, Current state of flight weight/volume availability
Monetary	Rate, Costs, Currency
Demand	Competitor rates, Total demand level time pattern, Product related demand, Demand changes over day, week, season
Route	Market served by route, Single Or Multi-leg, Bookings over more than one airline, Yield Management over routes, Network Optimisation
Customer	Size, Value, Importance, Reliability,
Product	Type, Rate offered, Demand, Conditions
Operations	Transhipping vs. Point To Point, Packed Units vs. Loose

Table 6.1 — Important Factors in Yield Management.

The following list details the current yield and capacity management systems in use by other airlines. These systems are all currently using expert system technology.

6.6.1. Lufthansa - CARGEX

CARGEX [13] is a decision support system for air freight processing. It was originally developed from work at the Koblenz School of Business Studies. It provides advice as to whether to accept or refuse potential air cargo. The system starts by assuming the freight will be accepted and tries to verify the quantitative and qualitative parameters to justify a refusal. The quantitative parameters include density, volume, routing, flight time. The qualitative parameters include customer's priority and loss of disposition freedom. It also takes account of the size of the market and the market trends. It then returns its conclusion to the sales agent using a window based graphical interface.

CARGEX is implemented on Symbolics workstations, and is to be integrated with Lufthansa's transaction oriented Unisys computer system.

6.6.2. SwissAir - ARGOS

Like the Lufthansa system, ARGOS [85] was developed from work carried out by the Koblenz School of Business Studies. It stores the current loading and capacity status of all flights and the details of all inquires received for additional consignments. It decides whether to accept a consignment according to four criteria

Physical Cargo Quality - This compares the consignment weight and volume with the capacity still available for the requested flight. Preference is given to consignments that achieve optimum use of the container or hold.

Monetary Cargo Quality - This compares the cost contribution margin with theoretical targets established for the route with preference to consignments according to how much they exceed the variable costs involved.

Market Trend - This assesses whether accepting the consignment may exacerbate a suspected bottleneck.

Customer Value - This assesses the value of the consignor as a customer.

These four assessments are carried out independently and then combined to produce an acceptance decision. The expert system can then give reasons for its recommendation. It also has a facility to supply a price at which a refuse recommendation becomes an accept and vice versa.

ARGOS has a high-resolution graphical interface to present the booking information, current capacity status and breakdown of cost information. SwissAir claim that they are looking at automating at least 40% of all booking enquiries using this system [85].

6.6.3. Cargo Lux - CHAMP

Cargo Lux claim that CHAMP is the only fully integrated industry standard cargo system available. It consists of a series of large databases accessed by 700 programs that provide integrated control over the following areas :

Space control	Operations	Ground Handling	Revenue
Accounting	Receivables	Settlement	Management Reports
Proration	Interline	Marketing	Rates

A key feature of this system is the use of the costs database to give accurate figures as to the marginal cost contribution of each shipment. This is linked to a report generation system that can give breakdowns as to the value of the cargo carried on a flight, route, time or agent basis.

The development costs were $10 million. Cargolux claim that it repays its investment every year with savings of $4 million in manpower, errors and payments and an increase in revenue of about 3 per cent gained from the marketing information about customer shipping patterns and effects of price adjustments.

Currently about nine airlines use the system including North West, Fin Air, Air Canada and Aeroflot. Unisys sell CHAMP's revenue accounting module as part of its USAS cargo system.

6.7. Hybrid Cargo Consignment Solution

Within the cargo consignment problem there are many variables to be considered, ranging from physical aspects to monetary constraints. These considerations are made worse by the short timescales involved in booking cargo and the number of requests for an ever decreasing amount of space. Therefore, a large airline like British Airways has a large amount of revenue dependent on the accurate judgement of its cargo booking sales staff. Performing this task well is worth hundreds of millions of pounds annually. Therefore, even small improvements in accuracy are extremely valuable.

The characteristics that make this application suitable for a hybrid solution using expert systems and neural networks are :

1. the problem has components that are well understood and static, as well as poorly understood and dynamic components.
2. a large number of factors interacting with each other in a complex fashion influencing outcomes and decisions.
3. some of the relationships are probabilistic and not directly expressible as rules.
4. some of the relationships are logical and well established, therefore they can be represented as rules.
5. the airline environment is a constant process of change, making adaptive behaviour necessary.

6. the large number of daily departures, each with known outcomes, provides a large database required for an adaptive system to learn effective behaviour.

The complex pattern of interacting factors makes it difficult for cargo experts to articulate factors that influence the decision making process. This combined with the large number of potential interactions and the non-linear decision factors makes the problem unsuitable for a purely rule-based approach. Even if an experienced analyst could identify all the factors required for an accurate judgement about cargo consignment, the large and unwieldy rule base needed to represent this knowledge would be difficult to update if conditions changed. The same difficulties also pose problems to conventional operations research and statistical approaches.

Therefore, it becomes apparent that probably the best approach to cargo consignment problem would be a combination of approaches. An expert system could be used to make logical decisions about the request, and to co-ordinate neural networks that could be used to classify patterns.

This is the approach that was adopted, where the expert system was used to accept the inputs, show the outputs and implement a knowledge base that encoded an algorithm based on the British Airways ASP (Acceptable Shipment Package) [37] weight-scoring system. British Airways use ASP to generate a point score for each cargo request and if this is above 100 they accept the request. The hybrid weight-scoring system was loosely based on the ASP algorithm devised by British Airways, because substantial effort and expertise had already been invested in devising the algorithm. The weighting system that has been encoded substantially improves on the ASP algorithm. Unfortunately, ASP does not take into account the full range of factors that yield analysts would consider. ASP has no concept of flight trends or historical information about how the flight fills. It doesn't reject on the basis that it is likely to get a better offer at a latter date. Another factor is that the points scoring system used to judge bookings may be too brittle to be capable of subtle fine-tuning. To take these factors into account, neural networks were trained on past cargo requests, to see if they could spot acceptance or rejection patterns for various destinations. These took account of factors such as how many days to take-off, the availability of space with regard to space required, and the monetary differences between the standard rate and the offered rate. The neural networks and expert system results were combined to give the final accept or reject decision.

In the following sections, the British Airways data for the problem, the design and implementation of the expert system components, the neural network components and final results are detailed. In Chapter 8, these results are assessed against the results from the original ASP system.

6.8. British Airways Cargo Consignment Data

The aim of the cargo consignment solution was to provide better quality decisions for accepting or rejecting cargo for the Australian region. The data that was supplied by British Airways covered cargo requests for flight BA0011 to Sydney that stopped at Auckland (AKL), Melbourne (MEL), Brisbane (BNE), Wellington (WLG) and Sydney (SYD). Table 6.2 descriptions the data that was available the Cargo Consignment problem.

Field	Data	Description
Flight No	BA 0011	Flight Number - London to Sydney
Flight Date	11/7/91	Flight Date
AWB	12561458320	Air Waybill - Unique code for each consignment
Station	OSL	Station requesting the consignment
Origin	LHR	Origin of Consignment
Destination	AKL	Destination of Consignment
Available Weight	2734	Available Weight on the plane
Available Volume	21.19	Available Volume on the plane
Standard Rate	2.1	Standard Rate for a consignment to a particular destination. Fixed at the start of the two seasons Summer (March-Sept) and Winter (Oct-Feb). Generally, a request would only be accepted if the rate offered (consignment rate) at least matches the standard rate. However if the flight isn't filling then a lower rate may accepted.
Consignment Weight	1092	Consignment Weight.
Consignment Volume	1.19	Consignment Volume.
Consignment Rate	2.5	Consignment Rate Offered - May be zero if in UK. In these cases the standard rate applies.
THRU	N	If N that the rate per kg, else Consignment Rate is per pallet.
Consignment Score	109	Score given by the ASP Algorithm (Accept if >= 100).
Recommendation	A	ASP Algorithms Recommendation (Accept or Reject).
Decision	A	Final Decision of BA Yield Controller.
Date Recieved	11/4/91	Date request was logged on to the system.
Time Recieved	736	Time the request was logged on to system.
Date of Decision	11/4/91	Date that the BA Yeild Controller made the final decision.
Time of Decision	750	Time that the BA Yield Controller made the final decission.

Table 6.2 — Description of the British Airways Cargo Consignment Data.

The supplied data was the actual data set that was compiled for use with the original British Airways ASP algorithm. It compromised of 2400 records for the BA0011 flight to the Australian region, starting from the 5th of November 1991 to the 2rd December 1992. The data required some initial pre-processing to eliminate duplicates and erroneous entries.

Upon detailed analysis of the data (e.g. by sorting by flight departure and then within that by destination and then by the date the request was received) it was discovered that there was insufficient data per flight departure date to build profiles of demand or build up a picture of how space is utilised for each flight. From this analysis it was also obvious that there was a large amount of cancellation of requests for space, but unfortunately this information was not explicitly available in the data. Upon enquiry to British Airways, it was discovered that the cancellation information was never actually recorded and that the fine-grained flight information was available somewhere on the British Airways mainframe machine but it was not cost effective to retrieve. Although this was a restriction on the detail that the hybrid solution could address, it was still possible to solve the problem of maximising space usage, by making the cargo acceptance decision procedure more efficient at the level of flight destination instead of at the level of individual flights.

6.9. Design and Implementation of the Expert System Component

The expert system component is built using the CLIPS programming shell described in chapter 5 and implements a system that takes into account a number of factors relating to the cargo booking, such as the offered rate and urgency of delivery. These are given a weighting relative to their importance. The level of the factors combined with their weighting are summed to give a total score for the booking request. If this score exceeds a set threshold then the booking is accepted; if it falls below the threshold then it is rejected. As explained earlier this is the method employed by ASP (Acceptable Shipment Package) [37], currently used by British Airways.

This approach is relatively simple to encode and easily comprehensible to the user. However, it has proved hard to fine-tune the weight settings with the result that there is frequent disagreement with ASP when the points score is in the borderline region between Accept and Reject. This current threshold standard score is set at 100.

The expert system component was designed based on the ASP algorithm and with help from domain experts, the ASP weight settings were improved. In conjunction to this fine-tuning, neural networks trained on the Auckland, Melbourne, and Sydney cargo requests were used to improve the performance of this weighting system for the borderline cases.

The following fragments of pseudo-code outline the very basic functionality of the weighting system used within the expert system. For the listing of the complete expert system see Appendix D. Firstly, the main decision making rules for accepting or rejecting the cargo requests are outlined.

```
If Weight of Consignment > available weight Then
     REJECT
If Volume of Consignment > available Volume Then
     REJECT

Consignment Score = ( Weight/Volume Total  x  weight/volume weighting +
                      Density total        x  density weighting +
                      Rate total           x  rate weighting +
                      Urgency total        x  urgency weighting )
                    * 100 / standard score

If consignment score >= standard score Then
        recommend ACCEPT
else
        recommend REJECT
```

To calculate the requested cargo consignment point score, various totals relating to the weight/volume, the density, the rate and urgency are calculated and weighted by appropriate weighting factors. The basic threshold decision making process is then implemented, where if the consignment score is greater than the standard score (currently 100) then the consignment is accepted, otherwise it is rejected. Due to the brittleness of this threshold level the hybrid solution concentrated on using neural networks for the borderline decision making. This also meant that the majority of the decision making could be easily conducted via the expert system and the borderline cases (where the score is close to 100) accounted for by comparison with previous borderline cases. The interaction of neural networks and expert systems is explained later in this chapter. The following pieces of pseudo-code explain how the totals needed to calculate the consignment score are worked out.

Weight/Volume Total

```
If (flight departs within 48 hours) and (available weight > 1000) and
   (available volume > 5) Then
      weight/volume total = 1.5
else if (flight departs in more than 48 hours) and
        (available weight < 1000) and (available volume < 5) Then
         weight/volume total = 0.5
     else
         weight/volume total = 1.0
```

Density Total

In the following example, if the available density is greater than 167 (threshold set by British Airways Cargo Consignment officers) then the flight is considered volume-restricted. This means that volume is more at a premium then weight so the system should prefer small volume, high weight consignments to maximise the space usage. If the reverse is true then the flight is weight-restricted.

```
If ( available density > 167) Then
 {
  If ( consignment density > available density )
      density total = 1.5
  else if ( consignment density <= 167 ) Then
           density total = consignment density / 167
       else
           density total = 1 + 0.5 * (consignment density-167) /
                                      (available density - 167)
 }
else
 if ( consignment density >= 167 ) Then
         density total = 1.0
   else
   density total = maximum(0.5,1+ (consgmt density - 167 ) /
                                  (available density - 167))
```

Rate Total

The rate total is calculated from the difference between the consignment rate and the standard rate. The difference between the rates, determines the rate total. The rate total has been a weak point for the original ASP algorithm. This stems from the fact that when the requesting station is in the UK, the consignment rate can often be zero. In these cases the standard rate applies. The original ASP algorithm scored these requests very low because it assumed the rate was actually zero. The method implemented actually corrects this problem and achieves better results. The basic ASP difference calculation is as follows:

```
Calculate diff = (Consignment rate - Standard Rate )
diff = ( > 50p )         Rate Total = 1.6
diff = (30p to 49p )  Rate Total = 1.4
diff = (20p to 29p )  Rate Total = 1.2
diff = (10p to 19p )  Rate Total = 1.1
diff = (-9p to 9p )    Rate Total = 1.0
diff = (-19p to -10p )  Rate Total = 0.9
diff = (-29p to -20p )  Rate Total = 0.8
diff = (-49p to -30p )  Rate Total = 0.6
diff = ( < -50p )  Rate Total = 0.4
```

Urgency Total

The original ASP calculation for the urgency total only took the urgency of a consignment into account, as follows:

```
If consignment is urgent
      urgency total = 1.2
 else
      urgency total = 1.0
```

Where as the hybrid solution also takes into account factors such as the importance of any favoured cargo requesting stations and makes all their requests urgent.

The complete expert system component for the hybrid British Airways solution is given in Appendix D. The full expert system and weighting factors were designed with the help of experts at British Airways Artificial Intelligence Unit and the Cargo Consignment department. It incorporates:

- the input of cargo consignment information from the keyboard (or a file for testing purposes),
- the setup and co-ordination of neural networks for classification of cargo trends for Auckland, Melbourne and Sydney routes, including the scaling of inputs to the neural networks from the expert system,
- the assertion of the results returned by the neural networks, into the expert systems factbase so that they can be used by the expert system,
- the improved weighting system including the use of results of the neural networks during the accept/reject decision-making process,
- finally, there are printing utilities for outputting the results and statistics of the hybrid systems operation.

Once the cargo consignment information has been input, the expert system calculates the consignment score via the weighting system, and depending on the destination, the inputs for the neural networks are scaled with the appropriate factors and then presented to the appropriate network. Once the results of the neural network have been placed into a form the expert system can use (a fact) the consignment score and the neural networks are used to make the final decision about accepting or rejecting the cargo request. This decision is explained by the following pseudo-code but the actual rules (Check-Weight1-2, Check-Volume1-2, and Consign-Decision1-3) can be found in Appendix D: (note that the standard score is currently 100)

```
If (consignment weight > available weight) or
   (consignment volume > available volume) and
   (consignment score is >= (standard score - 5)) and
   (consignment score is <= (standard score + 5))
Then
   (System Recommendation comes from the appropraite Neural Network)

If (consignment score > (standard score + 5))
Then
   (System Recommendation is ACCEPT)
Else If (consignment score < (standard score - 5))
     Then
        (System Recommendation is REJECT)
```

```
Else If (consignment score is > (standard score - 5)) and
        (consignment score is < (standard score + 5))
     Then
        (System Recommendation comes from the appropraite Neural
         Network)
```

These rules use the weighted consignment score to make a decision if there is a clear margin for acceptance or rejection. If there is a clear decision based on the consignment score then the expert system recommendation is used. If on the other hand, the score is on the borderline between accept or reject, then the neural network is employed to give a second opinion. The neural networks are used in this manner because they have been trained to classify previous accept/reject decisions for a particular route, and can take into account previous acceptance/rejection trends, and also take into account any patterns of demand or cancellation that may exist. The next section outlines the design and training of the neural networks used in the hybrid system.

6.10. Design and Implementation of the Neural Network Components

The neural network components within the hybrid solution to the British Airways cargo consignment problem consisted of three standard Back-propagation neural networks [92], one each for Auckland, Melbourne and Sydney destinations. The network simulator used to build these networks was described in Chapter 5. The neural networks were used to learn the mapping between previous cargo consignment details and the final decision of the cargo acceptance officers. The networks were used to build a profile of cargo acceptance and rejection for a particular route, so that they could be used to make decisions about new requests based on previous trends.

6.10.1. Data Preparation, Scaling, and Network Topology

Much of the success of neural networks has been due to their ability to learn from raw data. The importance of pre-processing the raw data is often forgotten but can make the difference between a network that trains in five days and performs quite well and a network that trains in five minutes and performs very well.

As explained earlier, the British Airways data had insufficient information per flight to build profiles or forecasts of demand or build up a picture of how space is utilised for each individual flight before departure. This meant that neural networks had to be trained at the destination level, not the more detailed flight level. Therefore the aim was to train three neural networks on the previous cargo acceptance/rejection characteristics of the three most popular routes (Auckland, Melbourne and Sydney).

From the original BA data three data sets for Auckland, Melbourne and Sydney were separated. These data sets were constructed because each destination may have individual characteristics. These were pre-processed to remove duplicates and any outliers in the data. From these data sets the *training* and *validation* sets were chosen for each neural network. The training set contained representative members and the actual output to be trained upon for all possible decision classes (i.e. representatives of accepts and rejects). Within each class an adequate variation was included. The validation set is also drawn from the same population as the training set but is not used for training but for validating the results of training. Due to the small number of rejects in each route, the training and validation sets were small (ideally they should be larger). The Auckland network training data consisted of 100 training requests and 22 for validation, the Melbourne network consisted of 74 training and 21 validation and the Sydney network had 96 training and 21 validation requests. For all destinations, the training and validation sets consisted of 50% accepts and 50% rejects.

Both the training and validation sets were scaled because the desired output must be within a range realisable by the activation function used within the Back-propagation network. As the logistic activation function $f(x)=\frac{1}{1+e^{-x}}$ is used in the feedforward neural networks, output activation limits should be 0.9 and 0.1 respectively. In theory the inputs need not be scaled, but stability of the network is improved by using comparable limits. Also uniform scaling will also equalise the importance of input variables. A linear scaling was used to map the variable's (input and desired output) practical extremes to the networks practical extremes (0.1 to 0.9). To scale a variable V to a value A, the simple formula below was used [64]:

$$A = r(V - V_{\min}) + A_{\min}$$

$$= rV + (A_{\min} - rV_{\min})$$

where $r = \frac{A_{\max} - A_{\min}}{V_{\max} + V_{\min}}$ and $V_{\max}$ and $V_{\min}$ are the maximum and minimum values of input variables and the $A_{\max}$ and $A_{\min}$ are the practical limits of the neural network (0.9 and 0.1) respectively. The validation set was also scaled using the same values for $V_{\max}$ and $V_{\min}$ as the training set. This was done to achieve a uniform input range for the networks. This scaling was also performed within the rules of the expert system (for each route network), so that cargo consignment inputs could be passed to the networks appropriately scaled.

The topology of the network (i.e. the number of input, hidden and output nodes) was determined by the input variables, the number of decision classes and monitoring the

training and validation errors in the case of hidden nodes. The input layer contained 6 nodes, one for each of the following inputs: available weight, available volume, consignment weight, consignment volume, rate difference (consignment rate - standard rate) and the days until the flight. As the output decision is a binary classification of Accept or Reject there is only one output node. Where an activation of 0.9 represents Accept and 0.1 represents Reject.

The selection of hidden nodes is an important aspect of the design of a neural network. The activity patterns in the hidden nodes or "learned-feature detectors" [111] is an encoding of what the network determines are significant features of the data. With too many hidden nodes, a network can simply memorise the correct response to each pattern in its training set, instead of learning a general solution. By limiting the size of the hidden layers, the network is forced to develop appropriate feature detectors to efficiently classify large sets of input patterns. These general-purpose feature detectors are more likely to correctly classify new inputs, and therefore the network would perform better.

There are many "rules of thumb" and trail by error procedures for determining the required number of hidden layers [40, 111, 64]. The procedure used was one of starting with a small number of hidden nodes and evaluating the performance against the validation set. Then adding one more node and re-evaluating the performance until no improvement could be made. This is the procedure recommend by Masters [64] to find the best number of hidden nodes. The final number of nodes that gave the best performance was 5 hidden nodes. The final network topology was 6 input nodes, 5 hidden node and 1 output node.

The networks were then trained using the supervised back-propagation learning algorithm, with the initial weights set between -0.5 and +0.5, learning rate set to 0.1 and momentum set to 0.9. These parameters were not experimented with but were chosen from the results of other researchers [92] who found they produced good results for back-propagation networks.

Table 6.3 shows the final results of the percentage of correct classification for the training and validation set for the three route networks.

	Percentage Correct on Training Set	Percentage Correct on Validation Set
Auckland Network	97%	81.81%
Melbourne Network	95%	69%
Sydney Network	96%	72.6%

Table 6.3 — Final results showing the performance of the three route networks.

As an experiment to see if the different routes had different cargo acceptance characteristics, the test set for the Auckland network was tested on the Melbourne and the Sydney networks. The Auckland test data on the Melbourne network scored 67% where as on the Sydney network, it scored only 60% correct. These results showed a marked drop in performance from the normal score for the network, which could point to there being route specific characteristics for accepting and rejecting cargo requests.

6.11. Results and Future Improvements for the Hybrid Solution

When the original training and validation sets were constructed, another test set was constructed from the three route sets chosen from the original BA dataset. This set was used as the test set for the hybrid solution. As it was extracted at the same time as the training and validation set it was independent of the data used for the neural network training. This test set consisted of 297 requests (234 accepts, 63 rejects) for Auckland, 199 requests (163 accepts, 36 rejects) for Melbourne and 296 requests (246 accepts, 50 rejects) for Sydney. These 792 requests were placed randomly into file and arranged into the following order: days until flight, requesting station, destination, available weight, available volume, standard rate, consignment weight, consignment volume, consignment rate, and the final human decision.

The hybrid system reads this file (although the system can accept input from the keyboard), opens the three neural networks, processes the input and produces a system recommendation for accepting or rejecting the request based on the criteria outlined above. Figure 6.5 shows the start and end of the final test run produced. The operation of the hybrid solution shows the hybrid system loading the neural networks, reading the batest.txt file, reporting the results of the neural network, then making the hybrid recommendation and finally reporting the percentage correct so far within the test file.

```
LOADING APPROPRIATE NEURAL NETWORKS INTO ENVIRONMENT
Please wait .....

*****************************************************
**  BRITISH AIRWAYS CARGO          **
**    CONSIGNMENT SYSTEM           **
*****************************************************

Accept Data from a File (y/n) : y
Enter Name of the File to Read ("*.txt") : "./../roff/BA/batest.txt"

DTF= -4, STA= DUS, DEST= AKL, AVLW= -1318, AVLV= -4.00, STR= 2.10 CONW=
1050.00, CONV= 12.38, CONR= 2.23
>>>> Interogating AKL Neural Network <<<<
Neural Network Result REJECT
Consignment Score => 94.78

[ System Recommendation is REJECT the request ] [ Actual Decision was R ]
[ 1 correct out of 1 ==> 100.00% ]

DTF= -6, STA= BCN, DEST= AKL, AVLW= -919, AVLV= 3.00, STR= 2.10 CONW= 203.00,
CONV= 0.32, CONR= 3.35
>>>> Interogating AKL Neural Network <<<<
Neural Network Result REJECT
Consignment Score => 106.05

[ System Recommendation is REJECT the request ] [ Actual Decision was R ]
[ 2 correct out of 2 ==> 100.00% ]

..... system continues until the end of the test set .....
..... last entry from the test file showing final result .....

DTF= -6, STA= VIE, DEST= SYD, AVLW= 12261, AVLV= 33.00, STR= 2.10 CONW=
1491.00, CONV= 13.75, CONR= 2.20
>>>> Interogating SYD Neural Network <<<<
Neural Network Result REJECT
Consignment Score => 88.99

[ System Recommendation is REJECT the request ] [ Actual Decision was R ]
[ 664 correct out of 792 ==> 83.84% ]
```

Figure 6.5 — The hybrid British Airways cargo consignment system in operation.

The test file had also been run on a purely expert system implementation of the British Airways ASP algorithm and achieved a success rate of 62.5% correct classification. The final results of the system showed that the hybrid solution achieved a success rate of 83.84% correct classification. This is an improvement of 21.34% over the purely expert system based approach. These results are assessed in the next chapter and conclusions drawn about their relevance.

Although the results showed that adopting a hybrid approach have significantly improved the quality of the decision making process for British Airways Cargo officers, there are still possibilities to improve the use of neural networks and expert systems within this application. If the quality of the British Airways data could be improved, then there are even more possibilities to improve the results of the system by performing some forecasts of demand.

As a flight opens for bookings 15 days before departure, this point could be used to forecast trends on historical data such as:

- Calculating the available space for sale: physical, overbooking & likely space released from passenger and permanent bookings utilisation
- Setting up booking acceptance profiles that dictate the rates for the available Cargo products.
- Setting up booking profiles for monitoring booking trends.

If more data was available a picture or profile of how the space builds up on each flight before it departs, could be constructed. This could be used to assess factors such as:

- how often does the flight fill before departure, does the flight become weight or volume constrained; if so how early.

- how often were consignments rejected which would have improved the revenue of the flight

Flight profiles for weight, volume and rate could be generated using neural networks. These could then be used as a basis for a system which could be given the current weight and volume availability, and the volume and weight of the current consignment and then the system could report how the consignment fits along the flight profile (e.g. if better then would normally be expected, then accept the consignment, or if the consignment does not fit the profile then reject the consignment, etc.). These types of improvements could easily be made if the appropriate level of detail was available in the British Airways data.

6.12. Summary

This chapter presents the application of the ORBERON environment to solving two hybrid problems. This has demonstrated the effectiveness of the environment's object-oriented design and shows that adopting a hybrid approach produces better results than applying a single processing technique to a problem.

The first application was in the financial arena of Profit Trend Analysis and was used to test the interface and communication functionality of the ORBERON environment. This application showed that the object-oriented environment performed well and was suitable for building hybrid systems. The second application was a real-world Cargo Consignment problem from British Airways. The results from this hybrid application showed that the ORBERON environment can also be applied to complex real-world problems and also highlighted the fact that a hybrid approach produces better results than any single processing approach.

Chapter 7

The Design and Implementation of the HANSA Framework

This chapter details the design and implementation of the object-oriented framework that is at the heart of the ESPRIT III HANSA (Heterogeneous Application geNerator Standard Architecture) project. The HANSA framework and the project as a whole were inspired by the research presented in this thesis.

7.1. Introduction

The research detailed within this thesis was the inspiration for the design and implementation of the European community funded, ESPRIT HANSA (Heterogeneous Application geNerator Standard Architecture) project. The motivation for the HANSA project was the need for an application development environment for the emerging multitude of windowing and operating systems, and the recent trend towards multi-platform computing. The HANSA project was initiated by the author of this thesis, to address this challenge for heterogeneous application development. The basic design concepts and the philosophy of the HANSA Framework were inspired by the components found in the ORBERON environment and the hybrid approach presented in this thesis. The main difference in the HANSA approach was to use current and emerging industry standard communication protocols and sophisticated programming techniques (such as client-server programming) to produce an object-oriented environment that could operate on many platforms for the integration of different processing tools.

The HANSA project has designed and implemented an object-oriented cross-platform framework to allow developers to rapidly configure industry standard utilities and artificial intelligence shells for marketing, banking, insurance and executive information systems (EIS) applications. The project is co-ordinated by Thorn EMI CRL, with University College London (UCL) as associate partners. The other European partners include: Brainware, IFS and ARTS Consultants from Germany; Mimetics and Promind from France; and O.Group Srl. from Italy.

The HANSA project uses object-oriented techniques, to develop domain-specific application generators, a Toolkit of application specific and industry standard tools, and a generic framework for their combination (as shown in Figure 7.1). The application generators are being constructed by the European partners in the business areas of direct marketing (Thorn EMI CRL), banking (Brainware, IFS and ARTS), insurance (O.Group) and executive information systems (Promind and Mimetics).

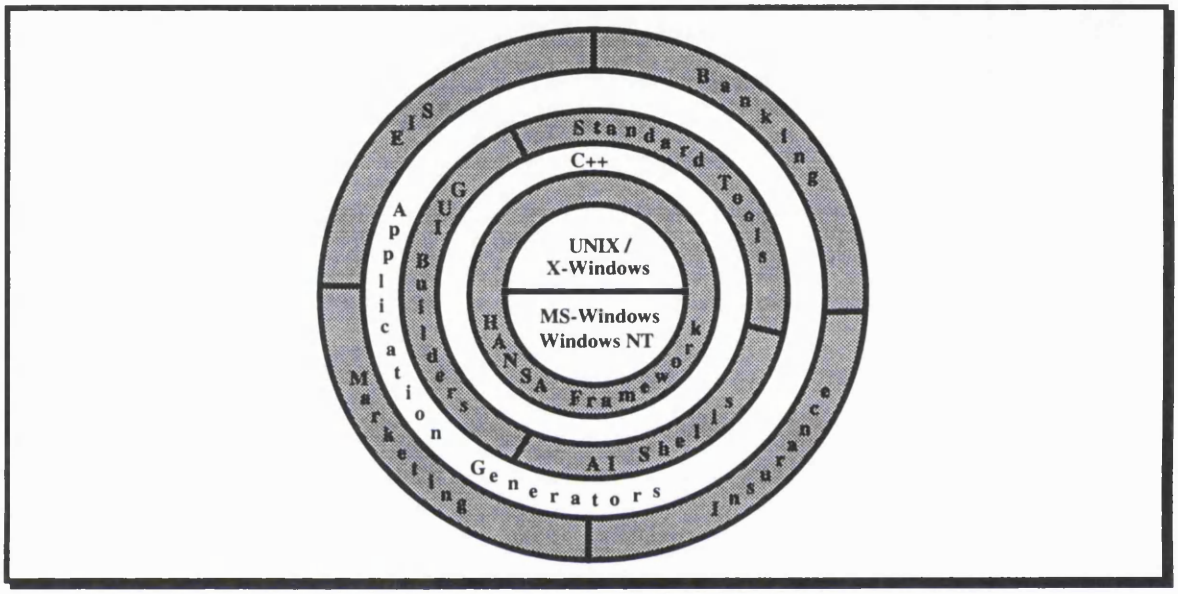


Figure 7.1 — The HANSA framework.

The HANSA Framework provides the support for integrating industry standard utilities (spreadsheets, databases, etc.) and novel artificial intelligence techniques (neural networks genetic algorithms, etc.). The framework offers facilities to generate immediate hybrid systems by providing rapid application generation capabilities and a communications architecture which is compliant to industrial standards such as OLE [18] and CORBA [38].

The HANSA framework is an object-oriented cross-platform communication architecture that allows processing components within hybrid applications to exchange information. The framework hides machine-related implementation details and promotes software reuse and rapid prototyping. On each of the target platforms (PCs running MS-Windows 3.1/NT and UNIX/X-Windows workstations) the framework implements a services interface inspired by, and compatible with, Microsoft's OLE protocol. For UNIX/X-Windows systems, HANSA is developing an OLE-like interface following closely the philosophy of the CORBA from the Object Management Group, by building the framework on top of the services supplied by the Object Request Broker. The CORBA compliance is archived via SunSoft's ToolTalk protocol [104]. The platform-independent

application programming interface allows easy porting of hybrid applications from one platform to another.

Figure 7.2 shows that a typical HANSA application is composed of: one HANSA Manager (an application generator), responsible for controlling the configuration and execution of an application; and one or more HANSA Tools, the basic utilities that can be selected from a collection of HANSA compliant tools, found in the HANSA Toolkit. The HANSA Manager stores it's controlling and configuration information in an entity know as the *Blueprint*. As the name suggests this is a description of the hybrid application in terms of the tools used, their configurations, how they are linked together and the ordering of data exchange between tools.

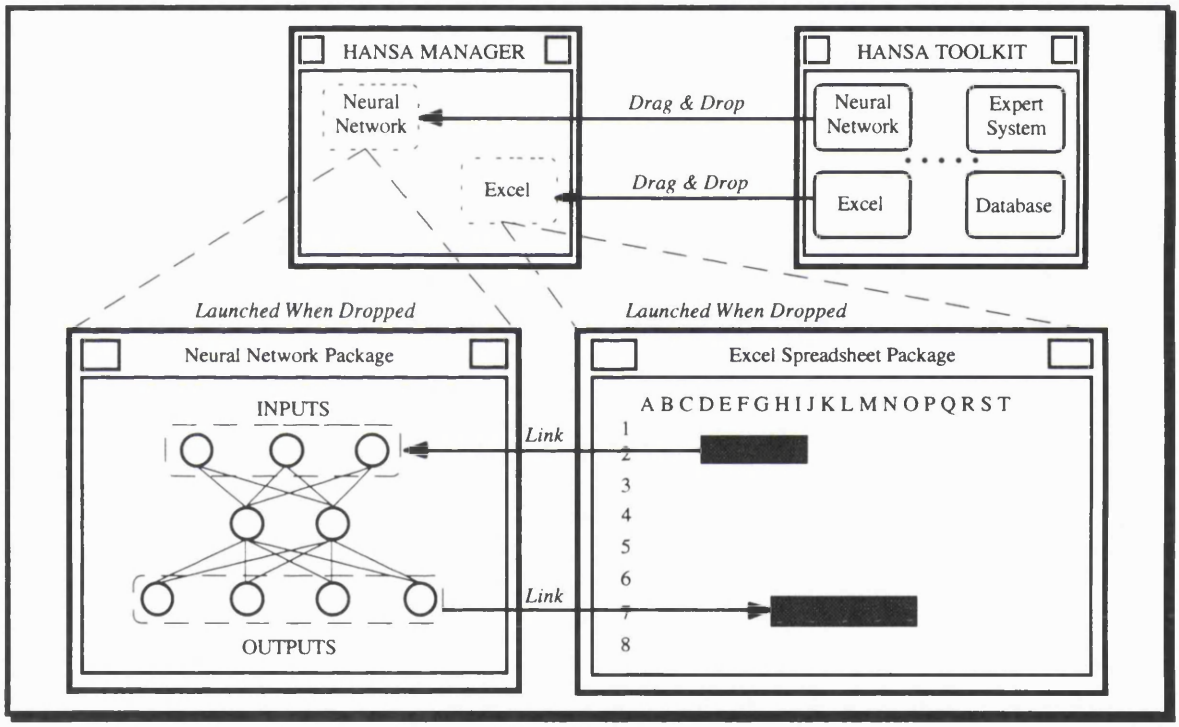


Figure 7.2 — A typical HANSA application generation procedure.

For each application domain there is at least one HANSA Manager in charge of controlling the configuration and execution of co-operating HANSA Tools. The HANSA Toolkit includes: neural network development environments such as MIMENICE from Mimetics [72], a Genetic Algorithm programming environment called QGAME from University College London [58]; as well as industry-standard utilities (such as Excel and Access from Microsoft) that comply with the HANSA or the OLE/DDE communication protocols on PCs or UNIX workstations.

By allowing the integration of such a diverse range of Tools, HANSA constitutes a natural programming environment for the implementation of hybrid applications.

Moreover, the possibility of using *off-the-shelf* tools, allows the user to configure and tune the application by utilising the most appropriate utilities for each task in the problem.

7.2. HANSA Framework

The HANSA Framework is the *virtual* architecture that supports HANSA applications. Its components are the *HANSA Communication Architecture* (HCA) and the *HANSA Registration Database*. Figure 7.3 shows the operation of applications within the communication architecture and the interaction with the HANSA Registration Database.

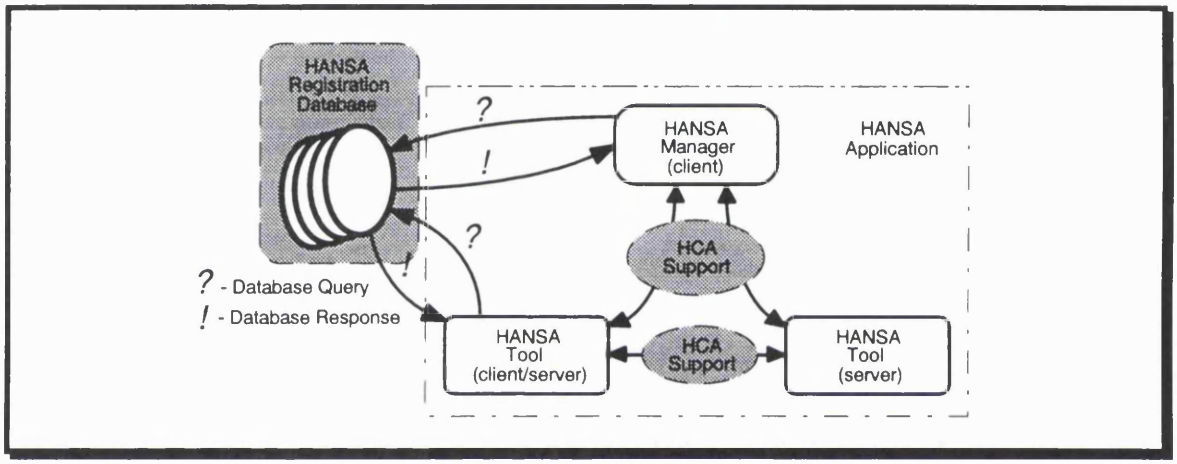


Figure 7.3 — HANSA operation.

The Communication Architecture is the infrastructure for configuration and data exchange within an application. The HANSA Registration Database is the holder of all information concerning HANSA Tools (software that is specifically targeted at HANSA applications, or available commercially for one particular platform). The HANSA Registration database acts as the repository of information about tools available to the user, in a similar manner to the Dynamic Environment Manager within the ORBERON environment presented in this thesis.

A developer of HANSA applications views the HANSA Framework through an *application programming interface* (API) that is common to all platforms targeted by the project. Figure 7.4 shows that the programming interface comprises of a set of classes divided in two groups : one group implements the Communication Architecture (with its communication protocols); and the other group implements utility support classes to allow, for example, the manipulation of the Registration Database.

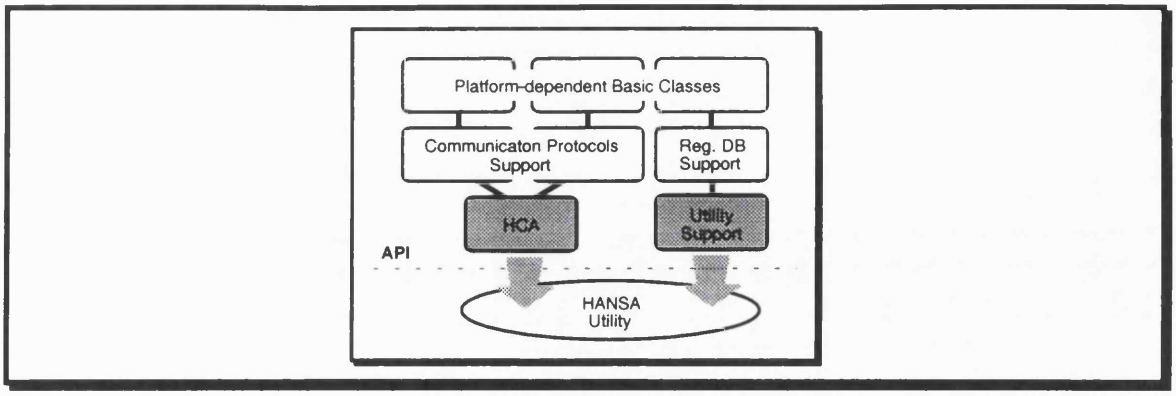


Figure 7.4 — HANSA Framework Architecture API Class Structure.

The classes hide all implementation and platform-specific details. Typically, HANSA Applications are constructed by one domain-specific *application generator* (or *HANSA Manager*), and one or more HANSA Tools. HANSA Managers simplify the integration and configuration of Tools within a specific domain. Each manager implementation determines the amount of work demanded from the user for the configuration of Tools and the final application. Developers can offer *black-box* applications that don't allow changes to a pre-defined configuration, or open-ended application generators, which will leave most of the configuration and linking of Tools to the user.

An application stores all configuration information in a *HANSA Blueprint*, which is managed by one of the application domain Managers. The information stored in the Blueprint is platform-independent and comprises: the name of the Tools composing the application; the specification of any data or Tool links, as well as the order in which these links are to be re-produced when the application is run; and the necessary configuration and Tool identification details to be used when activating each one of the Tools.

One of the tasks to be executed by a Manager is to recover all platform-dependent details to complete the information contained in the Blueprint. This platform-dependent information is held by the Registration Database.

7.3. HANSA Communication Architecture

Figure 7.5 shows that the HANSA Communication Architecture comprises three communication protocols structured in two layers: application, and framework.

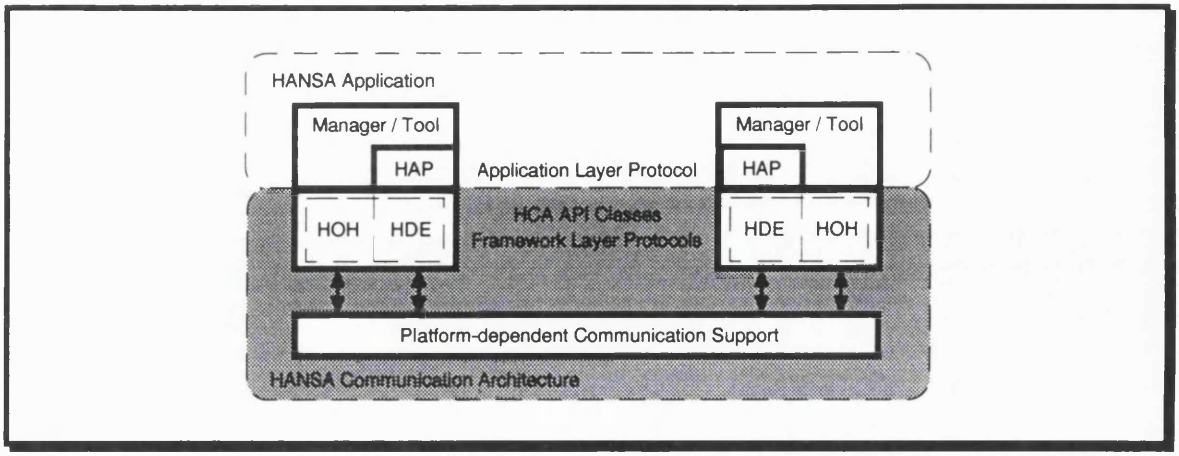


Figure 7.5 — HANSA communication architecture.

The communication protocols are the *Object Handling* (HOH) and *Data Exchange* (HDE) protocols, in the framework layer, and the *Application Protocol* (HAP), in the application layer. All protocols are implemented on top of a platform-dependent communication support hidden from the developers by a set of application programming interface classes. These constitute the platform-independent API that allows code to be portable across the HANSA platforms.

The communication architecture is based on industry standards, reflecting one of the requirements of the HANSA Framework. The most important of these standards is the *Object Linking and Embedding* protocol (OLE) developed for Microsoft Windows™ 3.1. The Object Handling Protocol is an implementation of OLE-compatible protocol on the platforms targeted by HANSA (PCs and UNIX Workstations). An important contribution of HANSA is the implementation of an OLE-style protocol on UNIX platforms.

The choice of OLE for one of the basic supporting technologies is a direct result of the desire to achieve *rapid prototyping* of applications, and to allow the use and integration of industry standard packages. It also implies the choice of a client/server architecture. In this context, HANSA Managers are full clients, while Tools can be simple servers, or client and servers simultaneously.

The communication protocols of the HANSA Framework realise the basic support for data exchange, and for configuration and control of application and Tools. To allow the integration of industry standard Tools and HANSA utilities, these protocols have to reflect the highest level of these industry standards. Specific requirements of HANSA applications are dealt with by the higher level protocol definition — the Application Protocol (HAP).

The Registration Database provides a mechanism for the integration of commercial utilities and HAP compliant Tools. In reality, this mechanism is a map that can be used to translate the HAP commands (which are known only to HANSA utilities) to those commands that are understood by commercial utilities. Using the HDE protocol for the transmission of these commands will guarantee the functionality of the HANSA Utility, independent of the versions of the HCA protocols.

7.3.1. HANSA Object Handling Protocol (HOH)

The Object Handling Protocol is aimed at generic data exchange, configuration, and utility execution control. It is an extendible protocol that permits one utility to use the services of another. Through the Object Handling Protocol a HANSA Manager can use HANSA Tools for specific tasks.

The HOH protocol is the supporting mechanism for handling links between Manager and Tools, and for dealing with point-to-point execution requests and data exchange. HOH can also be used as the transport mechanism for user defined messages.

The HANSA Object Handling Protocol is the HANSA implementation of Microsoft's OLE protocol. As such, utilities complying to HOH will also fully comply to OLE. In Microsoft Windows, HANSA utilities will be able to act as OLE servers and/or clients.

In UNIX/X-Windows, HANSA utilities will present the same functionality as its correspondent in MS-Windows. Moreover, the HOH API, in its final version, will be compliant to the *Common Request Broker Architecture* (CORBA).

The utilities developed using the functionality of the Communication Architecture API, along with the conformance with the OLE protocol, benefit from a platform-independent source code that is portable across HANSA platforms.

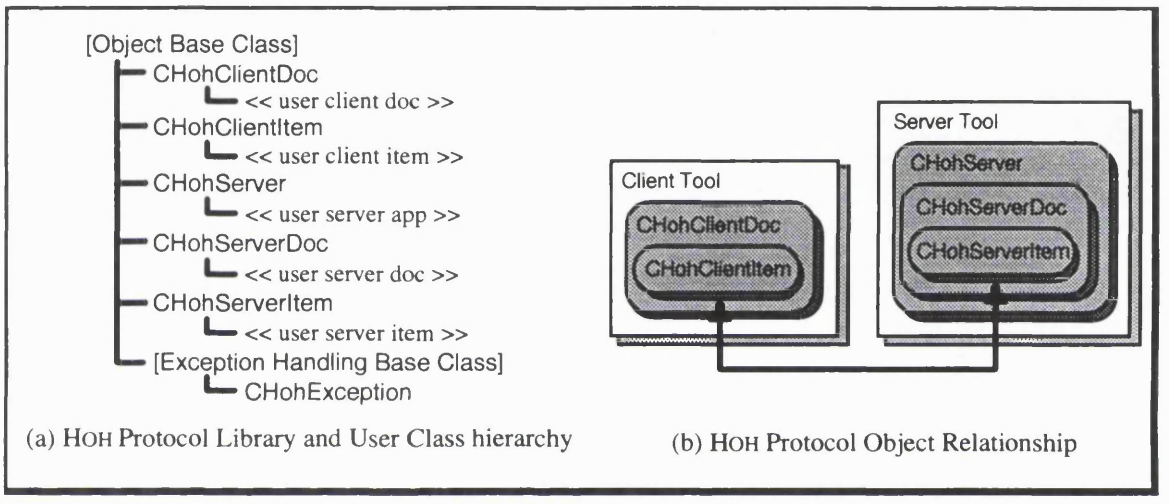
7.3.1.1. HOH Protocol API

The HOH Protocol API is a version of the OLE-specific classes found in the Microsoft Foundation Classes for Windows. The same API is present in all HANSA platforms and represents a common interface to the implementation-dependent details of the Communication Architecture. It also allows a smooth transition to new versions of OLE by hiding the complexity of the protocol.

The class hierarchy of the API implementation is shown in Figure 7.6a (HOH client and server). Figure 7.6b shows the object relationship in a typical application.

In this class hierarchy, a *document* is the **C++** object that holds and manages *items*. An *item*, in turn, is a **C++** object that manipulates the actual data. Figure 7.6 also shows that for each client utility (Manager or Tool), the developer must create one document class for each document that the utility supports (e.g., Blueprint in the case of a HANSA Manager), and one item class for each linked data item (within a Tool or Manager).

For server utilities, the developer must create a server class for each type of item supported[1]; one document class for each type of item supported; and one item class for each link or embedded data that is active.



(a) HOH Protocol Library and User Class hierarchy

(b) HOH Protocol Object Relationship

Figure 7.6 — HOH Client member functions: UNIX API Version 1.0.

Note that CHohClientItem and CHohServerItem are not two different categories of items, but different interfaces to the same item (one for the client document, one for the server document).

The HOH Protocol API is complex and extensive. The whole set is necessary to achieve compatibility with MS-Windows standard OLE applications. However, not all utilities will need the whole of the functionality available. For example, a simple HOH Client (a simple HANSA Manager), capable of managing a simple HANSA Blueprint with some links with Tools, uses circa 28 member functions of the over 100 existent. Figure 7.7 lists version 1.0 of the UNIX HOH API.

[1] For example, if graphs and worksheets are supported, the developer must create one server class for each one.

Class CHohClientDoc
::CHohClientDoc() ::AssertValid() ::Dump() ::DeleteContents() ::GetStartPosition() ::GetNextItem() ::OnNewDocument() ::OnOpenDocument() ::RegisterClientDoc()
TOTAL: 9

Class CHohClientItem	
::CHohClientItem() ::CheckCreate() ::CopyToClipboard() ::CreateFromClipboard() ::CreateFromFile() ::CreateLinkFromClipboard() ::CreateNewObject() ::CreateStaticFromClipboard() ::DoVerb() ::Draw() ::GetBounds()	::GetNextItem() ::GetRuntimeClass ::GetType() ::InWaitForRelease() ::OnChange() ::Serialize() ::SetBounds() ::SetHostNames()
TOTAL: 19	

Figure 7.7 — HOH Client member functions: UNIX API Version 1.0.

A similar analysis of a simple server utility results in an equivalent percentage of used API class member functions. The minimum set is composed by approximately 30 member functions that reflect the server equivalent of each of the member functions used in a client utility.

7.3.2. HANSA Data Exchange Protocol (HDE)

The HDE protocol is the low-level message transport protocol of the HANSA Communication Architecture. In contrast to the HOH protocol, all details of the communication, including the management of connections, have to specifically programmed.

As happens in the HOH protocol, a HDE server is the provider of information to a HDE client. Server and clients have *conversations* about *topics*, and exchange data related to *items*. A conversation is an exchange of information related to a specific item of a topic. For example, a Tool that retrieves quotations from stock markets can support multiple stock markets, and multiple stocks within that market. The topic names that the Tool uses can be *NYSE*, *FT100*, etc. The item names will identify stocks such as ICL, Siemens, etc.

To be able to exchange information in a conversation, both HDE client and server have to be aware of the available topics and items. The HANSA Application Protocol defines some specific topics and items for HANSA Tools.

A HDE HANSA utility must support at least the *system* topic. The system topic, together with its items, can be used to get information from HDE servers to establish conversations. Standard items within this topic are, for example:

- *status*,
- *item data format list*,

- *item list* (non-system items), and
- *topics list* (non-system topics).

The HDE protocol also defines the possible *transactions* that can occur during a conversation. A transaction is a request and associated acknowledgement. Examples of typical transactions are:

- *connect*, used to establish a conversation with a client;
- *execute*, to send a command string to a server; and
- *request*, to request data from a server.

All communication has to be done through the existent transactions defined by the HDE protocol API. All utilities implementing HOH through the Communication Architecture API support the HDE *connect* and *execute* transactions automatically. HDE is inspired by and compatible with Microsoft's DDE protocol. As such, utilities complying to HDE also comply to the DDE protocol.

In Microsoft Windows, HANSA utilities will be able to act as DDE servers and/or clients. In UNIX/X-Windows, HANSA utilities will present the same functionality as in MS-Windows. The utilities developed using the functionality of the Communication Architecture API, along with the conformance with the DDE protocol, benefit from a platform-independent source code that is portable across HANSA platforms. Moreover, the HDE API, in its final version, will also be compliant to the *Common Request Broker Architecture* (CORBA).

7.3.2.1. HDE Protocol API

The HDE Protocol API is defined in a similar way to the Object Handling Protocol API. A set of classes with associated member functions hides all the implementation details of the protocol. In basic terms, the HDE API comprises a set of classes that implement the server and client sides of each conversation. Figure 7.8 shows the class hierarchy of the API.

Client utilities using the HDE protocol are capable of managing multiple, simultaneous conversations. As can be seen in Figure 7.8, for each client HDE conversation, there will be a different instance of a CHdeClientConv object. In a similar way, a server can also take part in multiple conversations. With the Object Handling Protocol, a client needs one item for each link with a server. With the Data Exchange Protocol, clients can exchange information about multiple items through the same communication link.

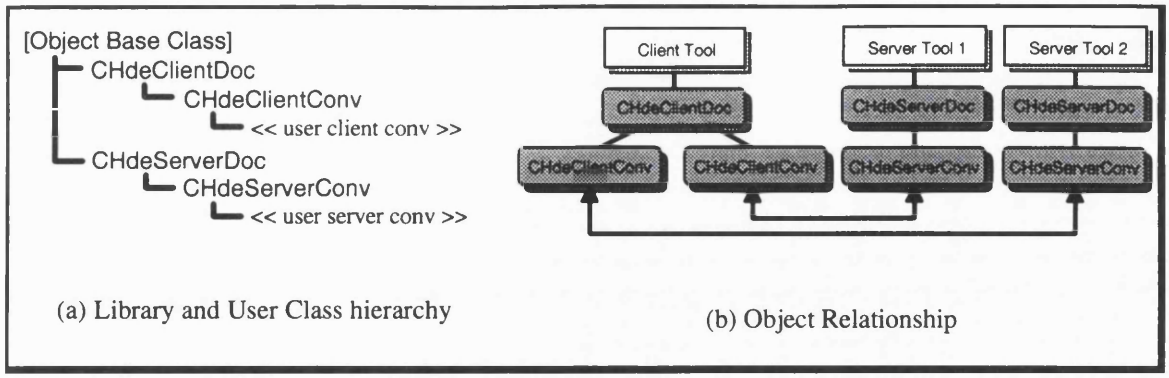


Figure 7.8 — HDE Protocol API Class Library Usage.

Figure 7.9 shows that the HDE API comprises 2 classes (CHdeClientConv and CHdeServerConv) and a total of 9 member functions. The simplicity of the API is a consequence of the simplicity of the protocol, particularly if compared to HOH.

Class CHdeServerConv	
::CHdeServerConv() ::Advise()	
TOTAL: 2	

Class CHdeClientConv	
::CHdeClientConv() ::Execute() ::Initialize() ::Link()	::Request() ::Send() ::Unlink()
TOTAL: 7	

Figure 7.9 — HDE Client and Server member functions: API Version 1.0.

7.3.3. HANSA Utilities Support API

The HANSA Framework API includes some additional classes used, for example, for file input and output, for manipulating the HANSA Registration Database, and for creating generic *documents* and *items* that present similar functionality to those used in the Communication Architecture API.

The most important of these additional classes is CHansaDB. This class encapsulates all the necessary functions for accessing and managing Registration Database entries. Utilities create an object of this class, and use it to retrieve and store information about Tools installed in the HANSA Toolkit. The CHansaDB class member functions are listed in Figure 7.10.

The utilities support API also includes some auxiliary functions and macros to help in the implementation of HANSA utilities. It also defines data types of parameters and returned values that are used in the implementation of Communication Architecture API.

Class CHansaDB			
::DeleteToolID	::GetHCAProt	::SetActivationStr	::SetToolID
::GetActivationStr	::GetLinkDir	::SetHAPCommand	::SetToolName
::GetHAPCommand	::GetToolID	::SetHAPMap	::SetToolPath
::GetHAPMap	::GetToolName	::SetHCAProt	
::GetToolPath	::InstallTool	::SetLinkDir	
TOTAL: 18			

Figure 7.10 — HANSA Database Class member functions: Support API Version 1.0.

7.3.4. HANSA Application Protocol (HAP)

The HANSA Application Protocol (HAP) is an application integration protocol that defines a high-level standard Tool message set. HAP is, in fact, a collection of HANSA-specific conversation topics and item definitions to be used in conjunction with the HDE protocols. These topics and item definitions can only be used for communication and control of HANSA Tools, or industry-standard Tools for which a map of commands has been stored in the Registration Database. This high-level protocol is similar to the message passing communication protocol implemented within the object-oriented hybrid environment detailed in this thesis. Both protocols implement a set of messages that allow high-level access and control of independent processing techniques. Both protocols represent interfaces to the major functions within the processing techniques within their respective integration environments. The HAP protocol in addition implements a standard set of messages that all HANSA tools should support.

The main purpose of the Application Protocol is to standardise on a set of control instructions and data formats that are common among HANSA Tools and HANSA Managers. The desire to benefit from existing protocols and *macro* languages used by Microsoft Excel and Mimenice is one of the motivations for the definition of HAP. A HANSA developer implements HAP conversations on top of the basic HDE protocol. Alternatively, the basic API functions of the HOH protocol can be used.

The HAP specification has been joint work of HANSA partners. To date, 25 commands have been defined in three groups: Control, Data, and Graphical Interface functions.

Control Functions

CheckLink	Continue	DeleteLink	Execute	Initiate
Interrupt	SaveState	SetLink	RestoreState	Terminate

Data Functions

GetData	GetObjList	GetOpList	LoadExt
NewDoc	SaveExt	SetData	SetObjList
SetOpList			

Graphical Interface Functions

GetWinState Resize	Maximise SetWinState	Minimise	Move

7.3.5. HANSA Registration Database

The HANSA Registration Database makes information about objects handled by servers available to all client utilities. It also allows the location of server Tools when their services are requested by a client

A server Tool must register itself during installation, or when it is started for the first time. One of the objectives of this registration process is to define the level of compliance to the HANSA Communication Architecture protocols.

The Registration Database is the platform-dependent component of the HANSA Framework. Figure 7.11 shows the structure of the Database.

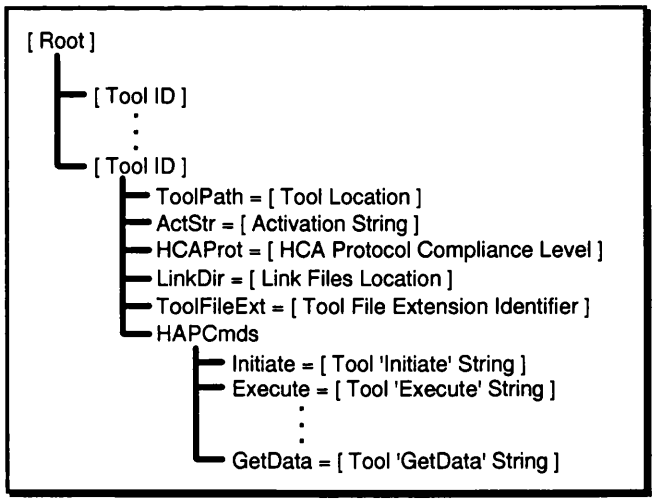


Figure 7.11 — HANSA Registration Database Structure.

Each Tool is registered in the database with a unique *ID* that is used as the search key. The entries in the database are:

- ToolID – the identification of the Tool in the database;
- ToolPath – the location of the executable code;

- ActStr – the command line for activating the Tool;
- HCAProt – the compliance level of the Tool to the HCA protocols;
- LinkDir – the system directory where the link files are stored;
- ToolFileExt – the identifier of Tool data files; and
- HAPMap – the translation map between Tool-specific command strings and those defined by the HANSA Application Protocol (this entry is only present if HCAProt indicates compliance with the Application Protocol).

Along with the direct benefits that the HANSA Framework brings in terms of development support, applications developed to the HANSA specification present strong commercial advantages:

1. HANSA applications allow user customisation in contrast to *black-box* business applications found in the market today.
2. HANSA Tools easily integrate with other commercial utilities. This allows the use of HANSA Tools in applications not targeted by the HANSA Project.
3. The HANSA Project adds value to all the technology in which its implementation is based by either adding functionality to the technology, or by porting it to new platforms.
4. The reusability of Tools simplifies the developer's task, who can then concentrate on specific aspects of each application (control, configuration details, data exchange, etc.).

The integration and configuration of Tools within a specific domain are to be simplified by the application generators (the HANSA Managers). Each developer will be able to choose from offering *black-box* applications, or user-configurable application generators.

The final success of the HANSA Framework is to be assessed when the project nears completion in October 1995, in a number of ways:

1. An application developed to the framework specification must be easily portable across HANSA platforms.
2. The framework must support the implementation of application generators (HANSA Managers) that allow user-controlled application configuration, customisation and extension.

3. The framework must also support the exchange of application components (HANSA Tools) of similar functionality under user control without consequences on the application behaviour.

4. When compared to identical stand-alone applications, the framework must not impose unacceptable computational overheads. This is to be accessed in terms of response time to user requests.

7.4. Distributed HANSA

With the success of the HANSA Framework the project has been extended until October 1995 and the planned future developments of the HANSA Framework will involve the design and implementation of a *distributed* framework that will allow applications to be built from tools that reside across a network of different computers. This will allow applications to distribute their computationally expensive tools on powerful workstations and show the results on PC's.

With the availability of WinSocks (an implementation of Berkeley UNIX sockets for the MS-Windows environment), it is now possible to implement the HDE protocol using the TCP/IP communication mechanism. This will enable applications running on UNIX/X-Windows platforms to be able to "talk" to applications running on MS-Windows 3.1/NT. This will enable HANSA applications to be distributed across PC's running MS-Windows and workstation's running UNIX/X-Windows.

The work will comprise of the following major components:

- The design and implementation of HDE using sockets for both MS-Windows/PC and UNIX/Workstation.
- The design and implementation of gateways to enable the seamless interoperability between the DDE, ToolTalk and TCP/IP versions of HDE
- The design and implementation of an application launcher to facilitate remote starting of applications.

If the results are satisfactory, the HANSA Framework and it's distributed counterpart, will improve the responsiveness of the application development process. A direct consequence of which will be the reduction of cost and time to develop hybrid applications.

7.5. Summary

With the growth in popularity of PC's and UNIX workstations, and the trend towards multi-platform applications, the HANSA Project aims to produce an object-oriented cross-platform environment for rapidly integrating standard tools and novel adaptive techniques. The basic components and the project concepts have been inspired by the research detailed in this thesis.

The design and implementation details for the HANSA Framework are shown and compared to components of the ORBERON environment. The object-oriented structure of the HANSA communication architecture is shown and the various communication protocols (HOH, HDE, HAP) are described. Finally, the design for a distributed version of the HANSA Framework, which would allow networked PC's and UNIX workstations to "talk" to each other, is outlined.

Chapter 8

Assessment

This chapter assesses the research work detailed in this thesis. The hybrid classification scheme, the ORBERON environment and the British Airways cargo consignment problem are assessed and compared against alternatives.

8.1. Target Review

The goal of this research has been to explore the fundamental issues in developing hybrid systems and to construct an environment that allows the pragmatic investigation of intelligent hybrid systems. Throughout this thesis, the field of hybrid systems has been analysed with respect to both the fundamental and the practical issues involved with integrating different processing techniques.

The foundation of this research has been the analysis of the main characteristics of symbolic and adaptive processing techniques. By analysing the theoretical and operational characteristics of these fundamentally different processing styles, it becomes clear that these processing methods have very complementary strengths and weaknesses. To highlight the complementary strengths and weaknesses, the issues and the pragmatic reasons for integrating symbolic and adaptive techniques, a detailed examination and comparison of expert systems and neural networks was carried out. These two techniques represent the most developed and prominent techniques from the respective symbolic and adaptive processing approaches.

Using this analysis coupled with a detailed examination of the processing requirements of real-world problems (such as knowledge acquisition, brittleness, high-level and low-level reasoning and explanation), it has been shown that there are three central arguments for hybrid systems: *Technique Enhancement*, the *Multiplicity of Application Tasks* and for *Realising Multi-functionality* within one system. In analysing the strengths and weaknesses of symbolic and adaptive techniques it is possible to highlight their complementary nature and the potential for creating a hybrid system that will overcome the limitations of one or both techniques. This concept of *technique enhancement* can produce superior hybrid reasoning systems. The analysis has also shown that the complex

nature and *multiplicity* of processing tasks within real-world applications requires the synergy of symbolic and adaptive processing characteristics. The other major motivation for developing hybrid systems has been the *realisation of multi-functionality* by unifying symbolic processing via an adaptive architecture, as found in the human brain. These types of single architecture, multi-functional hybrid systems could provide a means to understand and model human cognitive processes.

From this analysis a theoretical framework utilising a classification scheme to clarify and categorise hybrid systems with regards to functionality, communication and processing architecture was developed. This classification scheme allows for the coherent and qualitative evaluation of different hybrid systems. From this theoretical background and from practical experience gained during this research, a set of guidelines in the form of a development cycle for hybrid system is presented. The development cycle gives a standard procedure for designing and constructing hybrid systems. The development cycle breaks down the design and implementation processes into the following six stages: *Problem Analysis, Property Matching, Hybrid Category Selection, Implementation, Validation and Maintenance.* This would help an implementor by guiding them through the process of breaking the problem into sub-tasks, finding what techniques can solve these sub-tasks, how the techniques will interact, and finally the best methods for implementing, validating and maintaining the hybrid system.

During the development cycle of a hybrid system, a developer has to make a choice about an implementation strategy for the system. This research has shown that adopting an object-oriented approach offers many advantages with regards to design, communication and execution autonomy of different processing techniques. This research work also highlights how object-oriented programming can be used to implement the various types of hybrid systems defined in the hybrid classification scheme. Each processing technique can be represented as an independent object, and these active processing techniques can communicate with each other by passing messages through structured object interfaces. There are also many other software engineering related advantages of using object-oriented methods. These include code reusability, easy maintenance of software and a natural parallel mapping. It is these advantages that are now being recognised by major software vendors, who are producing object-oriented applications, operating systems and network communication software all using object-oriented components.

Upon establishing some theoretical background to the area of hybrid systems and offering practical guidelines in constructing hybrid applications, this research focused on the design and implementation of the ORBERON environment for integrating independent communicating symbolic and adaptive techniques. The techniques that were implemented

were expert systems and neural networks. These two techniques were used as a testbed to specify the interface, functionality and architectural requirements of *intercommunicating* hybrid systems and to develop the structures required to manage and manipulate different processing techniques within a hybrid environment. Adopting this object-oriented approach within the ORBERON environment allows for flexibility in operation and extendibility for the incorporation of new processing techniques.

Finally to test and assess the ORBERON environment and the hybrid philosophy in general, two applications were implemented using the environment. The first application was the Profit Trend Analysis problem and was mainly used to test the functionality of the object-oriented structures within the environment. This proof of concept application showed that the object-oriented structures within ORBERON provided the design, communication and execution autonomy required by hybrid applications. The second application was a real-world problem from British Airways, and showed that the hybrid approach to the complex Cargo Consignment problem faced by British Airways, can be more effective than the existing single technique (expert system) approach.

In the following sections, the main areas of this thesis work, namely the hybrid classification scheme, the ORBERON environment and the British Airways Cargo Consignment Problem, are assessed and compared with alternative methods.

8.2. Hybrid Classification Scheme

The hybrid classification scheme presented in this thesis was formulated primarily to address the issue of defining hybrid systems. As the field of hybrid systems expands, many conflicting interpretations of terminology are beginning to appear. This hybrid classification scheme resolves this conflict by categorising hybrid systems with respect to important factors found in all hybrid systems, such as the system's functionality, processing architecture and communication requirements. Apart from this clear definition of hybrid system systems into *intercommunicating*, *function-replacing* and *polymorphic* classes, the scheme also provides a means of comparing existing hybrid systems and provides a frame of reference to compare future hybrid systems.

Since this classification scheme was first reported in 1992 [36], several researchers in the field of hybrid systems have adopted it to describe their own research [97, 96, 63]. This classification scheme has also formed the basis for an edited book titled "Intelligent Hybrid Systems" [35] by the author of this thesis and Suran Goonatilake, a fellow researcher at University College London. This book brings together leading researchers and presents their work as clear examples of each class of hybrid system.

As with any classification scheme, there are situations where cases cannot be easily classified. These "border-line" cases can exhibit characteristics which can make the hybrid system appear to fall into more than one category. For example, a hybrid system where a neural network replaces the pattern matching function of an expert system, clearly has function-replacing characteristics, but nevertheless one can represent the neural network and expert system as quite distinct, separate processing modules. These modules can intercommunicate with each other and therefore, also appear to exhibit the characteristics of an intercommunicating hybrid system.

To resolve these border-line situations, the emphasis must be shifted from the mechanisms of communication, to the functionality of the system. This emphasis on functionality is highlighted in the definition of the hybrid classes. For example intercommunicating hybrids are characterised by the fact that they are independent processing modules that perform separate functions within an application and communicate their results to each other. The previously mentioned example of border-line categorisation only arises if the communication aspects of the classes are emphasised. If on the other hand the example is viewed from the functional aspects of the application, then the neural network and expert system in the example, are clearly not solving independent functions from the application domain. Therefore, the example is clearly a function-replacing hybrid because functionally the neural network is replacing a principal function of the expert system, despite the fact that there is a strong element of intercommunication. The majority of hybrid systems fall neatly into one of the three classes, in a small number of cases a detailed functional analysis of the application characteristics will be required.

Since the publication of this scheme there has been only one other hybrid classification scheme, which was developed by Medsker and Bailey [68]. This classification scheme was specifically concerned with classifying expert system and neural network hybrids. Medsker and Bailey's classification scheme has five categories : *stand-alone models*, *transformational models*, *loose-coupling models*, *tight-coupling models* and *full-integration models*.

The first category of stand alone models are defined as systems that have independent neural networks and expert systems both solving the *same* particular task, completely independent of each other without any exchange of information. Medsker and Bailey say that these types of systems can provide redundancy in processing in case one of the techniques fails to produce a result. It is debatable whether this category of system is actually a hybrid system. Firstly, because there is no exchange of information or interaction between the two techniques and secondly, because they are actually only solving the same

problem. Even Medsker and Bailey show some reservation on this category by declaring "standalone models are a degenerate case for integration purposes" [68].

Transformational models form the second category and are described as similar to stand-alone models but different because transformational models begin as one type of system (e.g. neural network) and end up as another (e.g. expert systems). There are two types of transformational models: expert systems that are transformed into neural networks, and neural networks that metamorphose into expert systems. Medsker and Bailey describe an example of such a system, where a neural network can be developed to identify trends and relationships within sales data and then that neural network can be used as the *basis* for an expert system. An expert system was targeted as the delivery system because it is useful to verify the knowledge and to justification capabilities. They also say that while less common, expert systems can be transformed into neural networks by using the expert system to set the initial conditions and training set for the neural network learning process.

The complexity of the transformational models is clear because Medsker and Bailey give no clear example of how such systems could be used and also say that "Limitations to transformational models are significant. There is no fully automated way of transforming an expert system into a neural network and vice versa. In fact, there is no known method for accurately and completely performing the transformation". For such transformational models to be viable would need a significant advance in our understanding of the knowledge representation and manipulation in neural networks. The only transformation that is similar is the extraction of rules from neural networks to be used to form an expert system [94, 31].

The third Medsker and Bailey category is loosely-coupled models which are defined as separate neural networks and expert systems components that communicate via *data files*. Examples of such systems are when neural networks are used as pre-processors or post-processors for expert systems. On inspection this category is very similar to the intercommunicating hybrid class.

The fourth category, tightly-coupled, is exactly the same as the loosely-coupled models except that they use memory resident data structures as opposed to external data files. These models also consist of separate processing models that exchange information and again the typical examples include neural networks as pre-processors for experts systems.

Both of the above loosely-coupled and tightly-coupled categories fall into the intercommunicating hybrid classification. The key definition of the intercommunicating

class was the multiplicity of application sub-domains, and both of these models can be used to solve different aspects of the application domain, regardless of whether they use files or memory structures for information exchange. In these two categories Medsker and Bailey use purely implementation details as the basis for the definition of hybrid classes. This is not very useful because it ignores the functional properties of the application domain. It is only by having a functional viewpoint of hybrid systems that users can conceptualise and understand the benefits of building such systems.

The last category in the Medsker and Bailey classification scheme is called fully-integrated models. This is the same as the polymorphic class in our scheme. The fully-integrated models are expert system/neural network models that share data structures and knowledge representation. Communication between the different components is accomplished via the dual nature (symbolic and neural) of the structures.

In summary, Medsker and Bailey have completely missed the notion of function-replacing hybrids, because their definition of hybrids has been based on implementation and communication issues and not taking into account the functional aspects of applications. Our classification scheme is simpler (three categories, as opposed to five) and does not contain redundant classes (such as the transformational category), thereby reducing the complexity of applying the classification scheme. Because our scheme is based on the functional aspects of applications and hybrid systems (e.g. using one techniques to overcome the limitations of another), it is more useful to developers than one based purely on implementation details of the hybrid system.

8.3. Object Orientation and the ORBERON Environment

With virtually every major provider and user of computer systems in the world today is planning to implement object-oriented tools and applications it is not surprising to find that major systems vendors (such as Sun Microsystems, HP and DEC), software developers, and user organisations, have joined together to form the Object Management Group (OMG). The OMG's main goal is to promote object-oriented programming and to provide a common architecture called the Object Management Architecture (OMA) incorporating the Object Request Broker (ORB) [38] for distributed object-oriented applications. The adoption of this architecture will make it possible to develop hybrid applications that can be distributed across all major hardware and operating systems.

The OMG defines object management as software development that models the real-world through the representation of "objects". These objects are the encapsulation of attributes, relationships and operations of identifiable program components. A key benefit of an object-oriented system is its ability to expand in functionality by extending existing

components and adding new objects to the system. Object orientation results in faster application development, easier maintenance and reusable software. This acceptance of object-oriented technology has reinforced the arguments presented in this thesis for using object-oriented integration to build hybrid systems.

In assessing this object-oriented approach to hybrid systems, this section will introduce key state-of-the-art integration technologies that use object-oriented methods for communication and co-ordination of components, and compare components of the ORBERON environment with components now appearing in these new technologies. These comparisons are at a purely conceptual and functional level. Although the components that are now beginning to appear in commercial object-oriented systems are technically better than the structures present in the ORBERON environment, they are still conceptually similar in their functionality to the components within the ORBERON environment.

8.3.1. Comparison With Current Integration Methods

In response to this object-oriented modular approach, the software industry is moving towards small specialised applications (or applets - little applications) and is moving away from monolithic packages that try to do everything moderately well. Using small, modular, reusable applications, allows the "mixing and matching" of tools within an application. If the user likes a particular text editor but needs a more powerful drawing feature, then he or she can just change the drawing tool, whilst keeping the text-editing tool.

To support this "mixing and matching" of applets, requires a new style of user interface and underlying model to allow the applets to interact. The emerging trends in object-oriented software that support these requirements are the Document Oriented Interface (DOI) and the client server model of interaction [62]. The DOI is an advanced user interface that makes documents, not applications, the focus of application development. The client server model of interaction allows individual objects to carry out autonomous tasks, where client tasks can request specialist services from server tasks.

The DOI is built on, and intimately tied to, an object-oriented foundation. It is a more powerful and intuitive way to interact with a computer. Instead of emphasising the individual software programs needed to do the job, the DOI concentrates on the task at hand and orchestrates transparent access to, and co-operation among, specialised programs. The DOI allows the user to think more about how to solve the problem, and spend less time on how to run the software.

DOI and client/server computing doesn't require new hardware, only a mechanism to allow these applications to be linked and manipulated. This linking mechanism will form the "glue" between the various object applications. This inter-application communication mechanism would allow the "Plug and Play" approach that the DOI and client/server applications require. This philosophy of representing applications as independent objects that communicate via an inter-application communication mechanism, is exactly what has been advocated throughout this thesis. This was the approach adopted within the implementation of the ORBERON environment, where symbolic and adaptive techniques could be represented as independent objects and could communicate with each other to form a client/server hybrid application. Also any tool that conforms to the communication message protocol defined in the Generic Interface Object of the system, could be "plugged and played" in the hybrid environment.

The mechanisms used to pass messages between objects are extremely important and have been the focus of attention for many organisations wishing to create a standard protocol for object communication and manipulation. Obviously, it is crucial to the success of object-oriented computing that there is agreement between manufacturers and software developers over the way objects communicate in distributed systems.

At present many vendors are offering a variety of inter-application communication methods that offer some, if not all the operations that a DOI and client/server application would require. The best known and possibly best developed, is Object Linking and Embedding (OLE) from Microsoft [18]. OLE allows various forms of information manipulation and visualisation through objects. For example, spreadsheet data, text and graphics can be used and displayed within a single document, regardless of their data formats. Rather than having to manually switch between different applications and import data or pictures into the document, with the DOI and OLE the user can automatically link or embed the required objects, directly into the document.

Object linking and embedding (OLE) is a major feature of Microsoft's Windows 3.1 environment, that lets you achieve real applications integration. OLE is an enhanced method for combining information from a number of different applications into a single document called a "compound document". This DOI allows the construction of applications by "mixing and matching" small specialised tools. OLE coupled with its DOI orchestrates transparent access to, and co-operation among, these specialised tools. OLE uses objects within compound documents, to store and manipulate exported data, independent of format, from other tools and applications. Figure 8.1 shows an example of a compound document, that enables you to view and manipulate various forms of information (for example, spreadsheets, text, and graphics).

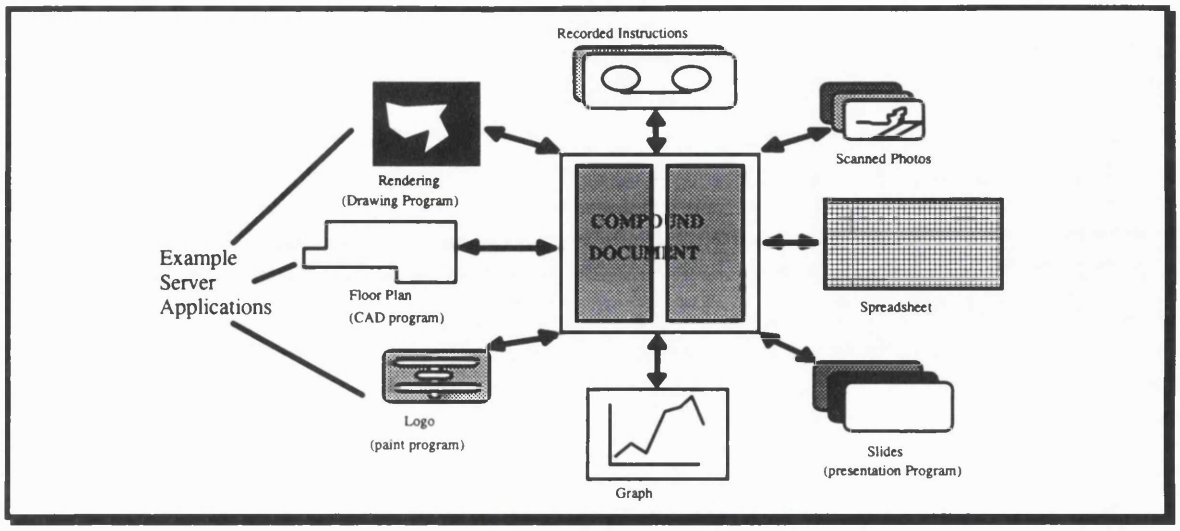


Figure 8.1 — A Compound Document.

OLE works via a system of servers (e.g. Paintbrush in Windows 3.1) and clients (e.g. Microsoft Word), that manipulate OLE objects. An object is any data that can be displayed in a compound document and manipulated by a user. An object can be anything from a single cell in a spreadsheet to an entire document or illustration. A server is an application that provides editing and rendering services for a linked or embedded object. When an object is activate (double-clicked) in a client document, the server application is invoked. A client is an application that receives, stores, and displays objects created by servers. If you use Word to create a report and then import your spreadsheet chart from Microsoft Excel, then Word would be the client and Excel would be the server in the exchange.

The ORBERON environment's Dynamic Environment Manager has similar functionality as a mechanism called the Registration Database within Microsoft's Windows environment. All applications must register with the registration database before they can function and exchange information with other applications within the Windows environment. The registration database is the mechanism that OLE programs use to get information about other OLE programs. Specifically OLE server applications have entries in the database that specify the executable filenames and paths, and addition information required to run the tools. The Dynamic Environment Manager is similar because it also runs in background of the ORBERON environment, stores all information about available and active processing tool, knows where every tool is stored and how to start that processing tool. The OLE mechanism also uses a simple transport mechanism and a standard message protocol. The transport mechanism that OLE is built on top of is called Dynamic Data Exchange (DDE) [16] (this is a simple message passing protocol for

client/server tools based on conversations about predefined topics), which is used to carry the complex OLE message protocol.

Other established systems vendors such as Apple, DEC, Sun and Hewlett Packard are also entering the field with their own interlinking mechanisms, which can interlink documents and applications from different machines [114]. DEC is attempting to address this issue with its ACA (Applications Control Architecture), CDA (Compound Document Architecture), and DDIF (Digital Document Interchange format) [76]. The only conceptual difference between ACA and OLE is that ACA will run on multiple architectures. Apple also has two technologies that compete with OLE: Apple Events and AppleScript. Together they also give applications a protocol for exchanging commands and data.

Workstation manufacturers are also providing products that will link not only programs running on their systems but also programs running on the systems of other vendors. Sun Microsystems in their new Solaris 2.0 operating system offer ToolTalk [104], a message service that allows independent applications to communicate. Applications create and send messages to the ToolTalk service. This ToolTalk service is responsible for maintaining a register of applications, and messages that each application is interested in, and for routing messages to the appropriate applications. The ToolTalk service is part of the Project DOE (Distributed Objects Everywhere), a joint development between Sun Microsystems and Hewlett-Packard. This project aims at providing a network-transparent object-messaging protocol for inter-application communication, that is similar to the Object Management Architecture from the Object Management Group.

In the Object Management Architecture, the ORB [38] enables client applications to access services and other objects that exist anywhere in the distributed system. Figure 8.2 shows how the ORB component within the OMG's evolving Object Management Architecture (OMA) will enable objects to transparently make, receive and respond to requests within an object-oriented environment. As the heart of the standard, the ORB will provide the "communication highway", enabling objects to interact over a network of different systems.

The ORB is the central message handling mechanism allowing "objects" to communicate independent of operating systems, hardware architecture's or network protocols. The main task of the ORB is to register the location of objects, receive, accept, and route requests for services. In addition the ORB ensures that the request is processed properly. When a request for a service is made, an ORB locates the object, starts the specified operation (method), and passes it the parameters found in the request message. Since objects can exist anywhere on a network, you can locate them via a name service or

a unique identifier. The ORB again behaves in a similar fashion to the ORBERON Dynamic Environment Manager, in that it knows where objects reside, receives requests, starts and routes messages to those objects.

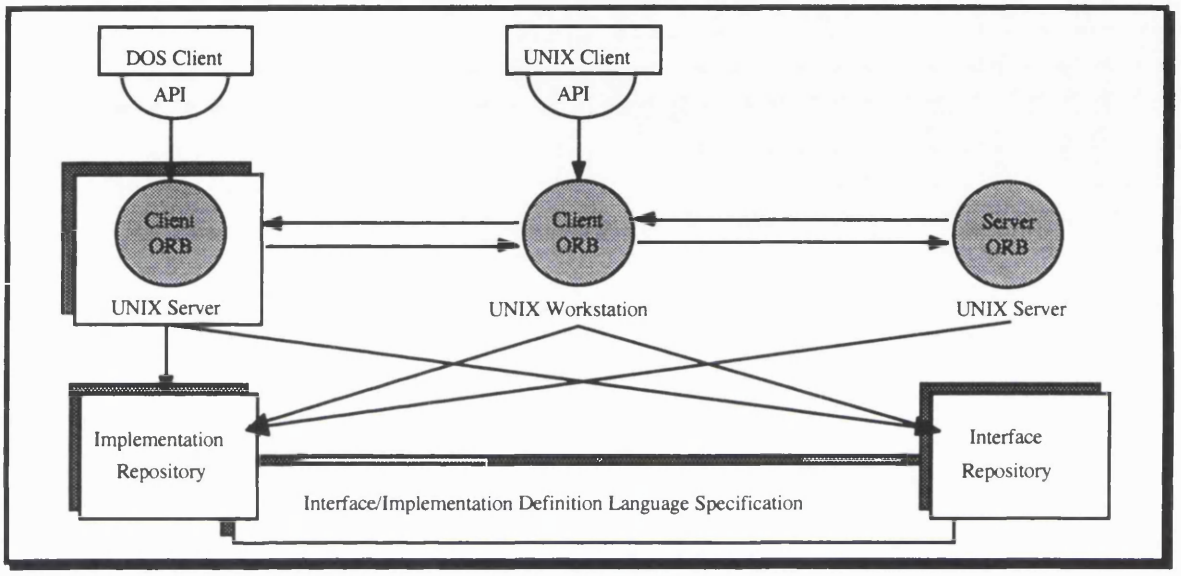


Figure 8.2 — The Object Request Broker in a distributed system.

In all of these inter-application communication mechanisms they use a transport mechanism and have a message protocol for exchanging commands and data. This was also the strategy adopted in the ORBERON environment, where object-oriented message passing was used as the transport mechanism and a message protocol composed of the interface functions to the symbolic and adaptive processing techniques found within the ORBERON environment. The message protocol is inherited by new processing objects by a process of "wrapping" the new processing technique in the Generic Interface Object, within which the message protocol was defined. As in Sun Microsystems ToolTalk service, the ORBERON Dynamic Environment Manager maintains a list of active processing objects, and routes messages to the appropriate processing tools. The advantage that ORBERON has over these commercial inter-application communication mechanisms, is it's simplicity and easy of usage for building hybrid systems. The commercial mechanisms have undoubtedly more functionality but at the cost of being complex to use.

In addition to commercial systems, various researchers [107, 100, 18] have designed systems that have components similar to those detailed in this thesis. For example, to provide for the integration of adaptive computing into more general symbolic systems, James Hendler has developed an intermediary mechanism that encapsulates a adaptive computing technique [123]. Hendler's method for integrating symbolic/adaptive techniques involved an intermediary called a *supervisor*, or a *super* [123]. A supervisor is a

software agent which provides an interface to and monitors the performance of a connectionist model. Supervisors and their respective networks can be created using a system called *Conncert*. Conncert consists of a set of objects for creating networks and their supervisors, written in the object-oriented language C++. Within the Conncert system, the supervisor provides the co-ordination of objects and a symbolic interface to the networks through which high-level information can be passed. Hendler's aim is similar to the design ideas behind the ORBERON environment, in that, a fully encapsulated adaptive module having a high-level symbolic interface, would be usable like any other software module. This is the role played of the Generic Interface Object and the Dynamic Environment Manager in the ORBERON environment. The supervisor is like a mix of both the Generic Interface Object and the Dynamic Environment Manager from the ORBERON environment. This is because it controls the environment in which the network object functions and presents the network's answers to the symbolic system, as does the Dynamic Environment Manager for the ORBERON environment. The supervisor also encapsulates the adaptive network, providing an interface to that network (see Figure 8.3), which is the function of the Generic Interface Object in the ORBERON environment. Using this structure maintains a distinct boundary between the network and the rest of the system and also the low level details of network operation are hidden.

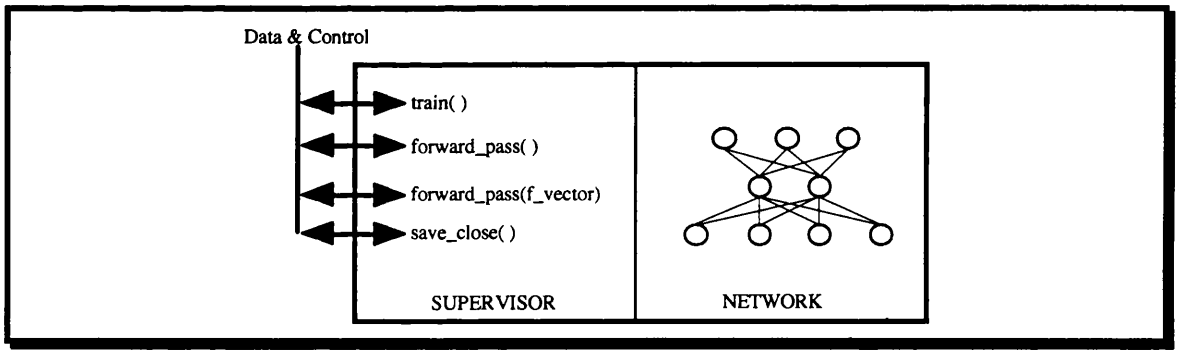


Figure 8.3 — Back Propagation Network Object utilising the Supervisor object.

In Hendler's system interactions between network objects occur via message passing as in the ORBERON environment. Message passing between network objects consist of high level control requests which invoke functions defined in the interface. At the same time adaptive data flows through the objects using paths which are distinct from those used by control messages. In the ORBERON environment this independent data and control message passing is not necessary because data and control messages are passed in the same message. Although Hendler's systems is very similar in functionality, the supervisor is the main component for encapsulation and control where as the ORBERON environment, has two distinct components for these task. Also Hendler's system is

specifically created for interfacing neural networks with symbolic systems and is not designed to be general purpose environment for building hybrid systems.

The use of object orientation in integrating diverse processing components within hybrid systems is also characteristic of recent developments in application integration software, such as Hewlett-Packard's NewWave, Object Linking and Embedding (OLE) in Microsoft Windows, and in many planned Distributed Computing Environments (DCE).

The HANSA project detailed in Chapter 7 offers a framework that also uses object-oriented techniques for integration of standard tools. The HANSA Framework offers an object-oriented cross-platform framework that was inspired by the ORBERON environment and recent commercial developments such as Microsoft's OLE and OMG's CORBA integration methods. The original design concepts for the HANSA project where inspired by the hybrid ideas presented within this thesis. These included the need to develop an object-oriented environment to integrate adaptive techniques with conventional commercial symbolic techniques, such as databases and spreadsheets. The HANSA Framework consists of the HANSA Communication Architecture (HCA) and the HANSA Registration Database. The HCA is the infrastructure for configuration and data exchange within an application and the HANSA registration database is the holder of all the information concerning HANSA tools. The registration database acts as the repository of information about tools available to the user, in a similar manner as the Dynamic Environment Manager within the ORBERON environment. The HCA also contains the basic concepts found within the ORBERON environment, such as a standard message-passing protocol for communication between objects.

To ensure that the HANSA Framework is compatible with the best emerging object-oriented integration methods, the underlying HANSA communication protocols (HDE, HOH, and HAP) were designed to be compatible with Microsoft's DDE and OLE protocol, and OMG's ORB. As previously described in this chapter, these inter-application communication mechanisms are technically better than the ORBERON environment, but still contain components that match the functionality of components in the ORBERON environment. The HANSA Framework can be thought of as the commercial evolution of the ORBERON environment, in that it extends the hybrid environment via industry standard protocols and new object-oriented techniques.

With the addition of a distributed version of the HANSA communication architecture, using TCP/IP as the network transport layer, the HANSA Framework will offer a truly unique method to build distributed hybrid applications that have communicating components running on PCs and workstations.

This thesis has shown that the requirements of building hybrid systems and distributed computing environments are extremely similar in that they require a representation scheme and a well-defined communication protocol. The goals of using object technology in all of these systems is to provide the "glue" which allows applications to share information with one another in an easy and consistent manner.

8.4. British Airways Cargo Consignment Problem

The main aim of cargo consignment is to maximise net profit by accepting the ideal types of cargo requests, prior to a aircraft taking off. Therefore, the problem that most airlines have to solve is when to accept or reject a potential cargo request. The acceptance or rejection of cargo depends on many pieces of qualitative and quantitative information which can cause a real decision problem for the human sales agent who has to make the final decision. This was the main reason that British Airways had devised the ASP algorithm (Acceptable Shipment Package) [37] to assist the decision making process.

As described in Chapter 6, the ASP algorithm is a simple weight-scoring mechanism that implements a point scoring system for each cargo request, based on factors such as the consignment weight and volume, density of consignment, rate offered and the urgency of the consignment. However, the weighting given to these various factors has been hard to fine-tune, with the result that cargo yield analysts often disagree with the ASP recommendation when a request is scored in the range 90 to 110 (where 100 is the threshold for accepting a booking). They often decide to accept bookings which ASP would reject because the flight is not filling as quickly as the usual trend. They may also reject the ASP accept recommendations if the booking requirement does not correspond with what is available.

The hybrid solution to the cargo consignment problem, as described in Chapter 6, uses neural networks trained on historical cargo acceptances and rejections, combined with an improved version of the ASP algorithm, to make the final accept or reject decision. The hybrid solution was run against a test set of previously unseen cargo requests for Auckland, Melbourne and Sydney. This same test set was also run against an implementation of the British Airways ASP algorithm and against the improved ASP algorithm used within the hybrid solution. Table 8.1 shows the results for these three systems.

These results clearly show that the hybrid solution produces a better quality result than the existing purely expert system based approach (i.e. ASP) to solving the cargo consignment problem. The hybrid solution classified 21.34% more requests correctly when compared to the original British Airways ASP algorithm. As British Airways has a large

amount of its revenue dependent on the judgement of it's cargo yield analysts, a system that can aid the decision making process with this degree of accuracy (when compared to the existing ASP system) would greatly increase the confidence that a correct decision has been made. Even a small increase in performance would be extremely valuable to British Airways.

	Percentage Correct on Test Set
British Airways ASP	62.5%
Improved ASP	67.5%
Hybrid Solution	83.84%

Table 8.1 — Comparison of results for the original ASP algorithm, the improved implementation and the final hybrid solution.

Also another important improvement would be the saving in time of processing a request. Currently it takes approximately 1 hour (from the British Airways data) from when the request is received to when the final decision is made. This could be made almost real-time, where the analyst could say almost immediately if the request would be accepted or not. This would not only improve the service offered to the client, but also increase the volume of requests that can be processed by the analyst. This inturn would help to increase the revenue of the company.

The use of such a hybrid system for cargo consignment, would give the following advantages:

1. Adaptability in the face of a changing environment, where the neural networks can be retrained as the cargo trends for a route or flight change.
2. Automation of all but the key decisions.
3. Combination of processing techniques to help make better quality decisions.
4. The cargo consignment process becomes profit and service driven rather than capacity driven.

The British Airways cargo consignment application has shown the suitability of the ORBERON environment and of the hybrid approach for building real-world solutions.

8.5. Summary

The main goals of this research work have been to establish the theoretical integration issues of hybrid systems and to construct the ORBERON environment for applying hybrid systems. The main areas covered during this research have been the definition of a hybrid classification scheme, the design and implementation of an object-oriented hybrid environment (ORBERON) and the implementation of a real-world cargo consignment application from British Airways.

The hybrid classification scheme has been compared to a scheme devised by Medsker and Bailey [68] and proved to be simpler, did not contain redundant classes and therefore was easier to apply to various types of hybrid systems.

The object-oriented approach to hybrid systems and the ORBERON environment were compared with emerging integration technologies (such as Microsoft's OLE and the Object Management Group's Object Request Broker) that use object-oriented methods for communication and co-ordination of components. The functional aspects of components of the ORBERON environment are compared with the components found in these new integration techniques. The HANSA Framework and the ORBERON environment, are also compared with respect to functionality.

Finally, the results of the hybrid British Airways cargo consignment solution has been assessed with respect to the results produced by the currently used expert system implementation of the Acceptable Shipment Package algorithm. The results from the hybrid solution showed a marked improvement in the quality of decisions and performance of the cargo acceptance procedure.

Chapter 9

Conclusions and Future Research

This chapter summaries and presents the conclusions of this thesis. Future extensions and developments of the ORBERON environment, the HANSA Framework and the hybrid classification scheme are also discussed.

9.1. Summary

The main objective of this thesis has been to investigate and establish a theoretical and practical foundation for the integration of symbolic and adaptive information processing. The necessity for hybrid systems has been recognised because of the failure of adopting purely symbolic or adaptive solutions to solve complex real-world problems.

By examining the history of the symbolic-adaptive divide and by focusing on the most popular processing techniques from symbolic and adaptive processing, this thesis presents an analysis of these two fundamentally different processing techniques. This work concentrates on expert systems and neural networks as popular representatives from symbolic and adaptive processing. A detailed examination of expert systems and neural networks with respect to their structure, strengths and weaknesses, applications and programming environments reveals that these techniques are not actually competing but complementary ones.

A special emphasis has been placed in examining the theory and clarifying the terminology found in this emerging field of hybrid systems. At present there are many interpretations of what a hybrid system is, and there is much confusion surrounding the functionality and architecture of these systems. This research has helped to resolve this confusion by examining the arguments for hybrid systems and categorising hybrid systems into three separate classes. This hybrid classification takes into account factors such as functionality, processing architecture and communication requirements. This analysis and classification of hybrid systems has been adopted by various researchers in the field [63, 97] and forms the foundation of an edited book titled "Intelligent Hybrid Systems".

Several intercommunicating hybrid configurations for integrating neural networks and experts systems are also analysed with regards to their functionality. Recent developments in connectionist expert systems, which fuse the functionality of expert systems within the architecture of connectionist networks (i.e. a polymorphic hybrid), are reviewed. An attempt is also made at highlighting possible problems that developers may encounter as hybrid systems evolve. To aid future implementors of hybrid systems, a development cycle for hybrid systems is presented.

A number of options are discussed for developing hybrid systems, and the most pragmatic choice has been made in favour of integrating symbolic and adaptive systems by using an object-oriented methodology. An object-oriented approach has many modular characteristics such as classes, encapsulation, inheritance, polymorphism and a well defined communication mechanism, that give many possibilities for constructing hybrid systems.

The design and implementation details of the object-oriented hybrid environment called ORBERON, are detailed with respect to the initial integration of expert systems and neural networks. Because the ORBERON environment is designed using object-oriented techniques other processing techniques (e.g. genetic algorithms) can be integrated without additional effort. Adopting an object-oriented approach allowed processing techniques to be represented as objects irrespective of their internal representation. All processing techniques were "wrapped" within the Generic Interface Object, within which a standard communication protocol was defined. Therefore, the processing techniques inherit the standard communication messages that would allow them to communicate to other techniques active within the system. Communication was conducted via the Dynamic Environment Manager, which maintained and manipulated active techniques within the environment. Object-orientation also has advantages that it allows for multiple copies of the same technique to be active at the same time, which allow techniques to be chained together. The principle components found within the environment have functional equivalence within current commercial integration mechanisms. Therefore, this work also analyses and compares functional aspects of the ORBERON environment with current developments in commercial object-oriented integration methods, namely OLE (object linking and embedding) from Microsoft [18] and the ORB (object request broker) from the Object Management Group [38].

The basic ideas and concepts of the ORBERON environment have been the inspiration for the European Esprit Project called HANSA (Heterogeneous Application geNerator Standard Architecture). This project aims to develop a framework to allow the rapid development of applications, by the integration of industry standard tools (spreadsheets, graphics packages) and novel artificial intelligence techniques (neural

networks, expert systems and genetic algorithms). The HANSA Framework can be viewed as the commercial evolution of the ORBERON environment, in that it retains some of the functional aspects of the environment but extends the architecture by using industry standard protocols and new object-oriented techniques.

To test the operation of the ORBERON environment and to show that a hybrid solution to a real-world problem is often better than a purely symbolic or adaptive approach, the ORBERON environment has been applied to two problems. The first application was used as the proof of concept for the environments functionality and involved a Profit Trend Analysis Problem. This problem had been tackled by the domain expert, via a purely symbolic processing method (expert systems) and had performed poorly. The use of neural networks to perform pattern recognition on the movement trends, coupled with an expert system for decision support showed that the hybrid approach and the ORBERON environment performed efficiently.

A real-world problem was provided by British Airways and involved finding a better solution to current expert system based method for Cargo Consignment. The problem has many interrelated parts, but simplified involved the efficient consignment of cargo so as to maximise profit and space utilisation. Both the Profit Trend Analysis and the Cargo Consignment problems are analysed and their solutions described.

Finally the results of this hybrid solution to the Cargo Consignment problem are compared with the results from the current symbolic Acceptable Shipment Package (ASP) used at British Airways. Also comparisons are made with the performance of the hybrid approach and that of the British Airways Acceptable Shipment Package.

9.2. Conclusions

The main conclusions of this thesis work are now presented on a chapter by chapter basis.

In Chapter 3, a comprehensive analysis of the arguments for hybrid systems is carried out using the results of the survey performed on expert systems and neural networks, in Chapter 2. This analysis leads to the formulation of the hybrid classification scheme. This examination continues with the investigation of methods of integrating expert systems and neural networks. These investigations gave some insight into the problems that future hybrid systems may encounter. The conclusions that can be drawn from this analysis are:

- Both symbolic and adaptive processing have a narrow area of competence and outside this area they function poorly.

- Symbolic and adaptive systems have complementary strengths and weaknesses especially in the areas of knowledge acquisition, brittleness, high-level and low-level reasoning, and explanation. By integrating these two fundamentally different models of computation one can avoid many of the weaknesses inherent in each methodology, whilst capitalising on their individual strengths.

- Most real-world problems are too complex for any single processing technique to solve in isolation. Most real-world domains have both "logical", static components (that can be easily handled by symbolic techniques) and fuzzy, dynamic, poorly understood components (which can be handled by adaptive processing techniques).

- Hybrid systems can be classified with respect to factors such as functionality, processing architecture and communication requirements to form three classes : Function-replacing, Intercommunicating and Polymorphic hybrid systems.

- There are many ways that neural networks and experts systems can be combined to utilise their strengths within a hybrid solution.

- Problems may occur in the future with hybrid systems in relation to efficient communication, learning and explanation. Although these problems may exist, they can be reduced by following the proposed development cycle for hybrid systems.

In Chapter 4, an object-oriented methodology is advocated as the most pragmatic method of constructing hybrid systems. Various properties and advantages of object-oriented integration for building hybrid systems are discussed. It can be concluded that :

- Intercommunicating hybrids require *Design, Communication and Execution Autonomy.*
- With an object-oriented approach, techniques can be integrated with communication being conducted through a well-defined message interface and they can be autonomous because components may independently and transparently change internally provided their interfaces are maintained.
- Object-oriented message passing is a well defined communications protocol that allows links to be established at runtime.
- Employing object-oriented techniques allows the ability to manage concurrency because message passing systems are adaptable to many types of hardware environments, including parallel multi-processor and distributed machines.

- Code can be reused, this allows for multiple copies of the same processing technique. Therefore, chained processing can occur where a combination of techniques can pass data between themselves.

In Chapter 5, the implementation details of the ORBERON object-oriented hybrid environment are discussed in depth. The design and implementation of the fundamental elements of the environment are explained. The neural network and expert systems objects are also described with relation to how they are integrated into the ORBERON environment. It can be concluded that :

- that object-oriented methodology is ideally suited for constructing hybrid systems especially intercommunicating hybrid systems. The utilisation of object-oriented benefits such as inheritance, polymorphism and code reuse are fundamental in the design, construction and functioning of the core elements of the environment.

In Chapter 6, two real-world problems that were solved using the ORBERON environment are analysed in detail. The first was a proof of concept application for profit trend analysis, that could advise portfolio managers whether to increase, maintain or decrease investment in a particular company. The second application was a complex real-world application from British Airways for the consignment of cargo. This basically involved the processing of cargo consignments for a particular flight and making a decision whether to accept or reject the cargo consignment. The goals being to optimise space and profit. The conclusions reached from the solution of these problems are :

- that real-world problems are too complex for either symbolic or adaptive techniques to solve in isolation, but require the strengths of both symbolic and adaptive processing.

- that a hybrid approach produced better results for British Airways, than a solution constructed from a single processing technique, namely an expert system.

- the ORBERON environment performed efficiently and allowed the design, communication and execution autonomy required by hybrid applications.

- the use of object-oriented programming allowed the flexibility and extendibility required by the ORBERON environment.

In Chapter 7, the design and implementation details of the Esprit III HANSA project are described. This project was inspired by the work presented in this thesis and it can be conclude that:

- the theories and functional concepts presented in this thesis are applicable to the production of a large scale commercial environment for producing hybrid applications.

- the object-oriented components found in the ORBERON environment are simplistic in comparison to emerging object-oriented integration methods (such as OLE and CORBA), but their functionality matches components within these elaborate integration methods.

In Chapter 8, an assessment is carried of the hybrid classification scheme, the ORBERON environment and the British Airways cargo consignment problem. The results from the cargo consignment hybrid solution are compared with the results of the purely symbolic British Airways ASP (Acceptable Shipment Package) system. The conclusions that were reached from this analysis are :

- the hybrid classification scheme provides a clear definition of hybrid systems.

- the hybrid classification scheme provides a means of comparing different hybrid systems.

- due to the complex nature of the Cargo Consignment problem, there were sections of the problem that required the use of neural networks to perform pattern classification and expert systems to co-ordinate the neural networks and make high-level logical decisions based on the results of the neural networks.

- that the hybrid solutions can improve the quality and accuracy of the decisions in comparison to the decisions from the current symbolic Acceptable Shipment Package used by British Airways.

- that the hybrid solution can help to improve the quality, reliability and time taken to make the final decisions by the human Cargo Consignment officer.

- the basic design concepts and principles that characterise the functionality of the core elements of the environment, have functional similarities to components found in current commercial object-oriented integration methods such as the OMG's Object Request Broker, and Microsoft's Object Linking and Embedding.

9.3. Future Research

There are many possibilities currently being explored for extending this research work, most have been incorporated into the Esprit III HANSA project, others in the form of extensions to the original concepts and to the ORBERON environment.

The most immediate extensions to the ORBERON environment concern the addition of other symbolic and adaptive techniques to the hybrid environment. This would increase the set of techniques available for solving real-world problems. The current interest in genetic algorithms (adaptive techniques for search and optimisation) show that they offer properties that could complement neural networks and expert systems. An added benefit of adding a Genetic Algorithm would be that complex fitness functions (i.e. the function that is being optimised) can be written as an expert system rulebase or a neural network.

As the field of intelligent hybrid systems grows, there will be a need to place some of the concepts such as functionality, communication, control, and processing architecture on a clear and solid foundation. These factors will be explored and the Hybrid Classes extended to meet the requirements imposed by these factors. To match these extensions in the hybrid classes the ORBERON environment could be extended to allow the modelling of not just intercommunicating hybrids, but also different function-replacing and polymorphic hybrids. This would be possible due to the object-oriented structure of the environment. The functionality of the fundamental components of the ORBERON environment could be extended so that function-replacing and polymorphic hybrids could be built. For function-replacing hybrids the main processing techniques could be built as a collection of objects, where internally they represent their principle functions as component objects. This means that a principle function object such as the pattern matcher in an expert system, could be easily replaced by a another object that encodes a processing technique that is more efficient at pattern matching, such as a neural network. Whereas, for polymorphic hybrids such as neural networks that behave like expert systems (i.e. connectionist expert systems) object-oriented environment components could be used to build the low-level structures of the neural architecture. This would allow symbolic processing mechanisms such as variable binding or logical inferencing, to be built into the low-level neural representation by simply extending the functionality of the neuron objects.

Another possible extension for the ORBERON environment is the distribution of the environment across a network of different computers. This would allow power hungry techniques such as neural networks to run on fast machines and still interact with other techniques running on different machines across a network. This is possible because of ORBERON's object-oriented representation, where objects are self-contained processing techniques that could be executed on any machine and these objects naturally communicate via message-passing, which is ideal for network communication.

The Cargo Consignment hybrid solution is now being evaluate by British Airways Artificial Intelligence Unit, to extend the solution into a full size system for cargo consignment for all regions. British Airways are currently interested in the application of

similar hybrid techniques for their seat forecasting and allocation problem, which should be an easier problem than cargo consignment.

The HANSA Framework can be viewed as the commercial extension of the design concepts found in the ORBERON environment by the addition of industry standard protocols and new object-oriented techniques. With current plans to extend the HANSA Framework with the addition of a distributed version of the HANSA communication architecture, using TCP/IP as the network transport layer, the HANSA Framework will allow distributed hybrid applications to be built from components running on PCs and workstations.

With the expansion of world-wide computer networks such as the internet, a future extension to the distributed HANSA framework could be the use of Intelligent Agents [52] as the data and control message carriers for truly wide area network communication. Intelligent Agents can bundle messages, data and control information into an intelligent program that travels over the network to a distant computer, delivers the messages etc. and returns with an answer. This would enable the HANSA applications to utilise components that may reside across a world-wide network and allow the HANSA Framework to intelligently control and co-ordinate the construction of truly real-world intelligent hybrid systems.

References

[1] Ackley, David H., "A Connectionist Algorithm For Genetic Search", *Proceedings of First International Conference on Genetic Algorithms and heir applications*, 1988

[2] Aikins, J., Kunz, J., and Shortcliffe, E., "PUFF: an expert system for interpretation of pulmonary function data", *Computers and Biomedical Research*, vol. 16, pp. 199-208, 1983

[3] Ajjanagadde, Venkat and Shastri, Lokendra, "Rules And Variables In Neural Nets", *Neural Computation*, vol. 3, no. 1, pp. 121-134, 1991

[4] Aleksander, Prof Igor, "Are Special Chips Necessary For Neural Computing ?", Department of Computing Imperial College of Science and Technology, London, 1988

[5] Arnold, Ken, "Concocting More Powerful Curses", *C Advisor*, pp. 71-74, April 1986

[6] Atkins, M. and Deich, R., "An Hybrid Artificial Diagnostics System", The Intelligent Systems Group, General Dynamics Electronics Division, San Diego, California., 1992

[7] Barker, Don, "Neural Networks and Expert Systems", *AI Week*, vol. 7, no. 9, pp. 1-6, May 1, 1990

[8] Belobaba, P.P, "Air Travel Demand and Airline Seat Inventory Management", Massachusetts Institute of Technology, *Doctoral Dissertation*, 1987

[9] Bergerson, Karl and Wunsch, Donald C., "A Commodity Trading Model Based On a Neural Network-Expert System Hybrid ", *Proceedings of International Joint Conference of Neural Networks, Seatle*, pp. 289-293, July 1991

[10] Boley, Harold, "Expert System Shells: Very-high-level Languages For Artificial Intelligence", *Expert Systems*, vol. 7, no. 1, Feb 1990

[11] Brachman, R., "On the epistemological status of semantic networks", in *Associative Networks, Representation and Use of Knowledge by Computer*, ed. N. Findler, Academic Press, 1979

[12] Buchanan, B. and Feigenbaum, E,, "DENDRAL and Meta-DENDRAL: their applications dimension", *Artificial Intelligence*, vol. 11, 1978

[13] Cargex, "Cargex helps lighten cargo agents load", *Expert Systems User*, March 1989

[14] Cerny, V., "Thermodynamic approach to the travelling salesman problem: an efficient simulation algorithm", *Journal of Optimisation Theory & Applications*, vol. 45, pp. 260-266, 1985

[15] Clancy, W.J, "The Epistemology of a Rule-based Expert System: A Framework For Explanation", *Artificial Intelligence* , vol. 20, pp. 215-251, 1983

[16] Clark, J. D., "Windows Programmers Guide to OLE/DDE", SAMS/Prentice Hall Computer Publishing, 1992

[17] Corkill, Daniel, "Blackboard Systems", *AI Expert*, pp. 41-47, Sept 1991

[18] Corporation, Microsoft, "Object Linking and Embedding", 10 June 1991

[19] Davis, R., Austin, H., Carlbom, I., Fawley, B., Prucknik, P., Sneiderman, R., and Gilreath, J., "The Dipmeter Advisor: interpretation of geologic signals", *International Joint Conference on Artificial Intelligence*, pp. 846-849, 1981

[20] Davis, R., Buchanan, B., and Shortcliffe, E., "Production Rules as a Representation For a Knowledge-Based Consultation Program", *Artificial Intelligence*, vol. 8, no. 1, pp. 15-45, 1977

[21] Diederich, Joachim, "Explanation and Artificial Neural Networks", GERMAN National Research Center for Computer Science(GMD), W.Germany, 1989

[22] Dietrich, W. C, Nackman, L. R, and Gracer, F, "Saving a legacy with objects", ed. N Meyrowtz, Addison-Wesley, Reading, MA, *Proceeedings of OOPSLA89*, 1989

[23] Dunker, Jurgen, Scherer, Andreas, and Schlageter, Gunter, "Integrating Neural Networks Into Distributed Knowledge Based Systems", *Proc of 12th conf on AI, Expert Systems and Natural language, Avignon, France*, 1992

[24] Dutta, Soumitra and Shekhar, Shashi, "Bond Rating : A Non-Conservative Application Of Neural Networks", *Proceedings of Second IEEE International conf on Neural Networks, San Diego*, vol. TWO, pp. 443-450, 1988

[25] Feigenbaum, E., "Themes and Case Studies of Knowledge Engineering", in *Expert Systems in the micro-technology age*, ed. D. Mitchie, Edinburgh University Press, 1979

[26] Feigenbaum, E., McCorduck, P., and Nii, H., *The Rise of the Expert Company*, Random House Publishing, New York, NY, 1988

[27] Feldman, Jerome A. and Ballard, Dona H., "Connectionist Models And Their Properties", *Cognitive Science*, p. 205, 1983

[28] Fodor, Jerry A. and Pylyshyn, Zenon W., "Connectionism and Cognitive Architecture : A Critical Analysis ", *Cognition*, vol. 28, pp. 3-71, 1988

[29] Fodor, Jerry A., *The Language of Thought*, Harvard University Press, Boston, MA, 1976

[30] Forgy, C., "OPS5 User's Manual", CMU-CS-81-135, Computer Science Dept, Carnegie-Mellon University, Pittsburgh, July 1981

[31] Gallant, Stephen I., "Connectionist Expert Systems", ACM, *Communications of the ACM*, vol. 31, no. 2, pp. 152-169, Feb 1988

[32] Gaschnig, J., "PROSPECTOR: An expert system for mineral exploration", *Machine Intelligence*, vol. 9, no. 3, 1981

[33] Gavarter, W., "The Nature and Evaluation of Commercial Expert System Building Tools", *IEEE Computer*, pp. 24-41, May 1987

[34] Giarratano, Joseph and Riley, Gary, "Chapter 7 - Introduction To Clips", in *Expert Systems, Principles and Programming*, PWS-Kent, 1989

[35] Goonatilake, Suran and Khebbal, Sukhdev, "Intelligent Hybrid Systems : Issues, Classes and Future Directions", in *Intelligent Hybrid Systems*, ed. S. Goonatilake and S. Khebbal, John Wiley, Jan 1995

[36] Goonatilake, Suran and Khebbal, Sukhdev, "Intelligent Hybrid Systems", *Proceedings of First International Conf on Intelligent Systems, Singapore*, pp. 207-212, Sept 1992

[37] Greenwood, Rosemary, "Acceptable Shipment Package (ASP) - A Functional Specification", British Airways, 3 Jan 1991

[38] Group, Object Management, "The Common Object Request Broker: Architecture and Specification", Object Management Group, 1992

[39] Gryphon, Coranth and Miller, Mark, "PCLIPS: Parallel CLIPS", Center of Productivity Enhancement, University of Massachusetts at Lowell, 1992

[40] Gutierrez, M., , J. Wang, and Grondin, R.O., "Estimating Hidden Units For Two-Layer Perceptrons ", *First IEE International Conference on Artificial Neural Networks*, pp. 120-124, October 1989

[41] Hall, Lawerance O. and Romaniuk, Steve G., "FUZZNET: towards a fuzzy connectionist expert system development tool", *Proceedings of 8th National Conf on Artificial Intelligence*, vol. Two, pp. 783-788, July 1990

[42] Handelman, David A., Lane, Stephen H., and Gelfand, Jack J., "Integrating Knowledge-Based System and Neural Network Techniques for Robotics Skill Acquisition", Morgan Kaufmann, *Proceedings of IJCAI, Detroit*, pp. 193-198, August 1989

[43] Harmon, P. and Morrissey, W., *Expert Systems, Tools, and Applications*, John Wiley, 1988

[44] Hayes-Roth, F., Waterman, D., and Lenat, D., "An Overview of Expert Systems", in *Building Expert Systems*, ed. F. Hayes-Roth, D. Waterman, and D. Lenat, Addison-Wesley, 1983

[45] Hendler, James A., "Editorial for Hybrid Systems Special Issue : On The Need For Hybrid Systems", Carfax publishing Ltd, Journals Oxford Ltd, *Connection Science*, vol. One, no. 3, pp. 227-229, 1989

[46] Hinton, Geoffrey E., "Learning In Parallel Networks", *BYTE*, pp. 265-273, APRIL 1985

[47] Holland, J., "Escaping Brittleness", in *Machine Learning*, ed. R. Michalski, J. Carbonell, and T. Mitchell, Morgan Kaufmann, vol. 2, 1986

[48] Honavar, V. and Uhr, L., "Integrating Symbol Processing Systems and Connectionist Networks", in *Intelligent Hybrid Systems*, ed. S. Goonatilake and S. Khebbal, John Wiley, Jan 1995

[49] Hopfield, John J., "Neural Networks And Physical Systems With Emergent Collective Computational Abilities", *Biophysics : Proceedings of National Academy of Sciences, USA*, vol. 79, pp. 2554-2558, April 1982

[50] Hutchinson, W.R and Stephens, K.R, "The airline marketing tactician (AMT): a commercial application of adaptive networking", IEEE Press, Piscataway, NJ, *Proc. of the 1st Int. Conf.on Neural Networks*, vol. 4, pp. 753-756, 1987

[51] Inc, New Science Associates, "Neural Networks : Prospects For Commercial Use", *Industry Report Spring 1988*, Spring 1988

[52] *Intelligent Agents: Theories, architectures and Languages*, ed. M. Wooldridge and N. Jennings, Springer-Verlag, 1995

[53] Johnston, M.D. and Adorf, H.M., "Spike : AI Scheduling for NASA's Hubble Space Telescope", *Proceedings of 6th Conf on Artificial Intelligence Applications (CAIA90)*, pp. 184-190, Santa Barbara, California, March 1990

[54] Jones, William P. and Hoskins, Josiah, "Back-Propagation : A Generalised Delta Learning Rule", *BYTE*, pp. 155-162, OCTOBER 1987

[55] Josin, Gary, "Neural Network Heuristics", *BYTE*, pp. 183-192, OCTOBER 1987

[56] Kehler, T. and Clemenson, G., "An application development system for expert systems", *Systems and Software*, vol. 3, no. 1, January 1984

[57] Khebbal, S. and Shamhong, D., "Tools and Environments for Building Hybrid Systems", in *Intelligent Hybrid Systems*, ed. S. Goonatilake and S. Khebbal, John Wiley, Jan 1995

[58] Kingdon, J., Filho, J. Ribeiro, and Treleaven, P., "The GAME Programming Environment Architecture", TR/92/28, Department of Computer Science, University College London, 1992

[59] Kohonen, Teuvo, *Self-Organisation and associative Memory*, Springer-Verlag, 1984

[60] Lee, Matthew K.O., "Neural Networks and Knowledge Based Systems", University College London , October 1988

[61] Linton, Mark A., Vlissides, John M., and Calder, Paul R., "Composing User Interfaces with InterViews", *IEEE Computer*, vol. 22, no. 2, pp. 8-22, February 1989

[62] Lu, Cary, "Objects for End Users", *BYTE*, pp. 143-152, December 1992

[63] Madey, Gregory, Weinroth, Jay, and Shah, Vijay, "Hybrid Intelligent Systems: Tools for Decision Making in Intelligent Manufacturing", in *Neural Networks for Intelligent Manufacturing*, Chapman Hall, 1993

[64] Masters, Timothy, *Practical Neural Network Recipes in C++*, Academic Press, 1993

[65] McDermott, D., "Extracting Knowledge From Expert Systems", *Proceeding of 8th International Joint Conference on Artificial Intelligence*, pp. 100-107, 1983

[66] McDermott, J., "R1: an expert in the computer systems domain", *AI Magazine*, vol. 2, no. 2, 1981

[67] McMillan, Clayton, Mozer, Michael C., and Smolensky, Paul, "The Connectionist Scientist Game: Rule Extraction and Refinement in A Neural Network", *Proc of the Thirteenth Annual Conf of Cognitive Science Society, Hillside, NJ*, 1991

[68] Medsker, L. R. and Bailey, D. L., "Models and Guidelines For Integrating Expert Systems and Neural Networks", in *Hybrid Architectures for Intelligent Systems*, ed. A. Kandel and G. Langholz, CRC Press, pp. 154-171, 1992

[69] Melle, W. Van, Shortcliffe, E. H., and Buchanan, B. G., "EMYCIN: A domain-independent production-rule system for consultation programs", *International Joint Conference on Artificial Intelligence*, pp. 923-925, 1979

[70] Melle, W. Van, Shortcliffe, E. H., Buchanan, B. G., Buchanan, B. G., and Shortcliffe, E. H., "EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems", in *Rule-based Expert Systems*, Addison-Wesley, pp. 302-328, 1984

[71] Michalski, R. and Chilansky, R. L., "Learning by Being Told and Learning From Examples", *Journal of Policy Analysis & Information Systems*, vol. 4, 1980

[72] Mimetics, "Mimenice Reference Manual", Paris, France, 1994

[73] Minsky, M. and Papert, S., *Perceptrons (expanded edition)*, MIT Press, Cambridge, MA, 1988

[74] MITI and Industry, Japanese Ministry of International Trade and, "The Master Plan for the Real-World Computing Program: Feasibility Study Committee of the Real-World Computing Program", May 1992

[75] Montana, D. and Davis, L., "Training Feedforward Neural Networks using Genetic Algorithms", *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pp. 762-767, 1989

[76] Mowbray, T. J. and Brando, T., "Intereroperability and CORBA-based Open Systems", *Object Magazine*, October 1993

[77] Neumann, John Von, "The Computer And The Brain", 1958

[78] Newall, Allen and Herbert, Simon, "Computer Science as Emperical Inquiry: Symbols and Search", *Communications of the ACM*, vol. 19, no. 3, 1976

[79] Obermeier, Klaus K. and Barron, Janet J., "Time To Get Fired Up", *BYTE*, pp. 217-224, August 1989

[80] Organisation, International Standards, "Reference Model of Open Systems Interconnection", ISO/TC97/SC15 N227, Aug 1979

[81] Pacheco, Marco and Treleaven, Philip, "A VLSI Word-Slice Architecture For Neurocomputing ", University College London , April 1989

[82] Palmer, Richard P., "A Proposal For a CMOS Multi-Layer Perceptron Implementation For Use In Speech Processing Applications", University College London, 1989

[83] Partridge, Derek, "Is Intuitive Expertise Rule Based?", University of Exeter, UK, 1986

[84] Partridge, Derek, "What's Wrong With Neural Architectures", University of Exeter, UK, 1987b

[85] Pulfer, Rolf, "ARGOS, guardian of our cargo", *Swissair News*, April 1990

[86] Quinlan, J. R., "Induction of Decision Trees", *Machine Learning*, vol. 1, pp. 81-106, 1986

[87] Rasmus, Daniel W, "Relating to Objects", *BYTE*, pp. 161-165, December 1991

[88] Rose, Daniel, "Appropriate Uses of Hybrid Systems", ed. David S. Touretzky, *Connectionist Models, Proceedings of the 1990 Summer School*, pp. 277-286, 1990

[89] Rosenblatt, F., *Principles of Neurodynamics, Perceptrons and the Theory of Brain Mechanisms*, Spartan Books, Portage, MI, 1962

[90] Rumelhart, David E. and Hinton, Geoffrey E., *Parallel Distributed Processing - Explorations in the Microstructure of Cognition : A Handbook of Models, Programs and Exercise.*, MIT Press, Cambridge, MASS, vol. THREE, 1986

[91] Rumelhart, David E., Hinton, Geoffrey E., and McClelland, James L., "The Appeal of Parallel Distributed Processing - Chapter 1", in *Parallel Distributed Processing - Explorations in the Microstructure of Cognition*, ed. David E. Rumelhart and James L. McClelland, MIT Press, Cambridge, MASS, vol. ONE, 1986

[92] Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J., "Learning Internal Representations By Error Propagation - Chapter 8", in *Parallel Distributed Processing - Explorations in the Microstructure of Cognition*, ed. David E. Rumelhart and James L. McClelland, MIT Press, Cambridge, MASS, vol. ONE, 1986

[93] Rumelhart, David E., McClelland, James L., and Hinton, Geoffrey E., "A General Framework For Parallel Distributed Processing - Chapter 2", in *Parallel Distributed Processing - Explorations in the Microstructure of Cognition*, ed. David E. Rumelhart and James L. McClelland, MIT Press, Cambridge, MASS, vol. ONE, 1986

[94] Saito, Kazumi and Nakano, Ryohei, "Medical Diagnostic Expert System Based On PDP Model", IEEE, *Proceedings of Second IEEE International conf on Neural Networks, San Diego*, vol. ONE, pp. 255-262, 1988

[95] Samad, Tariq, "Towards Connectionist Rule-Based Systems", IEEE, *Proceedings of Second IEEE International conf on Neural Networks, San Diego*, vol. Volume 2, pp. 525-532, 1988

[96] Scherer, A. and Schlageter, G., "A Multi Agent Approach for the Integration of Neural Networks and Expert Systems.", in *Intelligent Hybrid Systems*, ed. S. Goonatilake and S. Khebbal, John Wiley, Jan 1995

[97] Scherer, Andreas and Schlageter, Gunter, "Embedding Neural Networks In a Cooperative Framework: Philosophy and Architecture", *Proceedings of AAAI Workshop on Integrating Neural and Symbolic Processes,*, July 1992

[98] Schreinemakers, Jos F. and Touretzky, David S., "Interfacing a Neural Network with a Rule-Based Reasoner For Diagnosing Mastitis", *Proceedings of*

International Joint Conference of Neural Networks, Washington, pp. 487-490, July 1990

[99] Selviah, David, "Introduction To The UCL Matched Filter Model Of Neural Networks", University College London , 1988

[100] Sherald, Marge, "Neural Networks Versus Expert Systems: Is There Room For Both", *PC AI*, pp. 10-15, July/August 1989

[101] Shortcliffe, E., *Computer-based Medical Consultations: MYCIN*, Elsevier, 1976

[102] Southwick, Richard, "Explaining Reasoning: An Overview of Explanation in Knowledge-Based Systems", *The Knowledge Engineering Review*, vol. 6, no. 1, pp. 1-19, 1991

[103] Suddarth, Steven C. and Holden, Prof A., "A Symbolic Neural Method For Solving Control Problems", *Proceedings of Second IEEE International conf on Neural Networks, San Diego*, vol. ONE, pp. 516-523, 1988

[104] SunSoft, "Project DOE: Distributed Objects Everywhere", 2250 Garcia Avenue, Mountain View, CA, USA, December 1991

[105] *The Handbook of Artificial Intelligence*, ed. A. Barr and E. Feigenbaum, William Kaufmann, vol. 1, 1981

[106] Thomas, Dave, "What's In An Object", *BYTE*, pp. 231-240, March 1989

[107] Tirri, H., "Applying Neural Computing To Expert System Design: Coping With Complex Sensory Data And Attribute Selection", *Foundations of Data Organization and Algorithms, 3rd Int Conf, Paris* , pp. 474-488, June 1989

[108] Tirri, H., "Implementing Expert System Rule Conditions by Neural Networks", *New Generation Computing*, vol. 10, pp. 55-71, 1991

[109] Tirri, Henri, "Combining Connectionist Networks and Case-based Reasoning", in *Intelligent Hybrid Systems*, ed. S. Goonatilake and S. Khebbal, John Wiley, Jan 1995

[110] Touretzky, David S. and Hinton, Geoffrey E., "A Distributed Connectionist Production System", CMU-CS-86-172, 1986

[111] Touretzky, David S. and Pomerleau, Dean A., "What's Hidden In The Hidden Layers ?", *BYTE* , pp. 227-233, August 1989

[112] Treleaven, Philip, "Neurocomputers", University College London , 1989

[113] Treleaven, Philip, Recce, Michael, and Wang, Chang, "Neural Network Programming Environments: A Review", *Proceedings of the 4th European Seminars in Neurocomputing*, London Marriott Hotel, February 1991

[114] Ullman, Ellen, "Let The System Do The Porting: New Cross-Platform Operating Systems are Coming in the 1990s", *BYTE*, January 1992

[115] Voevodsky, John, "PLATO/ARISTOTLE: A Neural Network Knowledge Processor", *Proceedings of First IEEE International conf on Neural Networks, San Diego*, vol. TWO, pp. 399-407, 1987

[116] Waterman, D. A., *A Guide to Expert Systems*, Addison-Wesley , 1985

[117] Weber, Jack, "A Window Into The Brain", *Personal Computer World*, pp. 130-134 201-207, MARCH 1988, Neural Network Simulator for Apple

[118] Weigend, A., Huberman, B., and Rumelhart, D., "Predicting the Future: A Connectionist Approach", *International Journal of Neural Systems*, vol. 1, no. 3, 1990

[119] Wermter, Stefan and Lehnert, Wendy G., "A Hybrid Symbolic/Connectionist Model For Noun Phrase Understanding", Carfax publishing Ltd, Journals Oxford Ltd, *Connection Science*, vol. One, no. 3, pp. 255-272, 1989

[120] White, Halbert, "Economic Prediction using Neural Networks: The Case of IBM Daily Stock Returns", *Proceedings of Second IEEE International conf on Neural Networks, San Diego*, vol. TWO, pp. 451-458, 1988

[121] Wiener, Richard S. and Pinson, Lewis J., "An Introduction to Object-Oriented Programming and C++", Addison-Wesley Publishing Company, 1988

[122] Williams, C., "ART: the advanced reasoning tool.", Inference Corp Report, Inference Corp, 5300 W.Century Blvd, LA., 1984

[123] Wilson, A. and Hendler, J., "Linking Symbolic and Subsymbolic Computing", UMIACS-TR-93-9, February 1993

[124] Winston, Patrick and Horn, Berthold, *LISP*, Addison-Wesley, 1984

[125] Y.Otsuka, Hanaoka, K., and Konishi, M., "Applications of Neural Network Models to Iron and Steel Making Process", *Proceedings of the 2nd International Conference on Fuzzy Logic & Neural Networks*, pp. 815-818, Iizuka, Japan, July 1992

Appendix A

Successful Expert Systems and Their Application Domains

Category	System Name	Function
		CHEMISTRY
Interpretation	CRYSALIS	Infers the 3-D structure of a protein from an electrodensity map
	DENDRAL	Infers a compound's molecular structure from mass spectral an nucleur respose data
	TQMSTUNE	Fine tunes a triple quadraple mass spectrometer(TQMS) by intepreting TQMS signal data
Design/Planning	CLONER	Helps the molecular biologist design and create new molecules
	MOLGEN	Helps the molecular geneticist plan genecloning experiments
	SECS	Helps chemists synthesize complex organic molecules
	SPEX	Helps scientists plan complex laboratory experiments in molecular biology
	SYNCHEM2	Snthesizes complex organic molecules without human assistance.
		COMPUTER SYSTEMS
Prediction	PTRANS	Manages manufacture and distribution of DEC computer systems
Diagnosis	BDS	Locates faulty modules in a large signal-switching network
	DART	Helps diagnoses faults in computer hardware systems
	IDT	Helps locates defective units in PDP 11/03 computers
Designs	XCON	Configures VAX-11/780 computers
Monitoring	YES/MVS	Helps Monitor and control the MVS operating system
Debugging	TIMM/Tuner	Helps Tune VAX/WMS computers
		GEOLOGY
Interpretation	ELAS	Helps geologists use INLAN, a complex well log analysis program
	DIPMETER	Helps geologists interpret dipmeter logs
	LITHO	Helps geologists perform oil well log analysis
Diagnosis	DRILLING	Helps diagnose and correct oil rig "drill sticking" problems
	HYDRO	Helps hyrologist use HSPF, a water availability simulation program
	MUD	Helps diagnose and treat problems related to fluids used in drilling.
	PROSPECTOR	Helps geologists evaluate the mineral potential of a region
		MEDICINE
Interpretation	PUFF	Diagnoses lung disease by interpreting data from pulmonary tests
	SPE	Diagnoses inflammatory conditions by interpreting *scanning densitometer* data
	VM	Monitors Intensive-Care Unit patients by interpreting ICU data
Diagnosis	ABEL	Helps diagnose acid-base and electrolyte disorders.
	AI/COAG	Helps diagnose diseases of hemostasis.
	AI/REUM	Helps diagnose connective tissue diseases of clinical rheumatology
	CADUCEOUS	Helps diagnose diseases in general internal medicine
Monitoring	ANNA	Helps administer digitalis to patients with heart problems.
Diagnosis/debugging	BLUE BOX	Helps Diagnose and treat various forms of clinical despression
	GLAUCOMA	Helps diagnose and treat glaucoma-related diseases
	MYCIN	Helps diagnose and treat bacterial infections.
	ONCOCIN	Helps treat and manage cancer patients undergoing chemortherapy
Instruction	ATTENDING	Teaches methods of anesthetic management
	GUIDON	Teaches diagnosis & treatment of patients with bacterial infections

Appendix B

Neural Network Programming Environments

Class	Category	Organisation	System
Application-Oriented	Financial	Nestor	**Decision Learning System.** Adapts to changing risk environment, learns complex patterns from interpendent variables. Applications include mortgage underwriting, insurance & credit card sanctions.
	Transportation	BehavHeuristics	**Airline Marketing Tactician.** Based on adaptive network architecture. Accesses the data on advance booking and limits the number of seats that can be sold.
	Recognition	Excalibur	**Savvy** supports text retrieval, signal and vision recognition. For VAX/VMS and MS-DOS and Unix systems. Complete library of C routines.
	Diagnostics	Abtech	**Aim Problem Solver,** combines neural learning and statistical regression. Automatically determines network structure & node types. Allows Code Generation of networks to ANSI C.
Algorithm-Oriented	Algorithm-Specific	Cal. Scient. Softw.	**BrainMaker-**uses Back Error Propagation Consists of 3 packages : *NetMaker-*generates and manipulates networks, *BrainMaker-*Control environment & *Toolkit-*tools for manipulating input data. Uses arrays for network representation
	Algorithm Libraries	Olmsted & Watkins	**Owl,** Easily integrated, with extension packages : **OWL-I** - for IBM-AT & Macs has 19 popular neural models. **OWL-II** - Portable C code version for any machine
		Mimetics	**GALATEA C-library,** part of European Esprit GALATEA project. Has 18 classic algorithms parameterised. Users specify number of nodes, initial weights, learning rate and time constraints. Five parts : tools library, algorithm-independent section, algorithm modules, evaluation & applications library.
Programming Systems	Educational Systems	UCSD	**PDP,** Specific simulators for learning many PDP models. Source code in C & runs on IBM PC and UNIX machines. Contains 7 model simulators - Auto-Associator, Back Propagation, Competitive Learning, Constraint satisfaction, Interactive Activation, Interactive Activation & Competition & Pattern Association.
		NeuralWare	**Explorer,** User defined I/O specialised graphical displays. On-line help facility. Generates standard network types from its algorithm library.

	General-Purpose Systems	SAIC	**ANSim,** DBaseIII & Lotus 1-2-3 compatible. Has 13 neural models. Network programmed in object-oriented concurrent language. **ANSpec** allows different applications to include neural models. It is based on the actor model of concurrent distributed processing.
		HNC	**Anza/Axon,** for IBM PC and Sun 386i plus co-processor board. Consists of : *NetSet* - graphical usr interface; *AXON* - description language; *StubNet* - integration routines; *Library* - maths library routines; *Interact* - interactive debugger; *Parallelizer* - post-processor for ANZA co-processor boards.
		NeuralWare	**NeuralWorks Professional II,** for Macs, NCube, Sun and IBM PC's. Has 15 neural models. Generates C code application neural networks.
	Open Systems	Lucid	**Plexi,** Menu driven graphical interface, for Sun workstations. Has a Network and Pattern Editor.
		University College London	**PYGMALION,** part of European Esprit PYGMALION project. X-windows, C and C++ based. Consists of : *Graphic Monitor* - controls and monitors execution; *Algorithm Library* - common neural models *High Level Language N* - neural programming language; *Intermediate Language nC* - machine independent net specification language; *Compilers* - for Unix and Transputer machines.
	Hardware-Oriented Systems	Oregon Grad. Institute	**Anne,** for MIMD hyper-cube multi-computer & iPSC. Compiles Network Description Language and generates a low level Beaverton Intermediate Form (BIF). BIF is them mapped to processors.

Appendix C

Rulebase For Profit Trend Analysis Application

```
;;*****************************************************************************
;;**                                                                         **
;;** Profit Trend Analysis Knowledge Base                                    **
;;** Date Changed  : 03/12/91                                                **
;;** File Name     : PTA.CLP                                                 **
;;**                                                                         **
;;*****************************************************************************

;;*****************************************************************************
;;** Setup initial facts                                                     **
;;*****************************************************************************

(deffacts Defaults
        (Current Risk Level 1.0)
        (Number Of Days 1)
        (Average Weekly Share Index 0.0)
        (Total Profit 0.0)
        (TwoWeeks Profit 0.3 0.4 0.5 0.5 0.3 0.2 0.4 0.5 0.4 0.3)
        (Quarterly ShareIndex 0.4 0.6 0.7 0.6 0.5 0.3 0.2 0.3 0.5 0.6 0.5 0.4)
        (Yesterdays Profit Trend stable)
        (Sum Invested 246.00)
        (Investment 100)
        (Yesterdays ClosingPrice 2.46) )

;;*****************************************************************************
;;** Main Processing Rules                                                   **
;;*****************************************************************************

;;*****************************************************************************
;;** Get User Responses For Action Processing                                **
;;*****************************************************************************

(defrule Closing
        ?closing <-(Action Get Closing)
        =>
        (retract ?closing)
        (printout t t "Enter the Closing Price for the Invested Company : ")
        (bind ?CPrice (read))
        (while (not (numberp ?CPrice))
          (printout t "Enter the Closing Price Again (Must be a Number) : ")
        (bind ?CPrice (read)) )
        (assert (Todays ClosingPrice ?CPrice))
        (printout t ) )

(defrule Loss
        ?loss <-(Action Get Loss)
        =>
        (retract ?loss)
        (printout t t "Enter the Loss Limit for your investment : ")
        (bind ?LossLim (read))
        (while (not (numberp ?LossLim))
          (printout t "Enter the Loss Limit Again (Must be a Number) : ")
          (bind ?LossLim (read)) )
        (assert (Loss Limit ?LossLim))
        (assert (Action Profit Trend))
        (printout t ) )

(defrule First-Date
        ?date <-(Action Get First-Date)
        =>
        (retract ?date)
        (printout t t "Enter the Day (ie Mon, Tue Wed etc )  : ")
        (assert (Day =(read)))
        (printout t "Enter the Month (Jan Feb Mar Apr etc) : ")
        (assert (Month =(read)))
        (printout t t t) )
```

```
(defrule Date
        ?date <-(Action Get Date)
        ?day  <-(Day ?d)
        (Month ?Mon)
        =>
        (retract ?date ?day)
        (bind ?newd Tue)
        (if (eq Mon ?d) then
            (bind ?newd (str-cat "Tue"))
            (assert (Day Tue)))
        (if (eq Tue ?d) then
            (bind ?newd (str-cat "Wed"))
            (assert (Day Wed)))
        (if (eq Wed ?d) then
            (bind ?newd (str-cat "Thu"))
            (assert (Day Thu)))
        (if (eq Thu ?d) then
            (bind ?newd (str-cat "Fri"))
            (assert (Day Fri)))
        (if (eq Fri ?d) then
            (bind ?newd (str-cat "Mon"))
            (assert (Day Mon)))
        (bind ?mon (str-cat ?Mon))
        (format t  "%n   Day   is %s %n" ?newd)
        (format t  "   Month is %s %n" ?mon)
        (printout t t) )

;;*****************************************************************************
;;** Evaluate and Process Profit                                              **
;;*****************************************************************************

(defrule EvalProfit
        ?eval <-(Action Eval Profit)
        (Yesterdays ClosingPrice ?yclose)
        (Todays ClosingPrice ?tclose)
        (Investment ?invest)
        =>
        (retract ?eval)
        (assert (Todays Profit =(* (- ?tclose ?yclose) ?invest))) )

(defrule ProcProfit
        (Todays Profit ?prof)
        ?proc <-(Action Proc Profit)
        ?twp  <-(TwoWeeks Profit $?TWP)
        ?tp   <-(Total Profit ?totp)
        =>
        (retract ?twp ?tp ?proc)
        (assert (Total Profit =(+ ?totp ?prof)))
        (bind ?len (length $?TWP))
        (if (>= ?len 10) then
            (bind $?new (mv-delete 1 $?TWP)) )
        (bind ?scaleprof (/ ?prof 100))                    ;;** SCALE PROFIT
        (assert (TwoWeeks Profit =(mv-append $?new ?scaleprof))) )

;;*****************************************************************************
;;** Ascertain The Profit Trend From the PTA and QSI Neural Networks          **
;;*****************************************************************************

(defrule GetProfTrend
        ?act <-(Action Profit Trend)
        (TwoWeeks Profit $?TWP)
        =>
        (retract ?act)
        (printout t t ">>>==>>> Interrogating Neural Network to obtain Profit"crlf)
        (printout t   ">>>==>>> Trend for the last two weeks ......"crlf)
        (bind ?input (str-implode $?TWP))
        (bind ?string (Net_check "PTA" ?input))
        (bind $?output (str-explode ?string))
        (assert (Action ShareIndex Trend))
        (assert (NNOutput "PTA" $?output)) )

(defrule ProcNNOutput1
        ?proc <-(NNOutput "PTA" ?out $?more)
        (test (> ?out 0.4))
        (test (< ?out 0.6))
        =>
        (retract ?proc)
        (printout t t "Personal Profit Trend is Stable"crlf)
        (assert (Profit Trend stable)) )
```

```
(defrule ProcNNOutput2
        ?proc <-(NNOutput "PTA" ?out $?more)
        (test (> ?out 0.6))
        =>
        (retract ?proc)
        (printout t t "Personal Profit Trend is Increasing"crlf)
        (assert (Profit Trend increasing)) )

(defrule ProcNNOutput3
        ?proc <-(NNOutput "PTA" ?out $?more)
        (test (< ?out 0.4))
        =>
        (retract ?proc)
        (printout t t "Personal Profit Trend is Decreasing"crlf)
        (assert (Profit Trend decreasing)) )

;;*****************************************************************************
;;** Ascertain The Quarterly Share Index Trend from the Neural Network          **
;;*****************************************************************************

(defrule GetQuarterlyTrend
        ?act <-(Action ShareIndex Trend)
        (Quarterly ShareIndex $?QSI)
        =>
        (retract ?act)
        (printout t t ">>>==>>> Interrogating Neural Network to obtain Share"crlf)
        (printout t   ">>>==>>> Index Trend for the last quarter ......"crlf)
        (bind ?input (str-implode $?QSI))
        (bind ?string (Net_check "QSI" ?input))
        (bind $?output (str-explode ?string))
        (assert (NNOutput "QSI" $?output)) )

(defrule ProcQSIoutput1
        ?proc <-(NNOutput "QSI" ?out $?more)
        (test (> ?out 0.4))
        (test (< ?out 0.6))
        =>
        (retract ?proc)
        (printout t t "Quartly Share Index is Stable"crlf)
        (assert (Action Eval Position))
        (assert (ShareIndex Trend stable)) )

(defrule ProcQSIoutput2
        ?proc <-(NNOutput "QSI" ?out $?more)
        (test (> ?out 0.6))
        =>
        (retract ?proc)
        (printout t t "Quartly Share Index is Increasing"crlf)
        (assert (Action Eval Position))
        (assert (ShareIndex Trend increasing)) )

(defrule ProcQSIoutput3
        ?proc <-(NNOutput "QSI" ?out $?more)
        (test (< ?out 0.4))
        =>
        (retract ?proc)
        (printout t t "Quartly Share Index is Decreasing"crlf)
        (assert (Action Eval Position))
        (assert (ShareIndex Trend decreasing)) )

;;*****************************************************************************
;;** Change Risk Levels                                                       **
;;*****************************************************************************

(defrule Change_Risk1
        (Profit Trend increasing)
        (Yesterdays Profit Trend stable|increasing)
        (ShareIndex Trend increasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (> ?TP (* (/ 25 100) ?loss)))     ;;Profit > 25% of loss limit
        =>
        (assert (Decrease Risk 1.5 Levels)) )
```

```
(defrule Change_Risk2
        (Profit Trend increasing)
        (Yesterdays Profit Trend stable|increasing)
        (ShareIndex Trend stable|increasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (> ?TP (* (/ 10 100) ?loss)))     ;;Profit > 10% of loss limit
        (test (< ?TP (* (/ 25 100) ?loss)))     ;;AND < 25%
        =>
        (assert (Decrease Risk 1 Levels)) )

(defrule Change_Risk3
        (Profit Trend increasing)
        (ShareIndex Trend stable|increasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (> ?TP 0.0))                      ;;Profit > 0% of loss limit
        (test (< ?TP (* (/ 10 100) ?loss)))   ;;AND < 10%
        =>
        (assert (Decrease Risk 0.5 Levels)) )

(defrule Change_Risk4
        (Profit Trend decreasing)
        (Yesterdays Profit Trend stable|decreasing)
        (ShareIndex Trend decreasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (< ?TP 0.0))                      ;;Profit < 0% of loss limit
        (test (> ?TP (- 0 (* (/ 10 100) ?loss)))) ;;And > -10 of loss limit
        =>
        (assert (Increase Risk 0.5 Levels)) )

(defrule Change_Risk5
        (Profit Trend decreasing)
        (Yesterdays Profit Trend stable|decreasing)
        (ShareIndex Trend stable|decreasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (< ?TP (- 0 (* (/ 10 100) ?loss))))  ;;Profit < -10% of loss limit
        (test (> ?TP (- 0 (* (/ 25 100) ?loss))))  ;;And > -25 of loss limit
        =>
        (assert (Increase Risk 1 Levels)) )

(defrule Change_Risk6
        (Profit Trend decreasing)
        (ShareIndex Trend stable|decreasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (< ?TP (- 0 (* (/ 25 100) ?loss))))  ;;Profit <-25% of loss limit
        =>
        (assert (Increase Risk 1.5 Levels)) )

(defrule Change_Risk7
        (Profit Trend stable)
        (Yesterdays Profit Trend stable|decreasing)
        (ShareIndex Trend stable|decreasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (< ?TP 0.0))                      ;;Profit < 0% of loss limit
        =>
        (assert (Increase Risk 0.5 Levels)) )

(defrule Change_Risk8
        (Profit Trend stable)
        (Yesterdays Profit Trend stable|increasing)
        (ShareIndex Trend stable|increasing)
        (Loss Limit ?loss)
        (Todays Profit ?TP)
        (test (> ?TP 0.0))                      ;;Profit > 0% of loss limit
        =>
        (assert (Increase Risk 0.5 Levels)) )        ;;Maintain the same levels

(defrule Change_Risk9
        (Profit Trend decreasing)
        (Yesterdays Profit Trend stable|decreasing)
        (ShareIndex Trend stable|increasing)
        =>
        (assert (Increase Risk 0.5 Levels)) )
```

```
(defrule Change_Risk10
        (Profit Trend increasing)
        (Yesterdays Profit Trend stable|increasing)
        (ShareIndex Trend decreasing)
        =>
        (assert (Increase Risk 0.5 Levels)) )

(defrule Change_Risk11
        (Profit Trend stable)
        (Yesterdays Profit Trend stable|increasing)
        (ShareIndex Trend stable|increasing)
        =>
        (assert (Increase Risk 0 Levels)) )

;;*****************************************************************************
;;** Change Risk Levels depending on DAY and MONTH                            **
;;*****************************************************************************

(defrule Change_Risk9
        (Day Fri)
        =>
        (assert (Increase Risk 0.25 Levels)) )

(defrule Change_Risk10
        (Day Mon)
        =>
        (assert (Increase Risk 0.25 Levels)) )

(defrule Change_Risk11
        (Month Jan)
        =>
        (assert (Increase Risk 0.5 Levels)) )

(defrule Change_Risk12
        (Month Dec)
        =>
        (assert (Decrease Risk 0.5 Levels)) )

;;*****************************************************************************
;;** Evaluate position and Give recommendations                               **
;;*****************************************************************************

(defrule EvalPos1
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Current Risk Level ?lev)
        (test (> ?lev -2))
        (test (<= ?lev 3))
        =>
        (retract ?eval)
        (assert (Recommend Maintain Investment)) )

(defrule EvalPos2
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (>= ?TP 0.0))                    ;; Making Money
        (Current Risk Level ?lev)
        (test (> ?lev 3.0))
        (test (<= ?lev 5.0))
        =>
        (retract ?eval)
        (assert (Recommend Maintain Investment)) )

(defrule EvalPos3
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (>= ?TP 0.0))                    ;; Making Money
        (Current Risk Level ?lev)
        (test (> ?lev 5.0))
        (test (<= ?lev 7.0))
        =>
        (retract ?eval)
        (assert (Recommend Look To Take Profits)) )
```

```
(defrule EvalPos4
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (>= ?TP 0.0))                    ;; Making Money
        (Current Risk Level ?lev)
        (test (> ?lev 7.0))
        =>
        (retract ?eval)
        (assert (Recommend Take Profits At Once)) )

(defrule EvalPos5
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (< ?TP 0.0))                     ;; Losing Money
        (Current Risk Level ?lev)
        (test (> ?lev 2.0))
        (test (<= ?lev 4.0))
        =>
        (retract ?eval)
        (assert (Recommend Reduce Investment "25%")) )

(defrule EvalPos6
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (< ?TP 0.0))                     ;; Losing Money
        (Current Risk Level ?lev)
        (test (> ?lev 4.0))
        (test (<= ?lev 6.0))
        =>
        (retract ?eval)
        (assert (Recommend Reduce Investment "50%")) )

(defrule EvalPos7
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (< ?TP 0.0))                     ;; Losing Money
        (Current Risk Level ?lev)
        (test (> ?lev 6.0))
        =>
        (retract ?eval)
        (assert (Recommend Cut Losses At Once)) )

(defrule EvalPos8
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (< ?TP 0.0))                     ;; Losing Money
        (Current Risk Level ?lev)
        (test (<= ?lev -2.0))
        (test (> ?lev -4.0))
        =>
        (retract ?eval)
        (assert (Recommend Maintain Cautious Investment)) )

(defrule EvalPos9
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (< ?TP 0.0))                     ;; Losing Money
        (Current Risk Level ?lev)
        (test (<= ?lev -4.0))
        =>
        (retract ?eval)
        (assert (Recommend Cut Losses At Once)) )

(defrule EvalPos10
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (>= ?TP 0.0))                    ;; Making Money
        (Current Risk Level ?lev)
        (test (<= ?lev -2.0))
        (test (> ?lev -5.0))
        =>
        (retract ?eval)
        (assert (Recommend Take Profits OR Invest)) )
```

```
(defrule EvalPos11
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (>= ?TP 0.0))                    ;; Making Money
        (Current Risk Level ?lev)
        (test (<= ?lev -5.0))
        (test (> ?lev -7.0))
        =>
        (retract ?eval)
        (assert (Recommend Increase Investment)) )

(defrule EvalPos12
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (>= ?TP 0.0))                    ;; Making Money
        (Current Risk Level ?lev)
        (test (<= ?lev -7.0))
        (test (> ?lev -8.5))
        =>
        (retract ?eval)
        (assert (Recommend Look To Take Profits)) )

(defrule EvalPos13
        (declare (salience -5))
        ?eval <-(Action Eval Position)
        (Total Profit ?TP)
        (test (>= ?TP 0.0))                    ;; Making Money
        (Current Risk Level ?lev)
        (test (<= ?lev -8.5))
        =>
        (retract ?eval)
        (assert (Recommend Take Profits At Once)) )

;;*****************************************************************************
;;** Give Recommendation                                                      **
;;*****************************************************************************

(defrule Recomendation1
        (declare (salience -5))
        ?rec <-(Recommend Reduce Investment ?perc)
        (Total Profit ?prof)
        (Current Risk Level ?lev)
        =>
        (retract ?rec)
        (printout t t)
        (printout t "*********************************************************"crlf)
        (printout t "**                  RECOMMENDATIONS                    **"crlf)
        (printout t "**                                                     **"crlf)
        (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f        **%n" ?prof)
        (format   t "**           CURRENT RISK LEVEL    =      %4.2f        **%n" ?lev)
        (printout t "**                                                     **"crlf)
        (printout t "**  From the Information Supplied and the current     **"crlf)
        (printout t "**  trend in profit movement, it would be prudent     **"crlf)
        (printout t "**  to consider selling your shares in this           **"crlf)
        (printout t "**  company OR reducing your investment in this       **"crlf)
        (format   t "**  company by %s                                     **%n" ?perc)
        (printout t "**                                                     **"crlf)
        (printout t "*********************************************************"crlf))

(defrule Recomendation2
        (declare (salience -5))
        ?rec <-(Recommend Cut Losses At Once)
        (Total Profit ?prof)
        (Current Risk Level ?lev)
        =>
        (retract ?rec)
        (printout t t)
        (printout t "*********************************************************"crlf)
        (printout t "**                  RECOMMENDATIONS                    **"crlf)
        (printout t "**                                                     **"crlf)
        (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f        **%n" ?prof)
        (format   t "**           CURRENT RISK LEVEL    =      %4.2f        **%n" ?lev)
        (printout t "**                                                     **"crlf)
        (printout t "**  From the Information Supplied and the current     **"crlf)
        (printout t "**  trend in profit movement, it would be prudent     **"crlf)
        (printout t "**  to withdraw your investment in this company       **"crlf)
        (printout t "**  at once.                                           **"crlf)
        (printout t "*********************************************************"crlf))
```

```
(defrule Recomendation3
       (declare (salience -5))
       ?rec <-(Recommend Look To Take Profits)
       (Total Profit ?prof)
       (Current Risk Level ?lev)
       =>
       (retract ?rec)
       (printout t t)
       (printout t "*************************************************************"crlf)
       (printout t "**                  RECOMMENDATIONS                        **"crlf)
       (printout t "**                                                         **"crlf)
       (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f            **%n" ?prof)
       (format   t "**           CURRENT RISK LEVEL    =      %4.2f            **%n" ?lev)
       (printout t "**                                                         **"crlf)
       (printout t "**  From the Information Supplied and the current          **"crlf)
       (printout t "**  trend in profit movement, it would be advisable        **"crlf)
       (printout t "**  that you caustiously `look to take profits'.           **"crlf)
       (printout t "**                                                         **"crlf)
       (printout t "*************************************************************"crlf))

(defrule Recomendation4
       (declare (salience -5))
       ?rec <-(Recommend Take Profits At Once)
       (Total Profit ?prof)
       (Current Risk Level ?lev)
       =>
       (retract ?rec)
       (printout t t)
       (printout t "*************************************************************"crlf)
       (printout t "**                  RECOMMENDATIONS                        **"crlf)
       (printout t "**                                                         **"crlf)
       (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f            **%n" ?prof)
       (format   t "**           CURRENT RISK LEVEL    =      %4.2f            **%n" ?lev)
       (printout t "**                                                         **"crlf)
       (printout t "**  From the Information Supplied and the current          **"crlf)
       (printout t "**  trend in profit movement, it would be                  **"crlf)
       (printout t "**  recommended that you `look to take profits'            **"crlf)
       (printout t "**  immediatly.                                            **"crlf)
       (printout t "**                                                         **"crlf)
       (printout t "*************************************************************"crlf))

(defrule Recomendation5
       (declare (salience -5))
       ?rec <-(Recommend Maintain Investment)
       (Total Profit ?prof)
       (Current Risk Level ?lev)
       =>
       (retract ?rec)
       (printout t t)
       (printout t "*************************************************************"crlf)
       (printout t "**                  RECOMMENDATIONS                        **"crlf)
       (printout t "**                                                         **"crlf)
       (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f            **%n" ?prof)
       (format   t "**           CURRENT RISK LEVEL    =      %4.2f            **%n" ?lev)
       (printout t "**                                                         **"crlf)
       (printout t "**  From the Information Supplied and the current          **"crlf)
       (printout t "**  trend in profit movement, it would be                  **"crlf)
       (printout t "**  recommended that you maintain the current              **"crlf)
       (printout t "**  level of investment.                                   **"crlf)
       (printout t "**                                                         **"crlf)
       (printout t "*************************************************************"crlf))

(defrule Recomendation6
       (declare (salience -5))
       ?rec <-(Recommend Maintain Caustious Investment)
       (Total Profit ?prof)
       (Current Risk Level ?lev)
       =>
       (retract ?rec)
       (printout t t)
       (printout t "*************************************************************"crlf)
       (printout t "**                  RECOMMENDATIONS                        **"crlf)
       (printout t "**                                                         **"crlf)
       (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f            **%n" ?prof)
       (format   t "**           CURRENT RISK LEVEL    =      %4.2f            **%n" ?lev)
       (printout t "**                                                         **"crlf)
       (printout t "**  From the Information Supplied and the current          **"crlf)
       (printout t "**  trend in profit movement, it would be                  **"crlf)
       (printout t "**  recommended that you maintain the current              **"crlf)
       (printout t "**  level of investment but be caustious about the        **"crlf)
```

```
        (printout t "**  the performance of the company over the next few **"crlf)
        (printout t "**  days.                                              **"crlf)
        (printout t "**                                                     **"crlf)
        (printout t "*********************************************************"crlf))

(defrule Recomendation7
        (declare (salience -5))
        ?rec <-(Recommend Take Profits OR Invest)
        (Total Profit ?prof)
        (Current Risk Level ?lev)
        =>
        (retract ?rec)
        (printout t t)
        (printout t "*********************************************************"crlf)
        (printout t "**                  RECOMMENDATIONS                    **"crlf)
        (printout t "**                                                     **"crlf)
        (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f        **%n" ?prof)
        (format   t "**           CURRENT RISK LEVEL    =      %4.2f        **%n" ?lev)
        (printout t "**                                                     **"crlf)
        (printout t "**  From the Information Supplied and the current      **"crlf)
        (printout t "**  trend in profit movement, it would be prudent      **"crlf)
        (printout t "**  to consider 'taking ones profits' in  this         **"crlf)
        (printout t "**  company OR increasing your investment in this      **"crlf)
        (printout t "**  company.                                           **"crlf)
        (printout t "**                                                     **"crlf)
        (printout t "*********************************************************"crlf))

(defrule Recomendation6
        (declare (salience -5))
        ?rec <-(Recommend Increase Investment)
        (Total Profit ?prof)
        (Current Risk Level ?lev)
        =>
        (retract ?rec)
        (printout t t)
        (printout t "*********************************************************"crlf)
        (printout t "**                  RECOMMENDATIONS                    **"crlf)
        (printout t "**                                                     **"crlf)
        (format   t "**           TOTAL PROFIT TO DATE  =  $ %6.2f        **%n" ?prof)
        (format   t "**           CURRENT RISK LEVEL    =      %4.2f        **%n" ?lev)
        (printout t "**                                                     **"crlf)
        (printout t "**  From the Information Supplied and the current      **"crlf)
        (printout t "**  trend in profit movement, it would be              **"crlf)
        (printout t "**  recommended that you increase the current          **"crlf)
        (printout t "**  level of investment in this company.               **"crlf)
        (printout t "**                                                     **"crlf)
        (printout t "*********************************************************"crlf))

;;*****************************************************************************
;;** Utility Rules                                                           **
;;*****************************************************************************

(defrule PrintBanner
        ?print <- (Action Print Banner)
        (Investment ?invest)
        (Sum Invested ?sum)
        (Yesterdays ClosingPrice ?close)
        (Total Profit ?prof)
        (Current Risk Level ?lev)
        =>
        (retract ?print)
        (printout t t t t t t t t t t t t t t t)
        (printout t "*********************************************************"crlf)
        (printout t "**                                                     **"crlf)
        (printout t "**    P R O F I T   T R E N D   A N A L A Y S I S      **"crlf)
        (printout t "**                                                     **"crlf)
        (printout t "**              C O N S U L T A T I O N                **"crlf)
        (printout t "**                                                     **"crlf)
        (printout t "**                   S Y S T E M                       **"crlf)
        (printout t "**                                                     **"crlf)
        (printout t "*********************************************************"crlf)
        (printout t t t)
        (printout t "+-----------------------------------------------------+"crlf)
        (format   t "|    INITIAL SUM INVESTED             |    $ %6.2f   |%n" ?sum)
        (format   t "|    NUMBER OF SHARES INVESTED IN     |     %4d      |%n" ?invest)
        (printout t "+-------------------------------------+---------------+"crlf)
        (format   t "|    YESTERDAYS SHARE PRICE           |    $ %6.2f   |%n" ?close)
        (format   t "|    TOTAL PROFIT TO DATE             |    $ %6.2f   |%n" ?prof)
        (format   t "|    CURRENT RISK LEVEL               |        %4.2f   |%n" ?lev)
        (printout t "+-----------------------------------------------------+"crlf))
```

```
(defrule Decrease_Risk
        ?decrease  <-(Decrease Risk ?dec Levels)
        ?old_level <-(Current Risk Level ?level)
        =>
        (retract ?decrease)
        (retract ?old_level)
        (assert (Current Risk Level =(- ?level ?dec))) )

(defrule Increase_Risk
        ?increase  <-(Increase Risk ?inc Levels)
        ?old_level <-(Current Risk Level ?level)
        =>
        (retract ?increase)
        (retract ?old_level)
        (assert (Current Risk Level =(+ ?level ?inc))) )

(defrule ProcAvWSI
        ?nod  <-(Number Of Days ?num)
        (test (= ?num 5))
        ?wsi  <-(Average Weekly Share Index ?WSI)
        ?qtr  <-(Quarterly ShareIndex $?QSI)
        =>
        (retract ?nod ?wsi ?qtr)
        (assert (Average Weekly Share Index 0.0))
        (assert (Number Of Days 0))
        (bind ?len (length $?QSI))
        (if (>= ?len 12) then
        (bind $?new (mv-delete 1 $?QSI)) )
        (bind ?Av (/ (/ ?WSI 5) 10 ))                          ;;Scale +10-10
        (assert (Quarterly ShareIndex =(mv-append $?new ?Av))) )

;;*****************************************************************************
;;** Phase Control Rules                                                      **
;;*****************************************************************************

(defrule startup
        =>
        (printout t "LOADING APPROPRIATE NEURAL NETWORKS INTO ENVIRONMENT"crlf)
        (printout t "Please wait ........."crlf)
        (Net_win "PTA" "icon")
        (Net_win "QSI" "icon")
        (Net_load "PTA" "PTA.bep")
        (Net_load "QSI" "QSI.bep")
        (assert (Action Proc Profit))
        (assert (Action Eval Profit))
        (assert (Action Get Loss))
        (assert (Action Get Closing))
        (assert (Action Get First-Date))
        (assert (Action Print Banner))
        (assert (Action Start Again)) )

(defrule StartAgain
        (declare (salience -10))
        ?again <-(Action Start Again)
        ?yestd <-(Yesterdays ClosingPrice ?yclose)
        ?yestP <-(Yesterdays Profit Trend ?ypt)
        ?tpt   <-(Profit Trend ?TPT)
        ?today <-(Todays ClosingPrice ?tclose)
        ?todP  <-(Todays Profit ?P)
        ?loss  <-(Loss Limit ?l)
        ?awsi  <-(Average Weekly Share Index ?WSI)
        ?nod   <-(Number Of Days ?NOD)
        ?sit   <-(ShareIndex Trend ?trend)
        =>
        (printout t t t "Do you want to continue with the consultation (y/n) : ")
        (bind ?yesno (read))
        (while (and (neq ?yesno y) (neq ?yesno n))
          (printout t "Do you want to continue with the consultation (y/n) : ")
          (bind ?yesno (read)) )
        (if (eq ?yesno y) then
            (retract ?again ?yestd ?yestP ?nod ?awsi ?todP ?loss ?sit)
            (assert (Yesterdays ClosingPrice ?tclose))
            (assert (Yesterdays Profit Trend ?TPT))
            (assert (Number Of Days =(+ ?NOD 1)))
            (assert (Average Weekly Share Index =(+ ?WSI ?tclose)))
            (retract ?today ?tpt)
            (assert (Action Proc Profit))
            (assert (Action Eval Profit))
            (assert (Action Get Loss))
            (assert (Action Get Closing))
```

```
        (assert (Action Get Date))
        (assert (Action Print Banner))
        (assert (Action Start Again))
    ))
```

Appendix D

Rulebase For British Airways Cargo Consignment Application

```
;;*************************************************************************
;;**                                                                     **
;;**  British Airways Cargo Consigment Application Knowledge Base       **
;;**  File Name      : BACARGO.CLP                                      **
;;**  Date Changed   : 04/09/94                                         **
;;**                                                                     **
;;*************************************************************************

;;*********************************
;;* Set up initial facts          *
;;*********************************

(deffacts Defaults
     (ASP Standard Score 100)
     (Data Input File n)
     (Favoured Station ARN BCN BRU BSL CDG DUS FRA HAM LHR LIN MAN MST GOT ARN
                       JFK LYS TLV)
     (Weight/Volume Rating 9.0)
     (Density Weighting 22.9)
     (Rate Weighting -2.5)
     (Urgency Weighting 67.625)
     (AKLNetScale AVLWGHT -1318 10007)       ;;** Scaling factors for AKL Net
     (AKLNetScale AVLVOL  -5  49)
     (AKLNetScale CNSWGT  11  1089)
     (AKLNetScale CNSVOL  0.04 12.34)
     (AKLNetScale RateDiff -0.77 3.32)
     (AKLNetScale DaysToFlt -12 15)
     (MELNetScale AVLWGHT -826 18156)   ;;** Scaling factors for MEL Net
     (MELNetScale AVLVOL  0 43)
     (MELNetScale CNSWGT  1 1499)
     (MELNetScale CNSVOL  0.01 9.99)
     (MELNetScale RateDiff -3.61 4.32)
     (MELNetScale DaysToFlt -12  12)
     (SYDNetScale AVLWGHT -1052 13919)  ;;** Scaling factors for SYD Net
     (SYDNetScale AVLVOL   0 54)
     (SYDNetScale CNSWGT  75 5915)
     (SYDNetScale CNSVOL  0.07 58.1)
     (SYDNetScale RateDiff -1.33 1.78)
     (SYDNetScale DaysToFlt -12 26)
     (Correct Matches 0)
     (Request Count 0)
     )

;;*************************************************************************
;;**  Main Processing Rules                                              **
;;*************************************************************************

;;****************************************
;;* Get Cargo Consigment Request Details  *
;;****************************************

;;** Get Days-Until-Flight

(defrule Days-Until-Flight
        ?daysuntilflight <-(Action Get Days Until Flight)
        =>
        (retract ?daysuntilflight)
        (printout t t "Enter the number of days until the flight : ")
        (bind ?DaysUntilFlight (read))
        (while (not (numberp ?DaysUntilFlight))
          (printout t "Enter the days until the flight Again (Must be a Number) : ")
          (bind ?DaysUntilFlight (read)) )
        (assert (Days Until Flight ?DaysUntilFlight))
        (printout t ) )
```

```
;;** Get Station

(defrule Station
        ?station <-(Action Get Station)
        =>
        (retract ?station)
        (printout t t "Enter the Station requesting consignment : ")
        (bind ?Station (read))
        (assert (Requesting Station ?Station))
        (printout t ) )

(defrule Destination
        ?dest <-(Action Get Destination)
        =>
        (retract ?dest)
        (printout t t "Enter the Destination : ")
        (bind ?Destin (read))
        (assert (Destination ?Destin))
        (printout t ) )

(defrule Avail-Weight
        ?av-wght <-(Action Get Avail-Weight)
        =>
        (retract ?av-wght)
        (printout t t "Enter the Available Wieght on the plane : ")
        (bind ?Avail-Wght (read))
        (while (not (numberp ?Avail-Wght))
          (printout t "Enter the Available Weight Again (Must be a Number) : ")
          (bind ?Avail-Wght (read)) )
        (assert (Available Weight ?Avail-Wght))
        (printout t ) )

(defrule Avail-Volume
        ?av-vol <-(Action Get Avail-Volume)
        =>
        (retract ?av-vol)
        (printout t t "Enter the Available Volume on the plane : ")
        (bind ?Avail-Vol (read))
        (while (not (numberp ?Avail-Vol))
          (printout t "Enter the Available Volume Again (Must be a Number) : ")
          (bind ?Avail-Vol (read)) )
        (assert (Available Volume ?Avail-Vol))
        (printout t) )

(defrule Avail-Density1
       (Available Volume ?Avail-Vol)
       (Available Weight ?Avail-Wght)
       (test (eq ?Avail-Vol 0))
       =>
       (assert (Available Density 0))
       (assert (Density Total 1.5)) )

(defrule Avail-Density2
       (Available Volume ?Avail-Vol)
       (Available Weight ?Avail-Wght)
       (test (neq ?Avail-Vol 0))
       =>
       (bind ?density (/ ?Avail-Wght ?Avail-Vol))
       (assert (Available Density ?density)) )

(defrule Consign-Weight
        ?con-wght <-(Action Get Consign-Weight)
        =>
        (retract ?con-wght)
        (printout t t "Enter the Consignment Weight of the Cargo : ")
        (bind ?Con-Wght (read))
        (while (not (numberp ?Con-Wght))
          (printout t "Enter the Consigment Weight Again (Must be a Number) : ")
          (bind ?Con-Wght (read)) )
        (assert (Consignment Weight ?Con-Wght))
        (printout t ) )

(defrule Consign-Volume
        ?con-vol <-(Action Get Consign-Volume)
        =>
        (retract ?con-vol)
        (printout t t "Enter the Consignment Volume of the Cargo : ")
        (bind ?Con-Vol (read))
        (while (not (numberp ?Con-Vol))
          (printout t "Enter the Consigment Volume Again (Must be a Number) : ")
```

```
          (bind ?Con-Vol (read)) )
         (assert (Consignment Volume ?Con-Vol))
         (printout t ) )

(defrule Consign-Density1
        (Consignment Volume ?Consign-Vol)
        (Consignment Weight ?Consign-Wght)
        (test (eq ?Consign-Vol 0))
        =>
        (assert (Consignment Density 0))
        (assert (Density Total 1)) )

(defrule Consign-Density2
        (Consignment Volume ?Consign-Vol)
        (Consignment Weight ?Consign-Wght)
        (test (neq ?Consign-Vol 0))
        =>
        (bind ?density (/ ?Consign-Wght ?Consign-Vol))
        (assert (Consignment Density ?density)) )

(defrule Consign-Rate
        ?con-rate <-(Action Get Consign-Rate)
        =>
        (retract ?con-rate)
        (printout t t "Enter the Consignment Rate of the Cargo : ")
        (bind ?Con-Rate (read))
        (while (not (numberp ?Con-Rate))
          (printout t "Enter the Consigment Rate Again (Must be a Number) : ")
          (bind ?Con-Rate (read)) )
        (assert (Consignment Rate ?Con-Rate))
        (printout t ) )

(defrule Standard-Rate
        ?stand-rate <-(Action Get Standard-Rate)
        =>
        (retract ?stand-rate)
        (printout t t "Enter the Standard Rate on the plane : ")
        (bind ?Stand-Rate (read))
        (while (not (numberp ?Stand-Rate))
          (printout t "Enter the Standard Rate Again (Must be a Number) : ")
          (bind ?Stand-Rate (read)) )
        (assert (Standard Rate ?Stand-Rate))
        (printout t ) )

;;*************************************************************************
;;** Phase Control Rules                                                  **
;;*************************************************************************

(defrule startup
        =>
        (printout t "LOADING APPROPRIATE NEURAL NETWORKS INTO ENVIRONMENT"crlf)
        (printout t "Please wait ........."crlf)
        (Net_win "AKL" "icon")
        (Net_win "SYD" "icon")
        (Net_win "MEL" "icon")
        (Net_load "AKL" "AKL.bep")
        (Net_load "SYD" "SYD.bep")
        (Net_load "MEL" "MEL.bep")
       (close)                      ;;Close File Pointers that may be open
       (assert (Action File Or Input))
        (assert (Action Print Banner))
        (assert (Action Start Again)) )

(defrule StartAgain
        (declare (salience -10))
        ?again   <- (Action Start Again)
        ?flgdays <- (Days Until Flight ?DaysUntilFlight)
        ?stat    <- (Requesting Station ?Station)
        ?dest    <- (Destination ?destin)
        ?avlwght <- (Available Weight ?Avail-Wght)
        ?avlvol  <- (Available Volume ?Avail-Vol)
        ?avlden  <- (Available Density ?Avail-Den)
        ?conwght <- (Consignment Weight ?Con-Wght)
        ?convol  <- (Consignment Volume ?Con-Vol)
        ?conden  <- (Consignment Density ?Con-den)
        ?conrate <- (Consignment Rate ?Con-Rate)
        ?stndrat <- (Standard Rate ?Stand-Rate)
        ?score   <- (Consignment Score ?cscore)
        ?recom   <- (Recommendation ?recommend)
        ?w/vt    <- (Weight/Volume Total ?w/v)
```

```
        ?dent    <- (Density Total ?denstot)
        ?ratt    <- (Rate Total ?rattot)
        ?urgt    <- (Urgency Total ?urgtot)
        ?input   <- (Input $?Input)
        ?hum     <- (HUMAN Decision ?dec)
        ?nno     <- (NNOutput ?nnout)
        ?nnacrj  <- (NNResult ?acrj)
        =>
;;    Uncomment if required to stop end of each consultation
;;       (printout t t t "Process another request (y/n) : ")
;;       (bind ?yesno (read))
;;       (while (and (neq ?yesno y) (neq ?yesno n))
;;         (printout t "Process another request (y/n) : ")
;;         (bind ?yesno (read)) )
;;       (printout t)
;;       (if (eq ?yesno y) then
           (retract ?again ?flgdays ?stat ?avlwght ?avlvol ?avlden ?conwght
                    ?convol ?conden ?conrate ?stndrat ?score ?recom ?score
                    ?recom ?w/vt ?dent ?ratt ?urgt ?input ?hum ?nno ?nnacrj ?dest)
           (assert (Action File Or Input))
           (assert (Action Start Again))
;;       else
;;         (close) )
      )

;;*****************************************************************************
;;* File I/O Section
;;* Ask if File input or Data Input Functionality required
;;*****************************************************************************

(defrule FileExcist
        ?fileinp <- (Action File Or Input)
        (Data Input File ?yesno)
        (test (eq ?yesno y))
        =>
        (retract ?fileinp)
        (bind ?input (readline file))
        (bind ?inpfact (str-cat "Input " ?input))
        (str-assert ?inpfact) )

(defrule FileOrInput
        ?fileinp <- (Action File Or Input)
        ?data <- (Data Input File ?yn)
        (test (eq ?yn n))
        =>
        (retract ?fileinp ?data)
        (printout t t "Accept Data from a File (y/n) : ")
        (bind ?yesno (read))
        (while (and (neq ?yesno y) (neq ?yesno n))
          (printout t "Accept Data from a File (y/n) : ")
          (bind ?yesno (read)) )
        (printout t)
        (assert (Data Input File ?yesno))
        (if (eq ?yesno y) then
            (printout t "Enter Name of the File to Read (\"*.txt\") : ")
            (bind ?fname (read))
            (printout t)
            (while (eq =(open ?fname file "r") 0)
                (printout t "Enter Name of the File to Read (\"*.txt\") : ")
                (bind ?fname (read))
                (printout t) )

            (bind ?input (readline file))
            (bind ?inpfact (str-cat "Input " ?input))
            (str-assert ?inpfact)
         else
            (assert (Action Calculate Consignment Score))
            (assert (Action Calculate Urgency Total))
            (assert (Action Calculate Rate Total))
            (assert (Action Calculate Density Total))
            (assert (Action Calculate Weight/Volume Total))
            (assert (Action Get Consign-Rate))
            (assert (Action Get Consign-Volume))
            (assert (Action Get Consign-Weight))
            (assert (Action Get Standard-Rate))
            (assert (Action Get Avail-Volume))
            (assert (Action Get Avail-Weight))
            (assert (Action Get Destination))
            (assert (Action Get Station))
            (assert (Action Get Days Until Flight))
```

```
            (assert (Action Print Banner))
            (assert (Action Start Again)) ) )

(defrule FileInputAction
        (Input ?dtf ?sta ?dest ?avlw ?avlv ?stdra ?conw ?conv ?conra ?adec ?asp)
        =>
        (format t "%nDTF= %d, STA= %s, DEST= %s, AVLW= %d, AVLV= %4.2f, STR= %4.2f
                  CONW= %4.2f, CONV= %4.2f, CONR= %4.2f %n" ?dtf ?sta ?dest ?avlw
                  ?avlv ?stdra ?conw ?conv ?conra)
                                  ;;** If conrate == 0 then
                                  ;;**    conrate = standrate
         (if (eq ?conra 0.0) then
             (assert (Consignment Rate ?stdra))
         else
             (assert (Consignment Rate ?conra)) )
         (assert (Action Calculate Consignment Score))
         (assert (Action Calculate Urgency Total))
         (assert (Action Calculate Rate Total))
         (assert (Action Calculate Density Total))
         (assert (Action Calculate Weight/Volume Total))
         (assert (HUMAN Decision ?adec))
         (assert (Consignment Volume ?conv))
         (assert (Consignment Weight ?conw))
         (assert (Standard Rate ?stdra))
         (assert (Available Volume ?avlv))
         (assert (Available Weight ?avlw))
         (assert (Destination ?dest))
         (assert (Requesting Station ?sta))
         (assert (Days Until Flight ?dtf))
         (assert (ASP Score ?asp)) )

;;*****************************************************************************
;;* Control rules for Scaling Inputs to Neural Networks based on Stations
;;* and post processing the results from the Neural Networks
;;*****************************************************************************

(defrule GetNetAnswerAKL
   (Destination ?dest)
   (test (eq ?dest AKL))
   (Consignment Volume ?conv)
   (Consignment Weight ?conw)
   (Consignment Rate ?conra)
   (Standard Rate ?stdra)
   (Available Volume ?avlv)
   (Available Weight ?avlw)
   (Days Until Flight ?dtf)
   (AKLNetScale AVLWGHT ?avwmin ?avwd)
   (AKLNetScale AVLVOL  ?avvmin ?avvd)
   (AKLNetScale CNSWGT  ?cnwmin ?cnwd)
   (AKLNetScale CNSVOL  ?cnvmin ?cnvd)
   (AKLNetScale RateDiff ?ramin ?ratd)
   (AKLNetScale DaysToFlt ?dfmin ?dfd)
     =>
   (printout t ">>>> Interogating AKL Neural Network <<<<"crlf)
   (bind ?rtdif (- ?stdra ?conra))
   (bind ?savlw (+ (* (/ 0.8 ?avwd) ?avlw) (- 0.1 (* (/ 0.8 ?avwd) ?avwmin))))
   (bind ?savlv (+ (* (/ 0.8 ?avvd) ?avlv) (- 0.1 (* (/ 0.8 ?avvd) ?avvmin))))
   (bind ?scnsw (+ (* (/ 0.8 ?cnwd) ?conw) (- 0.1 (* (/ 0.8 ?cnwd) ?cnwmin))))
   (bind ?scnsv (+ (* (/ 0.8 ?cnvd) ?conv) (- 0.1 (* (/ 0.8 ?cnvd) ?cnvmin))))
   (bind ?sratd (+ (* (/ 0.8 ?ratd) ?rtdif) (- 0.1 (* (/ 0.8 ?ratd) ?ramin))))
   (bind ?sdtf  (+ (* (/ 0.8 ?dfd) ?dtf) (- 0.1 (* (/ 0.8 ?dfd) ?dfmin))))
   (bind $?inp (mv-append ?savlw ?savlv ?scnsw ?scnsv ?sratd ?sdtf))
   (bind ?nninpstr (str-implode $?inp))
;;   (format t "NNINPUT => %s" ?nninpstr)
   (bind ?output (Net_check "AKL" ?nninpstr))
   (bind ?outfact (str-cat "NNOutput " ?output))
   (str-assert ?outfact) )

(defrule GetNetAnswerSYD
   (Destination ?dest)
   (test (eq ?dest SYD))
   (Consignment Volume ?conv)
   (Consignment Weight ?conw)
   (Consignment Rate ?conra)
   (Standard Rate ?stdra)
   (Available Volume ?avlv)
   (Available Weight ?avlw)
   (Days Until Flight ?dtf)
   (SYDNetScale AVLWGHT ?avwmin ?avwd)
   (SYDNetScale AVLVOL  ?avvmin ?avvd)
```

```
   (SYDNetScale CNSWGT  ?cnwmin ?cnwd)
   (SYDNetScale CNSVOL  ?cnvmin ?cnvd)
   (SYDNetScale RateDiff ?ramin ?ratd)
   (SYDNetScale DaysToFlt ?dfmin ?dfd)
     =>
   (printout t ">>>> Interogating SYD Neural Network <<<<"crlf)
   (bind ?rtdif (- ?stdra ?conra))
   (bind ?savlw (+ (* (/ 0.8 ?avwd) ?avlw) (- 0.1 (* (/ 0.8 ?avwd) ?avwmin))))
   (bind ?savlv (+ (* (/ 0.8 ?avvd) ?avlv) (- 0.1 (* (/ 0.8 ?avvd) ?avvmin))))
   (bind ?scnsw (+ (* (/ 0.8 ?cnwd) ?conw) (- 0.1 (* (/ 0.8 ?cnwd) ?cnwmin))))
   (bind ?scnsv (+ (* (/ 0.8 ?cnvd) ?conv) (- 0.1 (* (/ 0.8 ?cnvd) ?cnvmin))))
   (bind ?sratd (+ (* (/ 0.8 ?ratd) ?rtdif) (- 0.1 (* (/ 0.8 ?ratd) ?ramin))))
   (bind ?sdtf  (+ (* (/ 0.8 ?dfd) ?dtf) (- 0.1 (* (/ 0.8 ?dfd) ?dfmin))))
   (bind $?inp (mv-append ?savlw ?savlv ?scnsw ?scnsv ?sratd ?sdtf))
   (bind ?nninpstr (str-implode $?inp))
   (bind ?output (Net_check "SYD" ?nninpstr))
   (bind ?outfact (str-cat "NNOutput " ?output))
   (str-assert ?outfact) )

(defrule GetNetAnswerMEL
   (Destination ?dest)
   (test (eq ?dest MEL))
   (Consignment Volume ?conv)
   (Consignment Weight ?conw)
   (Consignment Rate ?conra)
   (Standard Rate ?stdra)
   (Available Volume ?avlv)
   (Available Weight ?avlw)
   (Days Until Flight ?dtf)
   (MELNetScale AVLWGHT ?avwmin ?avwd)
   (MELNetScale AVLVOL  ?avvmin ?avvd)
   (MELNetScale CNSWGT  ?cnwmin ?cnwd)
   (MELNetScale CNSVOL  ?cnvmin ?cnvd)
   (MELNetScale RateDiff ?ramin ?ratd)
   (MELNetScale DaysToFlt ?dfmin ?dfd)
   =>
   (printout t ">>>> Interogating MEL Neural Network <<<<"crlf)
   (bind ?rtdif (- ?stdra ?conra))
   (bind ?savlw (+ (* (/ 0.8 ?avwd) ?avlw) (- 0.1 (* (/ 0.8 ?avwd) ?avwmin))))
   (bind ?savlv (+ (* (/ 0.8 ?avvd) ?avlv) (- 0.1 (* (/ 0.8 ?avvd) ?avvmin))))
   (bind ?scnsw (+ (* (/ 0.8 ?cnwd) ?conw) (- 0.1 (* (/ 0.8 ?cnwd) ?cnwmin))))
   (bind ?scnsv (+ (* (/ 0.8 ?cnvd) ?conv) (- 0.1 (* (/ 0.8 ?cnvd) ?cnvmin))))
   (bind ?sratd (+ (* (/ 0.8 ?ratd) ?rtdif) (- 0.1 (* (/ 0.8 ?ratd) ?ramin))))
   (bind ?sdtf  (+ (* (/ 0.8 ?dfd) ?dtf) (- 0.1 (* (/ 0.8 ?dfd) ?dfmin))))
   (bind $?inp (mv-append ?savlw ?savlv ?scnsw ?scnsv ?sratd ?sdtf))
   (bind ?nninpstr (str-implode $?inp))
   (bind ?output (Net_check "MEL" ?nninpstr))
   (bind ?outfact (str-cat "NNOutput " ?output))
   (str-assert ?outfact) )

(defrule NNAcceptReject
;;   ?nno <- (NNOutput ?output)
     (NNOutput ?output)
     =>
;;     (retract ?nno)
     (if (>= ?output 0.7) then
         (assert (NNResult ACCEPT))
         (printout t "Neural Network Result ACCEPT"crlf)
     else (if (<= ?output 0.3) then
               (assert (NNResult REJECT))
               (printout t "Neural Network Result REJECT"crlf)
          else
               (assert (NNResult REJECT))
               (printout t "Neural Network Result REJECT"crlf)) ) )

;;*****************************************************************************
;;* Control rule for Calculating the Consigment Score
;;* and for ACCEPTing and REJECTing based on Consigment score
;;*****************************************************************************

;;*Consignment Score = ( Weight/Volume Total  x  weight/volume rating +
;;*                      Density total        x  density weighting +
;;*                      Rate total           x  rate weighting +
;;*                      Container total      x  container weighting +
;;*                      Urgency total        x  urgency weighting )
;;*                     * 100 / standard score
```

```
(defrule Calc-Consign-Score
        ?consign <-  (Action Calculate Consignment Score)
        (Weight/Volume Total ?w/vtotal)
        (Weight/Volume Rating ?w/vrating)
        (Density Total ?dentotal)
        (Density Weighting ?denweight)
        (Rate Total ?ratetotal)
        (Rate Weighting ?rateweighting)
        (Urgency Total ?urgtotal)
        (Urgency Weighting ?urgweighting)
        (ASP Standard Score ?standscore)
        =>
        (retract ?consign)
        (bind ?consc1 (+ (* ?w/vtotal ?w/vrating) (* ?dentotal ?denweight)) )
        (bind ?consc2 (+ (* ?ratetotal ?rateweighting) (* ?urgtotal ?urgweighting)) )
        (bind ?consub (+ ?consc2 ?consc1) )
        (bind ?conscore (/ (* ?consub 100) ?standscore) )
        (format t "Consignment Score => %4.2f" ?conscore)
        (printout t t " "crlf)
        (assert (Consignment Score ?conscore) )
        (assert (Action Decision Control) )  )

;;*************************************************************************
;;** Decision Control for ACCEPT or REJECT                                **
;;** Useing Neural Networks & Expert System                              **
;;*************************************************************************

(defrule Check-Weight1
     ?decis <- (Action Decision Control)
     (Consignment Weight ?con-wght)
     (NNResult ?nnresult)
     (Available Weight ?avl-wght)
     (Consignment Score ?conscore)
     (ASP Standard Score ?stanscore)
     (test (> ?con-wght ?avl-wght))
     (test (>= ?conscore (- ?stanscore 5)) )
     (test (<= ?conscore (+ ?stanscore 5)) )
     =>                              ;; If avlw or avlv < conw or conv
     (retract ?decis)                ;; Then use Neural Networks Results
     (if (eq ?nnresult REJECT) then
         (assert (Recommendation ACCEPT))
     else (assert (Recommendation ?nnresult)) )
     (assert (Action Print Results)) )

(defrule Check-Volume1
     ?decis <- (Action Decision Control)
     (Consignment Volume ?con-vol)
     (NNResult ?nnresult)
     (Available Volume ?avl-vol)
     (Consignment Score ?conscore)
     (ASP Standard Score ?stanscore)
     (test (> ?con-vol ?avl-vol))
     (test (>= ?conscore (- ?stanscore 5)) )
     (test (<= ?conscore (+ ?stanscore 5)) )
     =>                              ;; If avlw or avlv < conw or conv
     (retract ?decis)                ;; Then use Neural Networks Result
     (if (eq ?nnresult REJECT) then
         (assert (Recommendation ACCEPT))
     else (assert (Recommendation ?nnresult)) )
     (assert (Action Print Results)) )

(defrule Check-Weight2
     ?decis <- (Action Decision Control)
     (Consignment Weight ?con-wght)
     (NNResult ?nnresult)
     (Available Weight ?avl-wght)
     (test (> ?con-wght ?avl-wght))
     =>                              ;; If avlw or avlv < conw or conv
     (retract ?decis)                ;; Then use Neural Networks Results
     (assert (Recommendation ?nnresult))
     (assert (Action Print Results)) )

(defrule Check-Volume2
     ?decis <- (Action Decision Control)
     (Consignment Volume ?con-vol)
     (NNResult ?nnresult)
     (Available Volume ?avl-vol)
     (test (> ?con-vol ?avl-vol))
     =>                              ;; If avlw or avlv < conw or conv
     (retract ?decis)                ;; Then use Neural Networks Result
```

```
     (assert (Recommendation ?nnresult))
     (assert (Action Print Results)) )

;;*If consignment score >= standard score   (CURRENTLY STANDARD SCORE = 100)
;;*        recommend ACCEPT
;;*otherwise
;;*        recommend REJECT

(defrule Consign-Decision1
        ?decis <- (Action Decision Control)
        (Consignment Score ?conscore)
        (ASP Standard Score ?stanscore)
        (test (>= ?conscore (+ ?stanscore 5)) )
        =>
        (retract ?decis)
        (assert (Recommendation ACCEPT))
        (assert (Action Print Results)) )

(defrule Consign-Desision2
        ?decis <- (Action Decision Control)
        (Consignment Score ?conscore)
        (ASP Standard Score ?stanscore)
        (NNResult ?nnresult)
        (test (>= ?conscore (- ?stanscore 5)) )
        (test (<= ?conscore (+ ?stanscore 5)) )
        =>
        (retract ?decis)
        (if (eq ?nnresult REJECT) then           ;; Use Neural Networks Results
            (assert (Recommendation ACCEPT))
        else (assert (Recommendation ?nnresult)) )
        (assert (Action Print Results)) )

(defrule Consign-Decision3
        ?decis <- (Action Decision Control)
        (Consignment Score ?conscore)
        (ASP Standard Score ?stanscore)
        (test (< ?conscore (- ?stanscore 5)) )
        =>
        (retract ?decis)
        (assert (Recommendation REJECT))
        (assert (Action Print Results)) )

;;*****************************************************************************
;;** Weight/Volume Total
;;*****************************************************************************
;;*If (flight departs within 48 hours) &
;;*   (available weight > 1000) &
;;*   (available volume > 5)
;;*then
;;*    weight/volume total = 1.5
;;*else if (flight departs in more than 48 hours) &
;;*        (avail wgt < 1000) &
;;*        (avail vol < 5)
;;*     then
;;*        weight/volume total = 0.5
;;*     else
;;*        weight/volume total = 1.0

(defrule Wght/VolTotal
         ?w/vtotal  <- (Action Calculate Weight/Volume Total)
        (Days Until Flight ?dayflight)
        (Available Weight ?avl-weight)
        (Available Volume ?avl-volume)
        =>
        (retract ?w/vtotal)
        (if (and (and (> ?dayflight -2)(> ?avl-weight 1000)) (> ?avl-volume 5))
        then
          (assert (Weight/Volume Total 1.5) )
        else
           (if (and (and (< ?dayflight -2)(< ?avl-weight 1000)) (< ?avl-volume 5))
           then
               (assert (Weight/Volume Total 0.5) )
           else
           (assert (Weight/Volume Total 1.0) ) ) ) )
```

```
;;*****************************************************************************
;;** Density Total
;;*****************************************************************************
;;* If available density is greater than 167
;;* then the flight is considered volume-restricted.
;;*
;;* This means that volume is more at a premium then weight so we want to
;;* prefer small volume, high weight consignments. If the reverse is true
;;* then the flight is weight-restricted.

;;* If ( available density > 167)
;;* {
;;*  If  ( consignment density > available density )
;;*   then
;;*    density total = 1.5
;;*  else if ( consignment density <= 167 )
;;*        then density total = consignment density / 167
;;*       else
;;*        density total = 1 + 0.5 * (consignment density - 167) /
;;*                                  (available density - 167 )
;;* }
;;* else
;;*  if ( consignment density >= 167 )
;;*   then density total = 1.0
;;*  else
;;*   density total = maximum(0.5, 1+ (consgmt density - 167 ) /
;;*                                  (available density - 167))

(defrule DensityTotal1
        ?dens  <- (Action Calculate Density Total)
        (Available Density ?availden)
        (Consignment Density ?conden)
        (test (> ?availden 167) )
        (test (> ?conden ?availden) )
        =>
        (retract ?dens)
        (assert (Density Total 1.5)) )

(defrule DensityTotal2
        ?dens  <- (Action Calculate Density Total)
        (Available Density ?availden)
        (Consignment Density ?conden)
        (test (neq ?availden 0))
        (test (neq ?conden 0))
        (test (> ?availden 167) )
        (test (<= ?conden ?availden) )
        (test (<=  ?conden 167))
        =>
        (retract ?dens)
        (assert (Density Total =(/ ?conden 167) ) )  )

(defrule DensityTotal3
        ?dens  <- (Action Calculate Density Total)
        (Available Density ?availden)
        (Consignment Density ?conden)
        (test (neq ?availden 0))
        (test (neq ?conden 0))
        (test (> ?availden 167) )
        (test (<= ?conden ?availden) )
        (test (>  ?conden 167))
        =>
        (retract ?dens)
        (bind ?max (* 0.5 (/ (- ?conden 167) (- ?availden 167)) ) )
        (assert (Density Total =(+ 1 ?max) ) ) )

(defrule DensityTotal4
        ?dens  <- (Action Calculate Density Total)
        (Available Density ?availden)
        (Consignment Density ?conden)
        (test (<= ?availden 167) )
        (test (>= ?conden 167) )
        =>
        (retract ?dens)
        (assert (Density Total 1.0)) )
```

```
(defrule DensityTotal5
        ?dens  <- (Action Calculate Density Total)
        (Available Density ?availden)
        (Consignment Density ?conden)
        (test (neq ?availden 0))
        (test (neq ?conden 0))
        (test (<= ?availden 167) )
        (test (< ?conden 167) )
        =>
        (retract ?dens)
        (bind ?max (+ 1 (/ (- ?conden 167) (- ?availden 167)) ) )
        (assert (Find Max ?max C.5)) )

(defrule Maximum-Value                          ;;** Work around extanded Maths Set
        ?fmax <- (Find Max ?num1 ?num2)
        =>
        (retract ?fmax)
        (if (> ?num1 ?num2)
        then
           (assert (Density Total ?num1))
        else
           (assert (Density Total ?num2)) ) )

;;*************************************************************************
;; Rate Total
;; ** NOTE : Thru requests are NOT considered for Rate Totalling
;;*************************************************************************

;;* Calculate diff = (Consignment rate - Standard Rate )
;;* diff = ( > 50p )        Rate Total = 1.6
;;* diff = (30p to 49p )  Rate Total = 1.4
;;* diff = (20p to 29p )  Rate Total = 1.2
;;* diff = (10p to 19p )  Rate Total = 1.1
;;* diff = (-9p to 9p )    Rate Total = 1.0
;;* diff = (-19p to -10p )  Rate Total = 0.9
;;* diff = (-29p to -20p )  Rate Total = 0.8
;;* diff = (-49p to -30p )  Rate Total = 0.6
;;* diff = ( < -50p )  Rate Total = 0.4

(defrule CalcDiff
        ?ratetot  <- (Action Calculate Rate Total)
        (Consignment Rate ?conrate)
        (Standard Rate ?standrate)
        =>                                                  ;; Do not retract ?ratetot
        (assert (Rate Difference =(- ?conrate ?standrate))) )

(defrule RateTotal
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff 50) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 1.6) ) )

(defrule RateTotal1
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff 50.00) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 1.6) ) )

(defrule RateTotal2
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff 30) )
        (test (<= ?diff 49) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 1.4) ) )

(defrule RateTotal3
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff 20) )
        (test (<= ?diff 29) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 1.2) ) )
```

```
(defrule RateTotal4
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff 10) )
        (test (<= ?diff 19) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 1.1) ) )

(defrule RateTotal5
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff -9) )
        (test (<= ?diff  9) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 1.0) ) )

(defrule RateTotal6
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff -19) )
        (test (<= ?diff -10) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 0.9) ) )

(defrule RateTotal7
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff -29) )
        (test (<= ?diff -20) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 0.8) ) )

(defrule RateTotal8
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (>= ?diff -49) )
        (test (<= ?diff -30) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 0.6) ) )

(defrule RateTotal9
        ?Diff <- (Rate Difference ?diff)
        ?ratetot  <- (Action Calculate Rate Total)
        (test (<= ?diff -50.00) )
        =>
        (retract ?Diff ?ratetot)
        (assert (Rate Total 0.4) ) )

;;*****************************************************************************
;;** Urgency Total
;;*****************************************************************************
;;* If consignment is urgent
;;*      urgency total = 1.2
;;* else
;;*      urgency total = 1.0

(defrule UrgencyTotal
        ?urg <- (Action Calculate Urgency Total)
        (Favoured Station $?stations)
        (Requesting Station ?statn)
        =>
        (retract ?urg)
          (if (member ?statn $?stations) then
              (assert (Urgency Total 1.2) )
          else
               (assert (Urgency Total 1.0)) ) )
```

```
;;*************************************************************************
;;* Printing Utility Rules
;;*************************************************************************

(defrule PrintBanner
       ?prin <- (Action Print Banner)
       =>
       (retract ?prin)
      (printout t t t t t t t t t t)
      (printout t "*********************************************************"crlf)
      (printout t "**     B R I T I S H   A I R W A Y S   C A R G O     **"crlf)
      (printout t "**       C O N S I G N M E N T   S Y S T E M         **"crlf)
      (printout t "*********************************************************"crlf))

(defrule PrintResults1
       ?prin<- (Action Print Results)
       ?cor <- (Correct Matches ?correct)
       ?cont<- (Request Count ?count)
       (Recommendation ?rec)
       (HUMAN Decision ?human)
       (test (eq ?rec ACCEPT))
       (test (eq ?human A))
       =>
       (retract ?prin ?cor ?cont)
       (bind ?ncor (+ ?correct 1))
       (bind ?ncount (+ ?count 1))
       (bind ?percor (* (/ ?ncor ?ncount) 100))
       (assert (Correct Matches ?ncor))
       (assert (Request Count ?ncount))
       (format t "[ System Recommendation is %s the request ] [ Actual Decision
                  was %s ]%n" ?rec ?human)
       (format t "[ %d correct out of %d ==> %4.2f%% ]%n" ?ncor ?ncount ?percor)
       )

(defrule PrintResults2
       ?prin<- (Action Print Results)
       ?cor <- (Correct Matches ?correct)
       ?cont<- (Request Count ?count)
       (Recommendation ?rec)
       (HUMAN Decision ?human)
       (test (eq ?rec REJECT))
       (test (eq ?human R))
       =>
       (retract ?prin ?cor ?cont)
       (bind ?ncor (+ ?correct 1))
       (bind ?ncount (+ ?count 1))
       (bind ?percor (* (/ ?ncor ?ncount) 100))
       (assert (Correct Matches ?ncor))
       (assert (Request Count ?ncount))
       (format t "[ System Recommendation is %s the request ] [ Actual Decision
                  was %s ]%n" ?rec ?human)
       (format t "[ %d correct out of %d ==> %4.2f%% ]%n" ?ncor ?ncount ?percor)
       )

(defrule PrintResults3
       ?prin<- (Action Print Results)
       (Correct Matches ?correct)
       ?cont<- (Request Count ?count)
       (Recommendation ?rec)
       (HUMAN Decision ?human)
       (test (eq ?rec REJECT))
       (test (eq ?human A))
       =>
       (retract ?prin ?cont)
       (bind ?ncount (+ ?count 1))
       (bind ?percor (* (/ ?correct ?ncount) 100))
       (assert (Request Count ?ncount))
       (format t "[ System Recommendation is %s the request ] [ Actual Decision
                  was %s ]%n" ?rec ?human)
       (format t "[ %d correct out of %d ==> %4.2f%% ]%n" ?correct ?ncount ?percor)
       )

(defrule PrintResults4
       ?prin<- (Action Print Results)
       (Correct Matches ?correct)
       ?cont<- (Request Count ?count)
       (Recommendation ?rec)
       (HUMAN Decision ?human)
       (test (eq ?rec ACCEPT))
       (test (eq ?human R))
```

```
        =>
        (retract ?prin ?cont)
        (bind ?ncount (+ ?count 1))
        (bind ?percor (* (/ ?correct ?ncount) 100))
        (assert (Request Count ?ncount))
        (format t "[ System Recommendation is %s the request ] [ Actual Decision was %s
]%n" ?rec ?human)
        (format t "[ %d correct out of %d ==> %4.2f%% ]%n" ?correct ?ncount ?percor)
        )

(defrule PrintResults5
        ?prin <- (Action Print Results)
        (Recommendation ?rec)
        =>
        (retract ?prin)
        (format t "[ System Recommendation is %s the request ]%n" ?rec)
        )
```