EVOLUTIONARY ENVIRONMENTAL MODELLING IN SELF-MANAGING SOFTWARE SYSTEMS

HENRY LEE FORSYTH MPhil., BSc (Hons)

A thesis submitted in partial fulfilment of the Requirements of Liverpool John Moores University For the degree of Doctor of Philosophy

December 2010

Abstract

Over recent years, the increasing richness and sophistication of modern software systems has challenged conventional design-time software modelling analysis and has led to a number of studies exploring non-conventional approaches particularly those inspired by nature. The natural world routinely produces organisms that can not only survive but also flourish in changing environments as a consequence of their ability to adapt and therefore improve their fitness in relation to the external environments in which they exist.

Following this biologically inspired systems' design approach, this study aims to test the hypothesis – can evolutionary techniques for runtime modelling of a given system's environment be more effective than traditional approaches, which are increasingly difficult to specify and model at design-time?

This work specifically focuses on investigating the requirements for software environment modelling at runtime via a proposed systemic integration of Learning Classifier Systems and Genetic Algorithms with the well-known managerial cybernetics Viable Systems Model.

The main novel contribution of this thesis is that it provides an evaluation of an approach by which software can create and crucially, maintain a current model of the environment, allowing the system to react more effectively to changes in that environment, thereby improving robustness and performance of the system. Detailed novel contributions include an evaluation of a variety of environmental modelling approaches to improving system robustness, the use of Learning Classifier Systems and genetic algorithms to provide the modelling element required of effective adaptive software systems. It also provides a conceptual framework of an Environmental Modelling, Monitoring and Adaptive system (EMMA) to manage the various elements required to achieve an effective environmental control system.

The key result of this research has been to demonstrate the value of the guiding principles provided by the field of cybernetics and the potential of Beer's

cybernetically based Viable System Model in providing a learning framework, and subsequently a roadmap, to developing self-managing autonomic systems.

The work is presented using a virtual world platform called "Second Life". This platform was used for experimental design and testing of results.

ACKNOWLEDGEMENTS

The author would like to thank the School of Computing & Mathematical Sciences at Liverpool John Moores University for the support received during the production of the thesis.

I would especially like to thank my supervisory team, Professor A. Taleb Bendiab and Andy Laws for their patience and support during my studies.

Finally, I would like to thank my family for their understanding in terms of the scarcity of time available for them. I'll try my best to make up for it in the future.

Contents

Table of Figures

"Every Good Regulator Of A System Must Be A Model Of That System"

(Conant & Ashby, 1970)

Chapter 1: Introduction

The growing complexity of software systems and their interaction with their external environment continues to increase and requires software to adapt to its operating environment in order to operate robustly and dependably. This adaptation is currently largely achieved by human interaction such as traditional software maintenance.

A potential solution to this problem is to design systems that can reduce this requirement for human interaction and perform some of the re-configuration functions currently left to software development and maintenance teams. The requirement for self-adaptive software is made more urgent by the fact that software development is becoming more complex and unpredictable. External pressures such as increasing competition and the requirement for 24/7 system availability has meant that traditional software development techniques are likely to find it difficult to satisfy the requirement for quality software.

Autonomic computing seeks to design, amongst other characteristics, environmentally aware software systems, which is the main research topic of this thesis, but some of the challenges remain largely unmet. This chapter is organised as follows. Firstly, the topic of the thesis is presented with its aims. Secondly, the novel contribution of the new approach posited within the thesis is presented. Thirdly, an overview of the chapters of the thesis is presented. Finally, the chapter is summarised.

1.1 Motivation

As software systems grow in complexity, it becomes unfeasible for humans to monitor, manage, and maintain every detail of their operations (Dixon, Pham, & Khosla, 2001).

The software engineering industry already is currently suffering from a burgeoning software maintenance backlog due to increasingly frequent requirements change in the "real world". Lehman had identified this issue with his "Laws of Software

Evolution" and subsequent classification of e-type systems which are embedded in the real world and thus become part of it, thereby changing it (Lehman & Weir, 1980). A key requirement was that

> *"it must continually evolve to satisfy the conditions, needs and operation requirements of a changing environment at each moment in time"*
>
> (Lehman, 1998)

Further work performed by Lehman *et al.* suggested that virtually all systems require continuous change, increased in complexity and experienced continuing growth (Lehman, Ramil, Wernick, Perry, & Turski, 1997). It has been suggested that the likely result of this problem will be that software development teams will increasingly become maintenance teams therefore reducing their capacity to develop new applications.

Laddaga (Laddaga, 1999) has identified further reasons why application complexity is increasing:-

- Simple growth in problem size, as a result of success at previous problem sizes and increased hardware capacity
- The fact that many applications are now more closely tethered to the real world, actively utilising sensors and actuators, or being required to respond in real time.
- The enmeshed nature of today's applications – such as enterprise requirements to link all the elements of business process where it is useful to do so.

These new computational environments pose many challenges including the

> *"coherent coordination of vast numbers of elements, where the individual will have limited resources and reliability and the interconnects between elements will be irregular, local and possibly time-varying. In addition the physical embodiment of these elements is strongly intertwined with the individual behaviour of the elements and the global goal"*
>
> (Nagpal, 2004).

Arguably, it is easier to make a program that recovers from errors than it is to make a program that goes to great lengths to avoid making any errors. The complexity of the real world means that using traditional software development techniques results in increasingly unwieldy, brittle software if the current method of trying to rigorously define requirements from the outset is followed.

> *"The goal of self adaptive software is the creation of technology to enable programs to understand, monitor and modify themselves"*
>
> (Laddaga, 1999).

Bose (Bose & Matthews, 2000) identified four main requirements that a self-adaptive software system must meet:

- Detecting a change in context or a change in needs
- Knowing the space of adaptations
- Reasoning for adaptation decision
- Integrating the change

According to Laddaga (Laddaga & Robinson, 2000) therefore self-adaptive software understands

- What it does?
- How it does it?
- How to evaluate its own performance?
- How to respond to changing conditions?

Hence, self-adaptive software must contain a set of performance goals, which will ultimately drive the program to constantly evaluate and improve itself to satisfy the goals. It must therefore be able to evaluate it own performance, which in turn requires the system to hold a model of its own composition. The software must take data from its operating environment, sensors, internal libraries etc and use this information in a structured way. Should a self-adaptive program merely monitor its performance and respond to tolerance levels or should it proactively seek to improve performance continuously? There is an overhead associated with constant evaluation, which may make over-evaluation wasteful of valuable resources.

This evaluation may well lead the software to utilise alternative algorithms in order to improve performance. A library of alternatives will be available for the software to choose from in order to successfully adapt, although this in itself limits the self-adaptation of the system to those, presumably human produced alternatives. The complexity lies in making sense of the various data to narrow down the choice of alternatives that the program has to try. This is likely to involve the software being able to use diagnosis, decision theory and deliberation in order to make the best choice or at least reduce the potential choices to only the most promising.

This is now entering the field of AI as we seek to give the software an increasing ability to learn and reason. This factor is a key component in enabling stable software, which needs to respond to the need for change within acceptable tolerance levels.

AI researchers have previously considered many pertinent questions such as:-

- What adaptations are possible for systems that learn from experience?
- How can systems change in response to new information?
- What kind of training should a learning program receive?

(Dean, Allen, & Alimonon, 1995)

The software would have to respond to changing conditions in order to maintain its robustness in unknown situations and continue to function effectively. Meng (Meng, 2000) suggests that in order to maintain robustness self-adaptive software should ideally consist of both feed-forward and feedback control models. He suggests that the feed-forward (deliberative) component prescribes the predicted behaviour of the system that guarantees consistency over a longer period while the feedback (reactive) component adapts to local changes in its environment.

It has been identified that there are different levels of software adaptation that range from relatively simple to complex. The First International Workshop on Self-Adaptive software identified the high-level components required for self-adaptation to take place. These included, amongst others, an environmental model, learning, self-monitoring and a library of alternatives (Laddaga, Robinson, & Shrobe, 1999).

Robustness of self-adaptive software is a key requirement, which is difficult to achieve when the software may be employed in complex or not well understood environments. The measurement of robustness will be how well the software operates in these very environments.

The challenge is to write software that is initially less complex than the environment in which it is operating yet has the capacity to learn from and evolve to become at least notionally equal to the complexity of the environment. The software will be required to have the ability to evaluate its situation and use the resources available to overcome any problems.

Some of the components required to achieve this are likely to include monitoring, diagnosis, evaluation, testing and repair. Laddaga has identified evaluation as the hardest and most important problem for self-adaptive software (Laddaga & Robinson, 2000). It may be that software has to operate robustly with an analytic problem without any historical environmental models available.

Laddaga suggests that is there is still much work to be done is terms of:-

- What classes of application require what forms of evaluation?
- Which tools will provide better evaluation capability?
- To what extent such tools will need to be specific to particular application domains

A further challenge is to develop an architecture, which will simplify the design of self-adaptive system software. Robinson (Robinson, 2000) suggests that by keeping the program code separate from the diagnosis and synthesis code the core code can remain simple and the synthesis code can be performed by a purpose built synthesis engine. He suggests that such an architecture makes writing the code "much less of a tangled web". However, at the moment the specifications for adaptive software architecture models are not forthcoming. It is vital that an effective reference framework model is produced to enable this software to begin to be more effectively produced.

Kephart acknowledged that “virtually every aspect of autonomic computing offers significant engineering challenges” including monitoring and problem determination which will provide the basis for adaptation, self-optimisation and re-configuration (Kephart & Chess, 2003).

1.2 Thesis aims

The overall aim of the thesis is to study requirements for a machine learning mechanism specifically in relation to a systems relationship with its environment. In terms of environment a system can be viewed as either an “open” or “closed system”. In an open system all interactions with the environment are considered whereas in a closed system no interactions with an environment are considered.

The difficulty of producing an effective closed system is that it is often increasingly difficult to specify the requirements for such a system unless it is an extremely well defined problem. An effective open system presents a different set of challenges in that the environment of such a system could effectively include everything that is not part of the system. Specifying the environment of such an open system and the effect of environmental change could therefore present a significant challenge when considering modelling such a system.

The behaviour of systems is not determined exclusively by the internal properties of the system but also by external factors occurring in the “real world”. Furthermore, the behaviour of the system may also influence the external environment of the system. This is increasingly the case when systems are tethered to “real world” applications where there is considerable complexity in the environment. Previously, programme designers attempted to predict every circumstance that the program would encounter and provide a means for the program to respond to that. The difficulty is that is it virtually impossible to accurately predict every possible circumstance that may occur and hence the need for software maintenance.

There is debate over whether a model-based approach is required in to allow systems to react to the environment. For a system to survive and fully exploit opportunities in such an environment it seems reasonable to suggest that the system needs to hold a "view" or "model" of the environment. Laddaga and Robinson (Laddaga & Robinson, 2000), state that self adaptive systems will use this model to:-

- diagnose program failures and performance problems
- provide contextual basis for sub-goal and reconfiguration
- provide a basis for choosing new strategies for the computation

Whilst, without this model it may be possible to respond to environmental change, the basis for this change would seem to be less convincing. It may well be that a hybrid approach is taken where certain elements of the environment need to be modelled whereas others do not. In Lehmann's e-type systems, for example, the size and complexity of the real-world situation makes it impossible to model these systems completely.

This also raises the question of "model completeness". A complete model of the environment is only really achievable in a closed context. An open system suggests that "completeness" cannot be achieved and therefore any model must be adaptive. The difficulty in obtaining a complete model of the environment means that we must consider the level of model completeness required to allow our systems enough environmental information on which to choose their strategies.

The modelling of this "real world" environment is a tremendous research challenge but is vital to the development of adaptive software. Desirable properties of adaptive software related to the environment would be the ability to:-

- React to threats / opportunities presented by environmental change.
- Ideally, anticipating environmental trends to allow pre-adaptation to take place.
- Allow historical modelling in terms of "learning" how the actions of our system influence the environment.

The ability to react to threats/opportunities is the first step to allowing adaptive software to survive in potentially hostile environments, but would only allow for a "quick and dirty" adaptation to take place with systems constantly adapting in real time to changes in the environment. Perhaps this is enough for many systems that require timely "good" solutions to environmental instability rather than searching for the "perfect" solution. Future enhancements would allow systems to perform adaptation in a more controlled and planned way with increasingly optimum solutions selected, which prediction and learning will facilitate.

Therefore, this work aims to study the generic requirements for a machine learning mechanism to support runtime monitoring of a given adaptive software system's environment.

The specific objectives of this thesis include:

- To test the applicability of cybernetic thinking, and specifically the VSM (Viable System Model), to the development of autonomic computing systems.
- To test the applicability of Learning Classifier Systems and Genetic Algorithms, in providing the learning required to develop an environmental model.
- To evaluate an environmental modelling prototype by designing a set of comparative experiments used in a controllable system and environment and test the issues relating to the management of dynamically changing environmental sensitivity.
- To present the design of a prototype control system (EMMA) Environmental Modelling, Monitoring and Adaptive system that manages the major environmental modelling and control functions required by software systems.
- To evaluate and generalise the results with a view to understanding how runtime generated environmental models can be used as a service to support self-adaptive software systems design.

1.3 Novel contribution of the thesis

The novel contributions of this thesis include:-

The use of learning approaches to provide for the development of a *"current"* external model of the environment.

For a system to survive in a changing, and potentially changing environment, it seems reasonable to suggest that the system to hold a "view" or "model" of the environment. However, to provide a "complete" model of the environment is becoming increasingly difficult at the design stage due to increasing environmental complexity. Whilst, without this model it may be possible to respond to environmental change, the basis for this change would seem to be less convincing. The use of online-learning may provide a potential solution in providing an evolutionary view of the environment in which the system is operating.

Colombetti et al. states that

> "an optimal coupling of an agent with its environment is only possible if the dynamical structure of the environment is somehow mirrored by the structure of the agent's representation of its own behaviour policy"
>
> (Colombetti & Dorigo, 2000).

Learning has been used to provide a model of the environment and allows the system to update this model over time to maintain model currency. By maintaining "model currency", the system can use this model as a basis for future adaptive choices. The concept of an Environmental Modelling, Monitoring and Adaptive system (EMMA) to manage the various elements required to achieve effective environmental monitoring provides a novel contribution of the thesis.

Applies cybernetic thinking to the research problem of developing adaptive environmental models.

Cybernetic thinking (and particularly managerial cybernetics) is used as a fundamental underpinning to the development of an approach to generate and autonomically maintain not only a current model of the environment but also an internal model of system capability. This thesis seeks to emphasise the applicability of a cybernetic approach to systems development and specifically in providing the fundamental building blocks required in developing environmentally aware systems able to monitor their external environment, but also provide appropriate responses to environmental conditions.

Provides experimental data regarding the use of Learning Classifier Systems (LCS) in a novel application area.

The thesis seeks to produce experimental data and evaluate the suitability of LCS as an approach in an online environmental simulator. This online learning aspect is of considerable interest in terms of assessing the suitability of the LCS approach in new and novel environments.

This view is endorsed by Bacardit et al.

> *"of common interest are issues such as applying learners beyond the traditional classification problems and extracting information from real-world datasets"*
> (Bacardit, Mansilla, & Butz, 2008).

Bull also suggests that

> *"future work must apply LCS to a wide range of problems and identify characteristics which make the task suitable to solution with Learning Classifier Systems"*
> (Bull, 2004)

Therefore this research will add to the body of knowledge in terms of the application and suitability of LCS in this particular application domain but perhaps more generally in online applications.

Provides novel design aspects in the development of Learning Classifier Systems. Within the development of the prototype LCS there have been specific novel implementation aspects developed as a consequence of the requirements of the development of the EMMA system.

Demonstrates the potential applicability of virtual world experimental platforms for research purposes.

The development of the prototype system has allowed increasingly complex and turbulent environments to be controlled and generated to identify problems posed by increasingly challenging environmental conditions for online systems. The simulation has provided many of the building blocks required such as detectors, effectors, communication channels etc. This thesis demonstrates the usefulness and potential applicability of these platforms for future research projects and challenges.

1.4 Outline of the rest of the thesis

The remainder as the thesis is organised as follows:-

In Chapter Two, *"The Grand Challenge, Autonomic Computing & Self-Adaptive Software Systems"*, IBM's "Grand Vision" of autonomic computing, is outlined and the major elements discussed within this chapter. A "biological approach" to designing systems is analysed and seeks to discover what lessons can be learned from successful systems in nature such as the human autonomic nervous systems, ant colonies and the Darwinian approach of natural selection. These lessons can potentially be applied when seeking to producing robust adaptive systems which can effectively survive and indeed flourish in changeable and unknown environments that would previously have "broken" systems designed using traditional software development techniques.

In Chapter Three, *"A Cybernetic View of the Environment and its Models"*, we discuss cybernetic theory and its relationship with autonomic systems and the environment. We concentrate on the potential use of cybernetic theory as a roadmap to designing software that can adapt and respond to its environment. Beer's Viable System Model (VSM) is discussed, with a specific focus on System 4 of this model which is specifically concerned with the requirement to hold a model of both the external environment of the system and an internal model of system capability. This chapter also seeks to set out a conceptual definition of the environment.

In Chapter Four, *"An Approach to Provide an Adaptive Model of the Environment"*, a potential solution to the problem of providing current models of the environment which can evolve and adapt during run-time is presented. Learning Classifier Systems / Genetic Algorithms are discussed and presented as a possible approach to constitute the learning component of our Environmental Monitoring Modelling Adaptive system (EMMA).

In Chapter Five, *"Design & Evaluation of Initial Experimental Approaches to Controlling and Modelling the Environment"*, we present and evaluate an initial set of experiments designed specifically to analyse the main issues involved with controlling an environment in a virtual world simulation experimental platform. A variety of experiments are presented including a static model control system and a "hybrid" learning approach. The results of these initial experiments are presented and form a baseline performance metric for the prototype systems results analysed later in the thesis.

In Chapter Six, *"EMMA Testbed Design to Provide a Model of the External Environment and Internal Capability"*, a detailed discussion of the proposed Environmental Modelling, Monitoring and Adaptive system (EMMA) is presented. This system outlines the motivation for the major design decisions taken during this development.

In Chapter Seven, *"Adaptive Environmental Modelling Prototype System Results"*, the results from a sample of experimental results obtained by the EMMA prototype system are presented. We analyse and evaluate the prototype in terms of its ability to develop and evolve a current model of the external environment, a model of the internal capability of the system and also its ability to maintain performance levels, with respect to a system goal. The issue of scalability of the prototype system is discussed and experimental results demonstrating the issues and results of the scalability experiments are presented and analysed.

In Chapter Eight, *"Results and Evaluation"*, an evaluation of the overall project work and the experimental work of the Environmental Modelling, Monitoring and Adaptive system (EMMA) prototype development is presented. The evaluation discusses an overall evaluation of the project with respect to the original aims outlined in Chapter 1. It also evaluates the experimental data in terms of the difficulties presented by the use of LCS in producing environmental models. The chapter concludes that this approach could be applied to other software systems, and therefore more widespread applicability, although there are a number of challenges remaining.

In Chapter Nine, *"Conclusions and Future Research"*, a summary of the thesis, the contributions of the work as presented and the opportunities for further research in this area are presented. We conclude that whilst the work has concentrated on one particular aspect of the J-Reference model (System 4) as its environmental focus was of most interest to the project undertaken. The result of the research undertaken may allow further development to be undertaken to incorporate other aspects of the Viable System Model in order to attain the full benefit of a cybernetic approach to developing adaptive software to be realised.

"Controlling Complexity is the Essence of Computer Programming"

(Kernighan & Plauger, 1976)

Chapter 2: "The Grand Challenge", Autonomic Computing & Self-Adaptive Software Systems

2.1 "The Grand Challenge"

Paul Horn (IBM, 2001) presented both a vision and a "Grand Challenge" to the computing industry during a conference held in October 2001. This vision was to solve the growing problems related to software complexity that was, it was suggested, the single most important challenge facing the IT industry. This was to be addressed by devolving some responsibility for software maintenance to the software itself. The analogy of human autonomic systems was adopted as a guiding principle, allowing computer systems to be built:

> *"that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies"*
> (Horn, 2001).

According to Kephart, a phrase with biological connotations was deliberately chosen, in that the

> *"autonomic nervous system governs our heart rate, body temperature etc., thus freeing our conscious brain from the burden of dealing with these and many other low level, yet vital, functions"*
> (Kephart & Chess, 2003).

This self management of systems is seen as of paramount importance because human based management is often seen as the "largest factor in the Total Cost of Ownership (TCO) of such systems" (Zhang, Lin, Lian, & Jin, 2004). As systems increase in complexity and must increasingly respond to external events in real-time, the difficulty of "tuning" to achieve desired quality of service requirements is likely to become increasingly challenging using manual methods (Kandasamy, Abdelwahed, & Hayes, 2004). It was recognised that a solution to this "challenge" would require a long term effort by researchers in a diverse range of fields.

Traditional approaches to adaptation were seen as *"unlikely to provide the required sophistication of behaviour"* and that any approach must offer the ability to

> *"abstract and isolate high level goals from low level actions, to integrate and act on imperfect and conflicting information, and to learn from past actions to improve future performance"*
>
> (Clark, Partridge, Ramming, & Wroclawski, 2003).

IBM identified a range of benefits to be realised by the use of autonomic computing principles. These ranged from *"short term benefits"* such as a simplified user experience through a more responsive, real time system and increased stability and availability of systems, through to *"long term, higher order benefits"* such as constructing autonomic federated systems and collaborative systems.

IBM went on to suggest a wish-list of eight characteristics of autonomic computing which consisted of the following:-

- An autonomic computing system needs to *"know itself"* - Since a "system" can exist at many levels, an autonomic system will need detailed knowledge of its components, current status, ultimate capacity and all connections to other systems, to govern itself. It will need to know the extent of its "owned" resources, those it can borrow or lend, and those that can be shared or should be isolated.
- An autonomic computing system must configure and reconfigure itself under varying and, in the future, even unpredictable conditions. System configuration or "setup" must occur automatically, as well as dynamic adjustments to that configuration to best handle changing environments.
- An autonomic computing system never settles for the status quo - it always looks for ways to optimize its workings. It will monitor its constituent parts and fine-tune workflow to achieve predetermined system goals.
- An autonomic computing system must perform something akin to healing - it must be able to recover from routine and extraordinary events that might cause some of its parts to malfunction. It must be able to discover problems

or potential problems, then find an alternate way of using resources or reconfiguring the system to keep functioning smoothly.

- A virtual world is no less dangerous than the physical one, so an autonomic computing system must be an expert in self-protection. It must detect, identify and protect itself against various types of attacks to maintain overall system security and integrity.
- An autonomic computing system must know its environment and the context surrounding its activity, and act accordingly. It will find and generate rules for how best to interact with neighbouring systems. It will tap available resources, even negotiate the use by other systems of its underutilized elements, changing both itself and its environment in the process - in a word, adapting.
- An autonomic computing system cannot exist in a hermetic environment. While independent in its ability to manage itself, it must function in a heterogeneous world and implement open standards - in other words, an autonomic computing system cannot, by definition, be a proprietary solution.
- An autonomic computing system will anticipate the optimized resources needed while keeping its complexity hidden. It must marshal IT resources to shrink the gap between the business or personal goals of the user, and the IT implementation necessary to achieve those goals -- without involving the user in that implementation.

(IBM, 2008)

The essence of autonomic computing is self management as captured by Kephart, who suggested that:

"the intent is to free systems administrators from the details of systems operation and maintenance and to provide users with a machine that runs at peak performance 24/7"

(Kephart & Chess, 2003)

Kephart further suggested that self management consisted of four main aspects which were self-configuration, self-optimisation, self-healing and self-protection and that over time "humans will need to make relatively less frequent, predominantly higher-level decisions, which the system will carry out automatically via more numerous, lower-level decisions and actions" (Kephart & Chess, 2003). To achieve the effective self management suggested previously it seems likely to require runtime adaptation. Runtime adaptation can be explained as

> *"any automated set of actions aimed at modifying the structure, behaviour and/or performance of a target software system while it continues operating"*
> (Valetto, Kaiser, & Phung, 2005).

IBM's "Grand Vision" for autonomic computing also outlined a number of challenges including:-

- The computing paradigm will change from one based on computational power to one driven by data.
- The way we measure computing performance will change from processor speed to the immediacy of the response.
- Improving network-monitoring functions to protect security, detect potential threats and achieve a level of decision-making that allows for the redirection of key activities or data.
- Smarter microprocessors that can detect errors and anticipate failures.

Adaptive software techniques may have a contribution to make in some or all of these "challenging" areas.

IBM suggested a roadmap for the design of autonomic systems in their "Architectural Blueprint for Autonomic Computing" series of articles (IBM, 2006). The main purpose of the blueprint documents was to define concepts and constructs for building self-managing abilities into system software. It sought to define the architectural building blocks of these abilities and provide a model for their adoption. The next section of the report will seek to provide an "overview" of the blueprint document.

Figure 1 provides an example of an autonomic computing reference architecture with a number of building blocks including

- Knowledge sources
- Level 1 Managed Resources
- Level 2 Manageability Endpoints (called touchpoints in previous blueprint documentation)
- Levels 3 & 4 Autonomic Managers
- Level 5 Manual Managers

Figure 1 IBM's Autonomic Computing Reference Model

(IBM, 2006)

The lowest layer of this architecture consists of the managed resources, which could be a hardware or software component, but must be able to be managed and by this we mean sensed and effected. Level 2 seeks to incorporate a consistent standard manageability interface to implement sensor and effector behaviour for the managed resource.

The goal of Levels 3 and 4 is to automate some portion of the process using the autonomic manager building blocks. The IBM vision of the autonomic manager is to

DIAGRAM
ON THIS PAGE
EXCLUDED
UNDER
INSTRUCTION
FROM THE
UNIVERSITY

provide four broad categories of self-management, namely self-configuration, self-healing, self-optimising and self-protecting functionality. The main function of the autonomic manager is to implement an intelligent control loop and to achieve this there must be four distinct functions within this component, as in Figure 2 below.

Figure 2 IBM Four Part Control Loop

(IBM, 2006)

IBM (IBM, 2006) defines this four part control loop as consisting of the following:

- The monitor function provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.
- The analyze function provides the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow the autonomic manager to learn about the IT environment and help predict future situations.
- The plan function provides the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.
- The execute function provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

Knowledge sources will provide the autonomic manager with a variety of information such as symptoms, policies, requests for change and change plans. There will be data created by the autonomic managers that will also be stored for future use including historical logs, metrics and experimentation results. This will allow learned knowledge to be collected, allowing more appropriate and effective planned responses in future. An autonomic manager should be able to load knowledge from a variety of sources to perform additional tasks as appropriate.

IBM also define a Level Four orchestrating autonomic manager that *"orchestrate"* other autonomic managers therefore offering control loops that have the *"broadest view of the overall IT infrastructure".*

IBM recognised that IT professionals may only be willing to delegate portions of the control loop to the autonomic manager. Therefore, they recognise that there is likely to remain a human managerial role that enables an IT professional to perform some management functions manually. Placing this building block at the top of the pyramid recognises that while human intervention is likely, it will probably differ depending on the organisation, type of systems being automated etc. It may well be that the human interaction may be goal specification for the system to autonomically develop plans to satisfy these higher levels goals or perhaps the autonomic system will be tasked with monitoring and suggesting plans but it will be the human element that decides which plan to execute.

IBM's blueprint document recognised that incorporating self-managing capabilities into an IT organisation is an "evolutionary process" (IBM, 2006) and have developed a maturity model called the "autonomic computing adaptation model" to help businesses recognise where they are and what they need to do to increase autonomic potential. The model is shown below in Figure 3.

DIAGRAM
ON THIS PAGE
EXCLUDED
UNDER
INSTRUCTION
FROM THE
UNIVERSITY

provide four broad categories of self-management, namely self-configuration, self-healing, self-optimising and self-protecting functionality. The main function of the autonomic manager is to implement an intelligent control loop and to achieve this there must be four distinct functions within this component, as in Figure 2 below.

Figure 2 IBM Four Part Control Loop

(IBM, 2006)

IBM (IBM, 2006) defines this four part control loop as consisting of the following:

- The monitor function provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.
- The analyze function provides the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow the autonomic manager to learn about the IT environment and help predict future situations.
- The plan function provides the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.
- The execute function provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

- At the closed loop level, the IT environment can automatically take actions based on the available information and the knowledge about what is happening in the environment
- At the closed loop with business processes level, business policies and objectives govern the IT infrastructure operation. Users interact with the autonomic technology tools to monitor business processes, later the objectives or both (IBM, 2008).

The control scope dimension (y axis) is defined by IBM as

- At the subcomponent level, portions of resources are managed, such as an operating system on a server or certain applications within an application server.
- At the single instance level, an entire standalone resource is managed, such as a server or complete application server environment.
- At the multiple instances of the same type level, homogeneous resources are managed, typically as a collection, such as a server pool or cluster of application servers
- At the multiple instances of different types level, heterogeneous resources are managed as a subsystem, such as a collection of servers, storage units and routers or a collection of applications servers, databases and queues.
- At the business system level, a complete set of hardware and software resources that perform business processes is managed from the business process perspective, such as a customer relationship management system or an IT change management system (IBM, 2006).

Therefore IBM suggest that autonomic maturity can evolve in three dimensions

- Automating more functions as the maturity level increases
- Applying automated functions to broader resource scopes
- Automating a range of tasks and activities in various IT management processes (IBM, 2006).

2.2 Approaches to Developing Self-Adaptive Software Systems

Throughout the first decade of the 21st century there has been significant research effort and contributions to the many challenges presented by the development of self-adaptive and autonomic software systems.

Khalid *et al.* suggests that current projects mostly focus on one of two design approaches, Externalisation and Internalisation.

- In the externalisation approach, modules enabling self-management lie outside the managed system
- In the internalisation approach, application specific self-management is done inside the managed system.

They state that the externalisation approach is more effective, and more common, because it

"localises problem detection and resolution in separate modules. It provides a more generic solution that can be used to enable autonomic behaviour in existing systems"

(Khalid, Haye, Khan, & Shamail, 2009)

A middleware based approach to the problem have been presented by, amongst others, Ginis (Ginis & Strom, 2004) with their SMILE (Smart Middleware Light Ends) prototype in which they sought to develop methods of automated system management, continuous self optimisation and efficient incremental computation techniques. Chen presented a solution for processing data streams that required the middleware to evaluate the load at any given stage, and then decide how the value should be adjusted (Chen & Agrawal, 2004).

Kumar *et al.* presented their IFLOW autonomic middleware for implementing self-management when building large scale distributed systems. This approach sought to represent information flows using an information flow graph. They suggested that

"the advantage of reducing each of these models to a common abstraction is that one can embed autonomic features into the middleware implementing this abstraction"

(Kumar, et al., 2006)

The Java Messaging Service (JMS) was used by Taton *et al.* to provide load balancing in a clustered queue application (Taton, De Palma, Philippe, & Bouchenak, 2007). In 2008, Zhu recognised that a middleware must be able to provide a "timely response" to important events and based their middleware on the existing Grid infrastructure and Service Oriented Archicture (SOA) concepts to provide this "timely response" (Zhu & Agrawal, 2008).

Kesaniemia *et al.* stated that an issue with existing middleware platforms for multi-agent systems was that they do not provide general support for observation, where observation is considered to be an important mechanism needed for realising effective and efficient co-ordination of agents. They proposed a framework called Agent Observable Environment (AOE) sought to "integrate information from various sources into one shared, observable state of the world" (Kesaniemi, Katasonov, & Terziyan, 2009).

Evolution and emergent behaviour has been applied to the development of robust, scalable systems with self-healing properties by Anthony (Anthony R. J., 2004), Goldsby *et al.* (Goldsby, Cheng, McKinley, Knoester, & Ofria, 2008), Beckmann *et al.* (Beckmann, Grabowski, McKinley, & Ofria, 2008) and Bisadi (Bisadi & Sharifi, 2009).

Anthony suggested that emergent systems tended to achieve

- Scalability through low-complexity communication strategies and total decentralisation
- Robustness through the use of low-value messages and anonymous nodes
- Low-latency through one-way communications and autonomous decisions based solely on nodes' local views
- Stability through randomness and feedback
- Efficiency through using small numbers of short messages and limited use of state information, thus having low processing overhead and storage requirement.

(Anthony R. J., 2004)

Goldsby *et al.* used a digital evolution approach to control an intelligent robot (T-ROT) which provided care and support for the elderly. They recognised that concerns remained such as how the approach will scale when used with larger applications and also that performance was dependent upon the complexity of the task (Goldsby, Cheng, McKinley, Knoester, & Ofria, 2008). Beckmann *et al.* used "digital organisms", seeking inspiration from Darwinism, in conjunction with the AVIDA digital evolution system to provide an iRobot with evolving behaviours. (Beckmann, Grabowski, McKinley, & Ofria, 2008).

Bisadi proposed a self-healing mechanism based on cellular adaptation that considers components as "black boxes" whose code need not be manipulated. They suggested that this approach was beneficial in that it provided the opportunity

> *"to use commercial components whose codes are not accessible, as well as increasing the robustness of the system and easing the maintenance of the system"*
>
> (Bisadi & Sharifi, 2009)

Reinforcement Learning has been applied in a number of "autonomic" prototype system developments. Littman *et al.* used a learning algorithm for the purpose of autonomic network repair. They stated that by implementing the prototype on a live network had "helped illustrate the robustness of the basic idea", and also indicated that they should "revisit some of the underlying assumptions" used in the model (Littman, Ravi, Fenson, & Howard, 2004).

In 2004, Chang *et al.* also used reinforcement learning to provide mobile ad-hoc networks with improved packet routing decision, and node mobility, therefore improving the connectivity of the network. They adapted the Q-Learning algorithm to provide an adaptive packet routing algorithm. They found that results were inconclusive and that one problem was that

> *"the learning agent can not distinguish between states where high network performance is due to its own good actions, versus states where the high network performance is due only to actions taken by other agents in the network"*
>
> (Chang, Ho, & Kaelbling, 2004)

One of the challenges provided by online reinforcement learning is potentially poor performance whilst learning is taking place. *Tesauro et al.* proposed that a potential solution to this issue is a hybrid reinforcement learning approach that they applied in Data Centres. This consisted of "a model-based policy to immediately achieve a high (or at least decent) level of performance as soon as it was implemented within a system". They stated that by running such a policy to obtain training data for reinforcement learning, they maintained acceptable performance in the system at all times. If significant re-training was required they fell back on the model based policy to deliver an acceptable performance level whilst retraining took place. (Tesauro, Jong, Das, & Bennani, 2006)

Seshia used online learning as a basis for self-repair using a simple network monitor to repair faults. Seshia stressed that the proposed online learning strategy (termed CE3), is not limited to network repairs but could, for example, be used in auto-tuners to improve the performance of a system as it runs (Seshia, 2007).

The use of reinforcement learning to provide adaptive policy driven autonomic management has been an area of interest to researchers. Bahati suggests that a key question is whether "a model learned from the use of one set of policies could be applied to another set of similar policies, or whether a new model must be learned from scratch as a result of changes to an active set of policies". Bahati illustrated how a reinforcement learning model might be adapted to accommodate such changes (Bahati & Bauer, 2009).

Researchers have been seeking to provide an underpinning architectural framework for the development of autonomic systems. Some interesting approaches include the use of utility functions (Walsh, Tesauro, Kephart, & Das, 2004), architectural design patterns (White, Hanson, Whalley, Chess, & Kephart, 2004) and dynamic decision points (Anthony, Pelc, Ward, Hawthorne, & Pulnah, 2008).

Increasingly we are seeing "real-world" applications of autonomic systems in increasingly complex situations. Chess *et al.* documented their work with the Unity System which involved resource allocation between application environments (Chess, Segal, Whalley, & White, 2004).

In 2007, Kephart *et al.* demonstrated that it was possible to get two autonomic managers (one managing performance and one managing power) to work together in order to achieve specified power-performance tradeoffs (Kephart, et al., 2007).

Xu *et al.* used fuzzy logic as an approach to data centre resource management. They used fizzy models to "learn the relationship between the workload and resource needs of virtual containers and to guide resource allocation based on online measurements" (Xu, Zhao, Fortes, Carpenter, & Yousif, 2007).

Other recent adaptive applications include Peer to peer (P2P) information sharing (Yeung, Yang, & Ndzi, 2009), Ambient assisted living (AAL) (Segarra & Andre, 2009) and adaptive network services (Strassner, et al., 2008).

The requirement of autonomic systems to be able to manage legacy systems has been recognised as an issue which requires architectural development to manage the complexities of such systems. In 2006, Griffith's Kheiron framework attempted to solve this issue by providing an approach to "retrofitting new functionality onto these systems" for legacy systems written in languages such as C/C++ (Griffith & Kaiser, 2006). Calinescu proposed a model-driven framework which sought to develop autonomic systems out of non-autonomic components (Calinescu, 2007).

Whilst there have been many impressive advancements in autonomic and adaptive software development, they have tended to be applicable to specific problem domains. There is still a lack of large-scale applications and a generalised framework to provide a roadmap for realising the full vision of autonomic computing.

2.3 "The Biological Response"

It is interesting to observe the re-emergence of biological concepts in the light of advances in autonomic and distributed computing. Babaoglu *et al.* suggests that the reason for this is that

> *"most of the biological structures have a number of "nice properties", which include robustness to failure of individual components, adaptation to changing conditions, and the lack of reliance on explicit central co-ordination"*
>
> (Babaoglu, et al., 2006).

Beer *et al.* (Beer, Chiel, & Sterling, 1990) states that,

> *"all animal behavior is adaptive in that as an animal confronts its environment, its behavior is continuously adjusted to meet the ever-changing internal and external conditions of the interaction"*

and that this behavior is goal-oriented, adaptive and robust. This behavior would seem to have the attributes we would wish to instill in our software systems to improve robustness in the face of environmental challenges.

It would not be sufficient to merely produce systems that merely "react" to the environment but rather to develop systems that can search and learn from the environment to produce better responses to be learned and developed for future challenges. Burgess suggests that "biology is nothing more than a colossal search algorithm, seeking organisms that can fit into an environment and play some role in the ecological network" (Burgess, 2007). If this is indeed the case then these searching capabilities are a desirable property of robust software systems.

Van Valens "red queen effect" of evolutionary theory describes the process whereby a species facing increased competition from its rivals will attempt to evolve to improve performance and ensure survival. Successful evolution results in increased competition for its rivals prompting a reactive evolution to improve their own performance, which again leads to increased competition for the original species and the cycle continues (Van Valen, 1973). It does not require much of a stretch of the imagination to view computer systems as being subjected to their own form of ongoing, accelerating red queen effect, as changing environmental conditions and an intensifying pursuit of optimised performance demands the iterative emergence of ever sleeker, fitter systems to manage instability and turbulence. This fitter rule set would require the learning element identified earlier as a key component.

The analogy of designing software systems based on a biological regulation system was an interesting one as it brought with it connotations of homeostatic regulation, receptors, sympathetic and parasympathetic systems. This would suggest there may be value in revisiting the research findings in the area of systems cybernetics (such as the Viable System Model presented by Beer) as a potential framework for an approach to the adaptive software development process.

In 1878, Bernard used the phrase "milieu interior" (or the environment within) and defined the importance of internal stability as follows:

> *"The fixity of the milieu supposes a perfection of the organism such that the external variations are at each instant compensated for and equilibrated. All of the vital mechanisms, however varied they may be, have always one goal, to maintain the uniformity of the conditions of life in the internal environment. The stability of the internal environment is the condition for the free and independent life."*
>
> (Bernard, 1879)

Drawing on this theme, Cannon introduced the term homeostasis or "steady state" in 1932 to describe the coordination and co-operation between physiological processes in the human body involving the brain and nerves, the heart, lungs, kidneys and spleen etc to achieve a condition, which is relatively stable. He also suggested that

> *"the perfection of the process of holding a stable state in spite of extensive shifts of outer circumstances in not a special gift bestowed upon the highest of organisms but is the consequence of a gradual evolution"*
>
> (Cannon, 1932)

Therefore, to some extent, homeostasis could be considered as "learned" through the evolutionary process. The human nervous systems in its most simple terms monitors internal and external changes and invokes appropriate responses in order to maintain homeostasis and does so without conscious thought. The human nervous system is often identified by autonomic researchers as a good example of a successful sophisticated autonomic system.

A homeostatic system has been described by Parashar as

> *"a system that reacts to every change in the environment, or to every random disturbance, through a series of modifications that are equal in size and opposite in direction to those created the disturbance. The goal of these modifications is to maintain internal balances"*
>
> (Parashar & Hariri, 2005).

The ability of a software system to autonomically maintain homeostasis would enable these systems to become more robust and therefore another example of a desirable trait in effectively making them "autonomic". Kiciman (Kiciman & Wang, 2004) suggested an approach to building autonomic systems based on combining autonomous intelligent agents in a well structured way. They illustrated that this approach

> *"mirrors the structure of the human brain wherein there are clearly defined, function specific processing centres connected by forward and backward communication channels and adaptive feedback loops"*
>
> (Bigus, Schlonnagle, Pilgrim, Mills III, & Diao, 2002).

The notion of self-organisation demonstrated by insect colonies (such as ants) is often cited as an example of how emergent properties from interactions at an individual level can develop. The concept of stigmergy introduced by Grasse in 1959 showed how "simple systems can produce a wide range of more complex coordinated behaviours, simply by exploiting the influence of the environment" (Serugendo, et al., 2004). This is important as it demonstrates how we may exploit the environment to improve system robustness, stability and performance.

An approach suggested by Nagpal was that we should look at previous biological studies into the ability of cells in an embryo to locally coordinate to develop into complex organisms. The ability of this process to regulate in the face of failures and natural variation can be used as a basis for designing robust algorithms that can achieve similar goals as outlined as the vision of autonomic computing (Nagpal, 2004).

Ashby (Ashby, 1954) promoted the architecture of the "Ultra Stable System", which included two control loops. One of these loops controlled small disturbances and a second loop that is responsible for adaptation when existing "behaviour sets" are unable to maintain homeostasis. The classic example often used to describe this concept is that of an automatic pilot that keeps an aeroplane horizontal in response to turbulent air. If the ailerons were connected in reverse, the autopilot, in response to air turbulence requiring some degree of "roll" to maintain its position, would increase the roll and would persist in its wrong action to the very end. A secondary control loop would mean that the system would recognise this deviation from expected behaviour and goes through a process of adapting its behaviour until it finds a scheme that does work in these circumstances.

However, to build software system applications that can react to environmental changes it seems likely that *"sophisticated monitoring is needed to collect and provide information about the services and resources and the complete system itself"* (Trumler, Petzold, Bagci, & Ungerer, 2004). Monitoring, in itself, can be an "expensive" overhead and therefore should ideally be targeted and adaptive in itself to maximise benefits to a system. Nagpal illustrated this point with an example of agents who broadcast an "aliveness" message to its neighbouring agents. The frequency of aliveness messages from neighbours determines the speed that faults can be detected but that faster response time comes with a higher cost of communication (Nagpal, 2004). One of the key objectives of an environmental monitoring system should be to achieve "the right balance between global optimisation effort, local computational expense, negotiation time, and decision quality". We may not always require an optimum decision but rather a good but timely decision.

The major difficulty in any system that monitors for problems is "knowing what the monitor should be looking for". Kiciman et al suggests that "negative identification" may provide a solution. This concept relies on describing what a "good configuration" looks like and then when this constraint is broken, it can be assumed that there is a problem even though the problem may not have be seen before (Kiciman & Wang, 2004).

Beer when talking about the practice of planning suggested that

> *"The plan has to be adaptive. By adaptation in the planning mode we may talk about timely updating, about flexibility, about constant review and revisions as events unfold"*
>
> (Beer S. , 1991)

It is likely that the solution to realising autonomic computing will require a multi-disciplinary approach with many research communities making contributions and providing a major input to the long term successful implementation of autonomic self-adaptive systems.

2.4 So where are we now?

Nine years after IBM's "Grand Vision", it is useful to reflect on the progress of the computing research community in terms of delivering the vision outlined by Horn in 2001.

IBM has continued to actively promote applications of autonomic computing throughout the period. An example of this would be research by IBM to achieve autonomic management of power and performance in data centres, which primarily focuses on achieving power savings without affecting service level agreements (Das, Kephart, Lefurgy, Tesauro, Levine, & Chan, 2008).

Vassev *et al.* states that despite many such initiatives it is still not pervasive across the IT industry and that the *"only significant progress of autonomic computing has been the integration of self-managing autonomic features into individual products such as chips, databases and network components"*

(Vassev & Hinchey, 2009)

Cooter suggests that "in some respects, it looks like the concept [autonomic computing] is still in the theoretical phase" (Cooter, 2010). Menasce and Kephart (one of the authors of "The vision of autonomic computing") conceded four years later in 2007 that

"we've by no means reached our destination. Although many autonomic components have been developed and are proving useful in their own right, no-one has yet built a large scale, fully autonomic computing system"

(Menasce & Kephart, 2007)

Despite progress in a number of areas related to autonomic computing, progress has perhaps been slower than might have been expected. Dobson, writing in 2010, suggested that "efforts since 2001 to design self-managing system have yielded many impressive achievements, yet the original vision of autonomic computing remains unfulfilled" (Dobson, Sterritt, Nixon, & Hinchey, 2010). Dobson further suggests the term autonomic computing remains closely associated with the IBM initiative but should be broadened to related initiatives such as "organic computing, bio-inspired computing, self-organising systems, ultra-stable computing, autonomous and adaptive systems, to name but a few".

Huebscher *et al.* stated in 2008 that "though autonomic computing has become increasingly interesting and popular, it remains a relatively immature topic" but that once more disciplines and their established research becomes more involved, autonomic computing will be "naturally embedded in the design process where all system architectures will have reflective and adaptive elements" (Huebscher & McCann, August 2008).

2.4 Summary

The requirement for more robust software that can self-manage significant aspects of its operation including the ability to self-configure, self-heal, self-optimise and self-protect through having the requisite functionality to respond and adapt to changes in its operational environment is both seductive and compelling. The IBM "Grand Vision" in conjunction with the Self-Adaptive Software research community has provided significant advances in terms of visualising the likely requirements and outlined the building blocks of such systems. There are a growing number of examples of partial implementations appearing in the literature and continued development across a number of areas in the future can be expected.

One of the less travelled areas of research concerns the problem of developing an accurate and current model of the environment in which such adaptive systems will operate. It would seem a compelling argument that holding a current model of both the environment and also the current capability of the system allowing the system to "know itself" are desirable additions to any adaptive system. As such they have a view of the complex space within which they can adapt and that without these properties the system could be only considered to be purely reactive.

In conclusion, while the original IBM "Grand Vision" has not yet been realised, it was probably always going to be an evolutionary rather than revolutionary process. There are and still remain many research challenges that require a multi and cross-disciplinary approach to building software systems that can survive and flourish in increasingly changing and complex environmental conditions. Despite the reservations expressed regarding the progress of autonomic computing, the original drive behind the vision that *"in the future, more software will have to be adaptive, changing itself to cope with new requirements or unforeseen circumstances or to ensure resilience in harsh environments"* (Black, Boca, Bowen, Gorman, & Hinchey, 2009) adds weight to continued research in this area. Models of the external environment and internal capability and how they can be developed during run-time will increasingly become central to how systems can survive in these particular circumstances and perhaps the natural world provides both inspiration and a roadmap forwards.

"The Idea That Information Can Be Stored In A Changing World Without An Overwhelming Depreciation Of Its Value Is False"

Norbert Wiener

3 A Cybernetic View of the Environment and its Models

"Cybernetics: The science of control and communications in the animal and the machine"

(Wiener, 1954)

Many of the issues discussed by the autonomic self-adaptive software community in making complex systems that are both robust and able to adapt when faced by environmental change have long been a significant area of study in cybernetics, which in turn has long advocated the applicability of lessons from the natural world.

3.1 Cybernetics and the Environment

Modern cybernetics is viewed as a scientific discipline that began and gained momentum in the late 1940s with the Macy Foundation Meetings (1946-1953) and the publication in 1948 of the seminal book entitled "Cybernetics" by Norbert Wiener. Early pioneers in the field included Wiener, Ross Ashby, Warren McCulloch, Grey Walter, Gregory Bateson, Julian Bigelow and Margaret Mead. The extreme cross disciplinary nature of this group should be noted, as they brought together concepts in scientific disciplines as diverse as control theory, computing, mathematics, biology, neurophysiology, psychology and philosophy. Francois suggests that cybernetics as a discipline is still developing with it being "a meta-language of concepts and models for trans-disciplinary use, still now evolving and far from being stabilised. This is the result of a slow process of accretion through inclusion and interconnection of many notions, which came and are still coming from very different disciplines" (François, 1999).

The driver for the study of cybernetics lies in the increasing complexity of systems and a desire to regulate these systems but it could be argued that much of the potential benefits of cybernetics have remained unrealised by the scientific community. Powers suggested that cybernetics had gained only the smallest of beachheads and that very few people had "a real sense of the kind of impact on the scientific community that is the potential still sleeping in cybernetics" (Powers, 1984).

The term “cybernetics” is derived from the Greek word “kybernetes” and was used by Plato to describe a pilot or helmsman, who undertook the responsibility for steering (or controlling) a boat. The responsibility of the helmsman was to maintain the course of the boat in response to any deviation caused by weather and tidal patterns. In this sense the helmsman used feedback from the environment to both chart the position of the boat (perhaps using a lighthouse) and take real-time corrective action if required and therefore could be considered a negative feedback “control system”. In any control system, the notion of feedback is central to the successful regulation of the system. Feedback can be defined as the use of information resulting from the execution of a process, which is subsequently used to alter the process in order to maintain a goal.

Regulation can be defined as the ability of a system to compensate in response to an environmental input or output variable. In feedback regulation (or control as it is technically referred to), an error signal is produced between the existing state of the system and the desired regulation level. This error signal is acted upon operationally, amplified in power, and fed to an actuator to operate a network which can influence the regulated variable so as to reduce the error (Dictionary of Cybernetics and Systems, 2010).

There are numerous examples of control systems that use feedback from sensors to alter the process. An example could be the use of a Passive Infrared (PIR) sensor to control the switching on of lights when someone enters a room and switching them off when they leave. Another example is the often used example of a central heating system that uses feedback (obtained from the thermostat in a room) to control the switching on and off of the boiler to maintain room temperature at the required level. These are examples of systems that use information obtained from a sensor that allow the system to detect a change from the desired goal of the system, namely to maintain temperature levels at the desired level. When a change or deviation from the desired goal is detected, the system effects a change to correct the error. Therefore, these systems are self regulating through the use of feedback and subsequent corrective actions that they can effect. The timeliness of information on the system state and any corrective action is crucial to the appropriateness of the action taken by a system.

The study of the human body has played an important role in the development of cybernetic thinking as the human body is seen as one of the most successful examples of systems using self-regulation to promote robustness and survival. The human brain is often seen as the most successful regulator of all, regulating the human body and there are numerous examples of how this self-regulation is manifested within the "human system". The classic example is the maintenance of body temperature through the use of perspiration and shivering in an attempt to respond to feedback and control any the deviation of body temperature from its "normal" operating temperature but also could include the production of antibodies to neutralise foreign objects such as viruses to minimise the potential impact of illness and infection, the dilation of the pupils in response to varying lighting conditions etc.

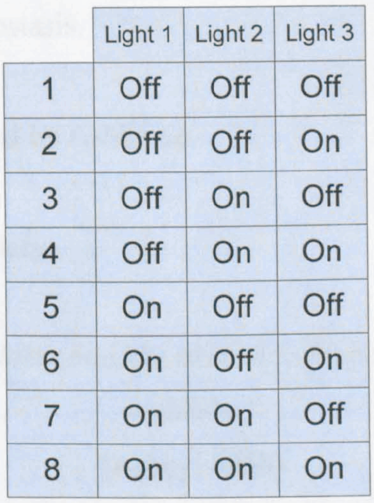
These responses can be seen as an attempt to maintain homeostasis or equilibrium of the system in response to changes its in operating environment. Ashby defined "essential variables" as those system variables which have to be kept with specific boundaries for the organism to survive (Ashby, 1956) and therefore the ability of a system to enact change to bring these variables back to a state of homeostasis is vital. Interestingly, as shall be seen later, these systems are classed as "autonomic" in the medical literature, where "autonomic" is defined as "the part of the nervous system responsible for control of bodily functions not consciously directed" (Oxford English Reference Dictionary, 2002)

Animals and the natural world were also a rich source of inspiration for cybernetic study in an attempt to understand systems of varying complexity and how they respond to environment change both to maintain homeostasis and also explore their environments to gain advantage from opportunities provided by the environment. If we can capture an essence of how this is achieved then this may prove to a valuable contribution to the development of environmentally aware, self-adaptive software.

3.2 Key Cybernetic Environmental Concepts

Variety

The term variety was coined by Ashby in a cybernetic context to represent the total number of states that a system can adopt. By system state we mean a snapshot of the unique values for each of the state variables at a particular moment in time. A simple example of this concept is a system consisting of three light-bulbs that can either be on or off at any moment in time. The variety of this system is considered to be the total number of states that can the system can adopt (Figure 4).



	Light 1	Light 2	Light 3
1	Off	Off	Off
2	Off	Off	On
3	Off	On	Off
4	Off	On	On
5	On	Off	Off
6	On	Off	On
7	On	On	Off
8	On	On	On

Figure 4 Variety of a "Simple Light-Bulb System"

Therefore, the simple light-bulb system can appear to adopt eight states, although this excludes exception states such as a light bulb being defective. Such exception states can cause significant problems to a system as they are often unaccounted for and consequently the system may not be able to respond to this particular system state.

A recent example of this factor would be the European Organization for Nuclear Research (CERN) particle accelerator which, in 2008, had major problems due to an unbalanced longitudinal force which the support structures in the magnets were inadequate to withstand. Therefore the system was unable to respond to the exception state, in this particular instance, and the Large Hadron System was shut down to allow for repair work to be undertaken.

To demonstrate how variety can increase significantly as system complexity increases then if one more light-bulb is added to the system then the variety of the system would increase from eight to sixteen potential states (again not including the considerable problems caused by exception states).

In a control system variety is encountered in two main ways:

- The variety in terms of the potential number of disturbances that a system can potentially be confronted with.
- The variety in terms of the potential number of responses (or actions) that the control system can "respond" with to counteract the effect of the disturbance and maintain homeostasis.

This situation was formalised by Ashby as:

The Law of Requisite Variety

> *"The variety of a regulator must be at least as large as that of the system it regulates"*
>
> (Ashby, 1956)

Ashby's view was that any control mechanism must have sufficient variety in terms of its responses when compared to the variety of the disturbances that it faces to maintain effective control of the system. This has been often quoted as that *"only variety can destroy variety"* (Ashby, 1956).

One difficulty in this approach is that when seeking to control complex systems increasingly tethered to real world environments, it soon becomes apparent that the control system seeking to maintain homeostasis must absorb any increase in the complexity of the environment it is trying to control, and provide a corresponding variety of response. This is Ashby's Law of Requisite Variety, which states that as a system becomes more complex, the controller of that system must in turn become more complex to respond to this increased variety. If the controller fails to absorb the variety it will experience situations that are out of its control and to which it will have no adequate response. This, in turn, may well threaten the survival of the system.

There are two cybernetic approaches to control the problem of variety overwhelming the controller of the system. Either the variety of the controlled system is reduced or attenuated to the number of states that the controller can respond to or conversely the variety of the controller is amplified to match or exceed that required to control the situation. This can be achieved in a number of ways, for example

> *"Managers may autocratically use threats or even guns as amplifiers of their own variety. And the most effective attenuator of environmental variety is often sheer ignorance within this subsystem of how the environment actually works. Management which is effective and ethical, however, will design the regulators, and put them in place"*
>
> (Beer S. , 2004)

To ensure the effective control of a system, the Conant-Ashby theorem (Conant & Ashby, 1970) states that "every good regulator of a system must be a model of that system" to ensure effective control. To achieve this, would require the controller to be or contain an "isomorphic" model of the system and its environment to be controlled. Isomorphic comes from the Greek word meaning "equal shape". By isomorphic model, we essentially mean a one to one mapping between the environment and the model. Obtaining such a model is problematic for complex systems operating in turbulent and changeable environments.

Laws (Laws, Taleb-Bendiab, Wade, & Reilly, 2003) suggests that

> *"when such isomorphism is not possible as in highly complex systems, then the regulator must be, i.e. contain, a strongly homomorphic model of the situation"*

where homomorphic is a one-to-many mapping, meaning the model becomes an abstraction of the environment and therefore is incomplete.

The quality of the model becomes of considerable importance if they are the primary input into the control system. Schwaninger states that a process cannot be better than the model on which it is based "except by chance because stochasticity can also favour the fool, at any time" (Schwaninger, 2004).

The principle of incomplete knowledge suggests that the model in an "open" control system is necessarily incomplete as "the moment information arrives, it is already obsolete to some extent" and that models cannot be expected to attain "any form of complete representation of an infinitely complex environment" (Heylighen F. , 2000).

Therefore, when thinking about modelling an "open" environment, the homomorphic model may well be have to be sufficient for our purposes.

The Law of Requisite Knowledge

> *"In order to adequately compensate perturbations, a control system must "know" which action to select from the variety of available actions"*
>
> (Aulin, 1982)

While having sufficient variety within the control system to respond to disturbances is necessary for control, it is not sufficient. The regulating system must also know which action to select given a certain disturbance and therefore have some knowledge related to the effectiveness of the action. Without this knowledge our systems would be restricted to randomly selecting actions until it stumbled upon a solution to the disturbance. Even once the correct action was chosen the system would not necessarily store this knowledge and may have to continue to cycle through its random actions the next time the system faced the same problem. Whilst this might not be a particular problem with our simple light-bulb system consisting of seven states, it is likely to be a significant problem for complex systems with significant variety requiring control. Therefore, as variety increases for a control system the requirement for a knowledge or learning element becomes increasingly important to the performance of that system.

A good regulator will require an action for each condition it will face and will have the ability to improve this knowledge over time. Feedback will be a necessary requirement of an effective *"learning"* system as changes in the environment or system may mean that whilst an action may successfully counter a disturbance today it may not be the most appropriate choice tomorrow. This feedback "adds to the knowledge base of intelligence required for the formulation of plans" (Achterbergh & Vriens, 2002).

Another feature of a good regulator will be the ability to extend its repertoire of actions by developing new actions in response to unforeseen environmental conditions. This will enable the control system to increase the variety of its actions in response to increased variety being observed in the environment.

3.3 Conceptual view of the environment

Hariri and colleagues suggests that we should consider the environment as consisting of two distinct parts – internal and external. He further defines the internal environment as consisting of "changes internal to the application that characterises the runtime state of the application" whereas the external environment can be thought of as "characterising the state of the execution environment" (Hariri, Khargharia, Chen, Yang, & Zhang, 2006).

The theoretical environment of a system is everything that is not included in the system (Figure 5). In effect, everything that is external to the system forms the environment of that system and then potentially changes in distant sections of the environment could eventually impact on the system in question, Such "interconnectedness of all things" form a basis of chaos theory.

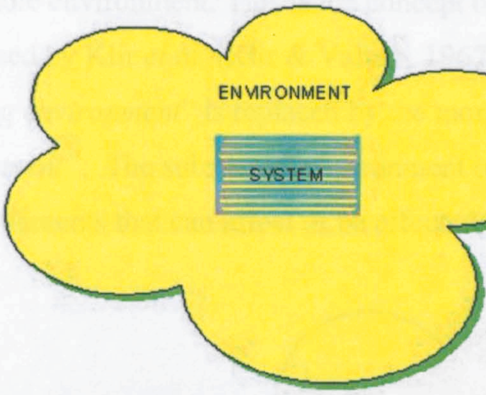


Figure 5 Theoretical Environment of a System

Experiments by Edward Lorenz in 1960 laid the foundations of chaos theory by demonstrating that the behaviour of the atmosphere is unstable with respect to small perturbations, which has since become known as the butterfly effect.
Lorenz asked the question of "whether the flap of a butterfly's wings in Brazil set off a tornado in Texas?" (Lorenz, 1972). An example of this theory is provided by Stewart who suggested that

> *"The flapping of a single butterfly's wing today produces a tiny change in the state of the atmosphere. Over a period of time, what the atmosphere does diverges from what it would have done. So, in a month's time, a tornado that would have devastated the Indonesian coast doesn't happen. Or maybe one that wasn't going to happen, does"*
> (Stewart, 2002)

Lorenz proposed that

> *"If a single flap of a butterfly's wings can be instrumental in generating a tornado, so can all the previous and subsequent flaps and its wings, as can the flaps of the wings of millions of other butterflies, not to mention the activities of innumerable more powerful creatures, including our own species"*
> (Lorenz, 1972)

This means that is would be impossible to predict weather patterns at a sufficiently distant future time. The butterfly effect would require us to consider this theoretical environment of a system, however, in practice we usually want to restrict our interest to a finite number of defined relations between the system and its environment rather than consider the whole environment. This is the concept of the "substantial environment" proposed by Klir *et al.* (Klir & Valach, 1967). Therefore, the concept of the *"all embracing environment*" is replaced by the more restrictive concept of *"substantial environment"*. The substantial environment of the system can be considered as those elements that can affect or be affected by the system (Figure 6).

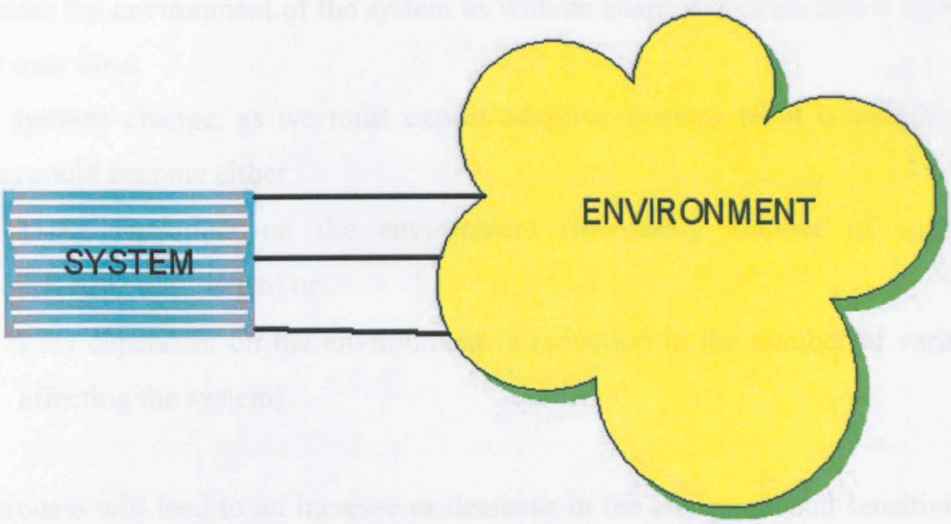


Figure 6 The Substantial Environment

Even taking the concept of a "substantial environment" is still problematic as it would be necessary to identify all the essential factors that affect or are affected by a system to enable this coupling with the environment to take place. In an adaptive system the substantial environment is likely to change and therefore couplings will change to reflect the turbulence of the environment. The ability of the system to recognise and respond to such changes and shifts in the environment becomes paramount to its ability to survive in that environment. In this sense, the system "selects" the components of the substantial environment.

> *"We have seen that the individual organism in some sense determines its own environment by its sensitivity. The only environment to which the organism can react is one that its sensitivity reveals. The sort of environment that can exist for the organism, then, is one that the organism in some sense determines. If in the development of the form there is an increase in the diversity of sensitivity there will be an increase in the responses of the organism to its environment, that is, the*

organism will have a correspondingly larger environment. In this sense, it selects and picks out what constitutes its environment."

(Mead, 1934).

This concept of environmental sensitivity is an important element in developing what constitutes the environment of the system as with an adaptive system this is likely to evolve over time.

When systems change, as we must expect adaptive systems to, it is possible the systems could become either

- More dependent on the environment (increasing number of variables affecting the system) or
- Less dependent on the environment (a reduction in the number of variables affecting the system)

This process will lead to an increase or decrease in the environmental sensitivity of the system, which can be achieved by an increase or decrease of sensing and environmental scanning. The more sensitive and responsive a system is to its environment the better it can survive and adapt to environmental changes. However, there is, of course, an overhead associated with environmental monitoring and potentially a danger of information overload from a heightened environmental sensitivity and consequently the need for effective management and filtering of information being provided by the environment. This management will consist of providing environmental data at the appropriate time and level.

Therefore, a successful adaptive system should be able to adapt its environmental sensitivity to maintain optimum performance of the overall system. Some further refinement of the composition of the "substantial environment" was provided by Gallopin and Patten. Gallopin suggested that the environment could be divided into

- Purely Influencing Environment (that portion of the environment coupled to elements of the system via its inputs but which the actions of our system can not influence)

- Purely Influenced Environment (that portion of the environment coupled to elements of the system via its outputs which the system can affect but can not be affected by)
- Influencing / influenced Environment (that portion of the environment that can both influence the system and are also influenced by it)

(Gallopin, 1981)

This approach is useful in classifying our learning requirements and how elements of the environment need to be monitored. For example, a part of our environment that is "purely influencing" may not require monitoring in terms of the effect that the actions of our system had on this element of the environment as the system has no "influence" over that aspect of the environment.

Also the "purely influenced" environmental elements will not affect the system even though our system impacts on this element. Therefore our system may choose to limit monitoring or even abandon it altogether.

On the other hand, Patten (Patten, 1978) suggests a classification system consisting of

- Operational Environment
- Potential Environment
- Non-Environment

This approach is also of value as it breaks the environment into those elements that currently affect the system, those that may affect the system in the future and those that will not affect the system.

It may be that a combination of these two views of the environment will prove useful in developing an effective model of the environment in terms of adaptive software development.

It is easy to comprehend why the "influencing" environment would be important to the system, however the "influenced" environment may also be of importance when two or more interacting systems are considered together. It may be that the purely influenced environment of one system is an influencing environment of an interacting system. The most successful adaptive software is likely to use "feedback" from the environment to lead to improvements and/or learning in our systems. If this environmental element is likely to be important, the environment of the system would need to be viewed as a subsystem (Figure 7).

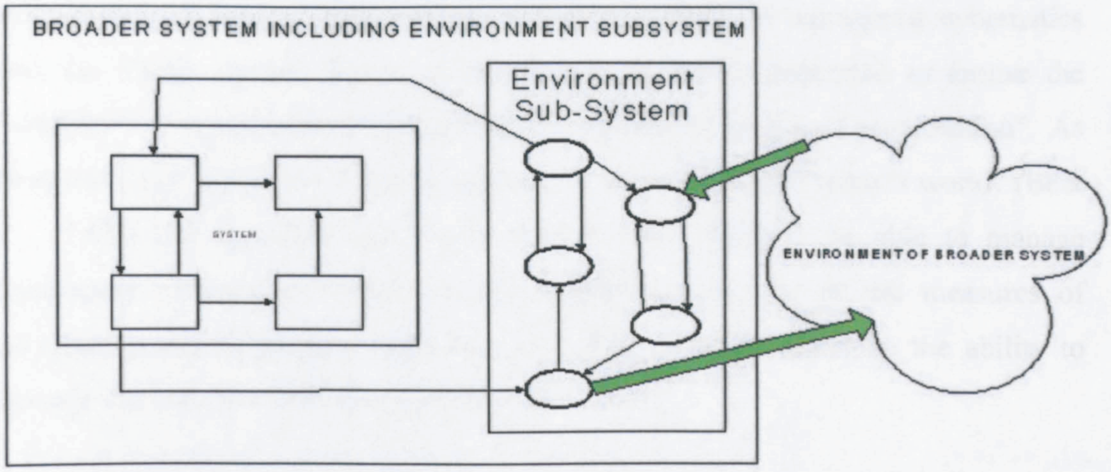

BROADER SYSTEM INCLUDING ENVIRONMENT SUBSYSTEM
Environment Sub-System
SYSTEM
ENVIRONMENT OF BROADER SYSTEM

Figure 7 Broader systems including environmental subsystem

This environmental subsystem may be necessary if we are interested in the organisation and responses of the environment to system influences.

3.4 Second Order Cybernetics and the Viable System Model

The fundamental or first order cybernetic principles of the earliest cybernetic investigators were extended by the likes of Von Foerster, Maturana, Pask and Beer to understand the role of the observer on the regulation of systems, and leading to the establishment of "second order cybernetics", sometimes described as "the cybernetics of cybernetics" (Foerster, 2003). They suggested that whilst attempting to study and understand a social system, observers were unable to separate themselves from the system or prevent themselves from having an effect on it. Applying this effect to a software system analogy we must be conscious of the effect that monitoring a software system has on the system and indeed the environment. It may well be the case that the very fact of monitoring the system could itself be detrimental to the performance of the system. It may be impossible to quantify the effect that observing the system has on the system as it is difficult to know how it performs when observations are not occurring.

This second order cybernetics was increasingly concerned with autonomous systems that sought to define their own goals rather than in the regulation of controlled systems with pre-defined goals. Beer established a subset of the field of cybernetics, namely managerial cybernetics, which attempted to apply cybernetic principles to the management of human organisations. The main vehicle of managerial cybernetics was the Viable System Model (VSM) Figure 8, which attempted to ensure the "viability" of organisations by providing a "cybernetic model of organisation". As Beer had himself suggested that "complexity is the very stuff of today's world" (Beer S. , 1975) and therefore any viable system would need to be able to manage complexity effectively. Variety is often considered as one of the measures of complexity and therefore a requirement of viable system would be the ability to manage the variety within the environment.

The VSM has subsequently been mapped onto a diverse range of business organisations and indeed a whole country (Beer S. , 1995). Cybernetic thinking may provide a solution to this management of complexity from an autonomic computing viewpoint. Viable systems have been defined as *"being robust against internal malfunction and external disturbances and have the ability to continually respond and adapt to unexpected stimuli allowing them to survive in a changing and unpredictable environment"* (Laws, Taleb-Bendiab, Wade, & Reilly, 2003). To achieve this goal the key cybernetic principles identified by Morgan (Morgan, 1986) when talking about the applicability of cybernetics to business organisational learning are relevant to adaptive software system design in that they must be able to

- Sense, monitor, and scan significant portions of their environment
- Relate this information to the operating norms that guide system behaviour
- Detect any significant deviations from these norms
- Initiate corrective action when discrepancies are detected

These cybernetic goals outlined previously are closely related to the desirable properties of autonomic and self-adaptive software as defined by IBM and the research community currently investigating the area.

Leonard states that as the system is being buffeted by events in the environment it must have

> *"the capacity to adapt in order to cope with them. The success of that adaptation depends on the quality of the system's intelligence about the environment and the resources available to make use of that intelligence"*
>
> (Leonard, 2009).

These attributes again mirror many of those required for robust autonomic software and therefore lends weight to the potential benefits of using cybernetic theory for such system development.

The concept of viable systems that can adapt to changing environments would seem to be applicable to the development of effective adaptive software systems especially as one of the features of the VSM is its recursive nature, which makes it scalable to systems of increasing size and complexity. Schuhmann suggests that systems are *"produced by observations, which are generated by their system. A system is always an observer that gives birth to itself and vice versa. Enlightenment enlightens itself"* (Schuhmann, 2004). It seems likely that our prototype will use the concept of observing and learning from the environment to provide a model of the environment for our particular system at a particular moment in time. Our sense of the environment becomes central to the process of successful adaptation in that a cybernetic view would maintain that "in order to make adaptation possible, the environment provides much of the determination about how the organism should act" (Bowker & Chou, 2009).

DIAGRAM
ON THIS PAGE
EXCLUDED
UNDER
INSTRUCTION
FROM THE
UNIVERSITY

Figure 8 Beer's Viable System Model

(Beer S. , 1985)

There are six major systems of the Viable Systems Model which include:-

S1 Operations. System 1 performs the basic activities of the enterprise. These activities require monitoring and management control in order to ensure that they are working as expected.

S2 Co-ordination. System 2 performs the co-ordination of S1 activities within the organisation as parts of S1 activities interact with each other. S2 seeks to limit the occurrence of S1 activities experiencing instability in their operations.

S3 Control. System 3 could be considered as the optimising system within the VSM. It is concerned with using existing resources and ensuring that it uses the data from operations for effective decision making.

S3* Audit. System 3* provides the auditing facilities for intermittent audits of S1 activities to be performed. This provides data above and beyond normal reporting procedures.

S4 Intelligence. System 4 is responsible for adaptation of the enterprise in response to a changing environment. It uses information obtained from both a model of the external environment and its own model of internal capability to develop plans. Therefore, it is necessary that these models are current to ensure any decisions and plans are based on up-to-date information.

S5 Policy. System 5 determines the policy development and overall purpose of the enterprise. Policy is informed by the S4 Intelligence activities regarding environmental models and current capability.

Beer suggests that the cybernetic properties embodied in the VSM emerge from the "holistic performance of the entire system and they do not reside anywhere at all least of all can any one of them be identified with any one subsystem" (Beer S. , 1994).

In terms of the role of the environment and the application of the VSM model system S4 is of most interest. S4 (intelligence) is concerned with making sense of what may confront the system in the future by collecting and analysing information from the environment i.e. identifying threats and benefits in time to either avoid or take advantage of them.

Laws *et al.* described S4 as being "concerned with planning the way ahead in the light of external environmental changes and internal organisational capabilities" and that to ensure these plans are grounded in an accurate appreciation of the current environment it needed to "contain an up-to-date model of organisational capability" (Laws, Taleb-Bendiab, Wade, & Reilly, 2003).

Lewis states that the goal of S4 is to "define its boundaries, to model and monitor, and to make predictions on future environmental trends" and that this would require an in-built model which is a simplification of its environment (Lewis, 1997).

Schwaninger saw S4 as providing a comprehensive external orientation to the future, the ability to grasp, diagnose and model (Schwaninger, 2006). The difficulty in building these models is a significant consideration and one approach discussed by Jackson (Jackson, 1991) and Gregory (Gregory, 2007) is by treating such systems as black boxes with only the inputs and outputs clear but the internal workings "shrouded from view". This means that systems can be investigated without knowledge or assumptions about its internal structure or parts that will not "shed any light on how to manage such a system".

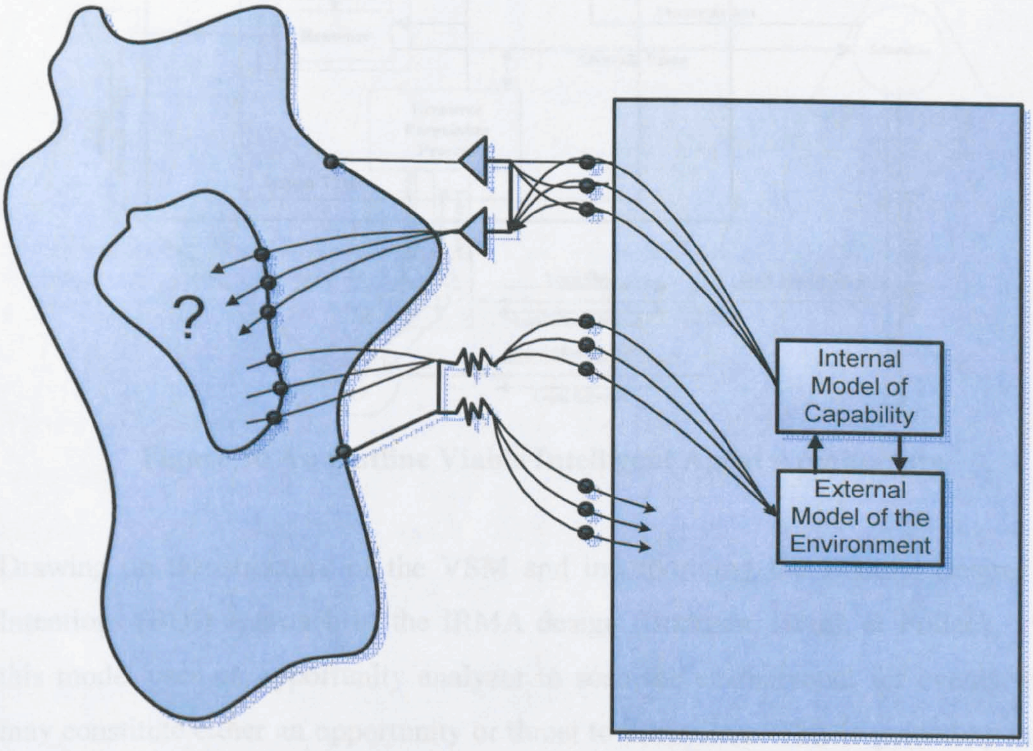

?
Internal Model of Capability
External Model of the Environment

Figure 9 System 4 Mechanism For Interrogating The Problematic Environment

Laws *et al.* has developed an outline viable Intelligent Agent Architecture (Figure 10) incorporating environmental change in the S4 (Intelligence) stage of the model.

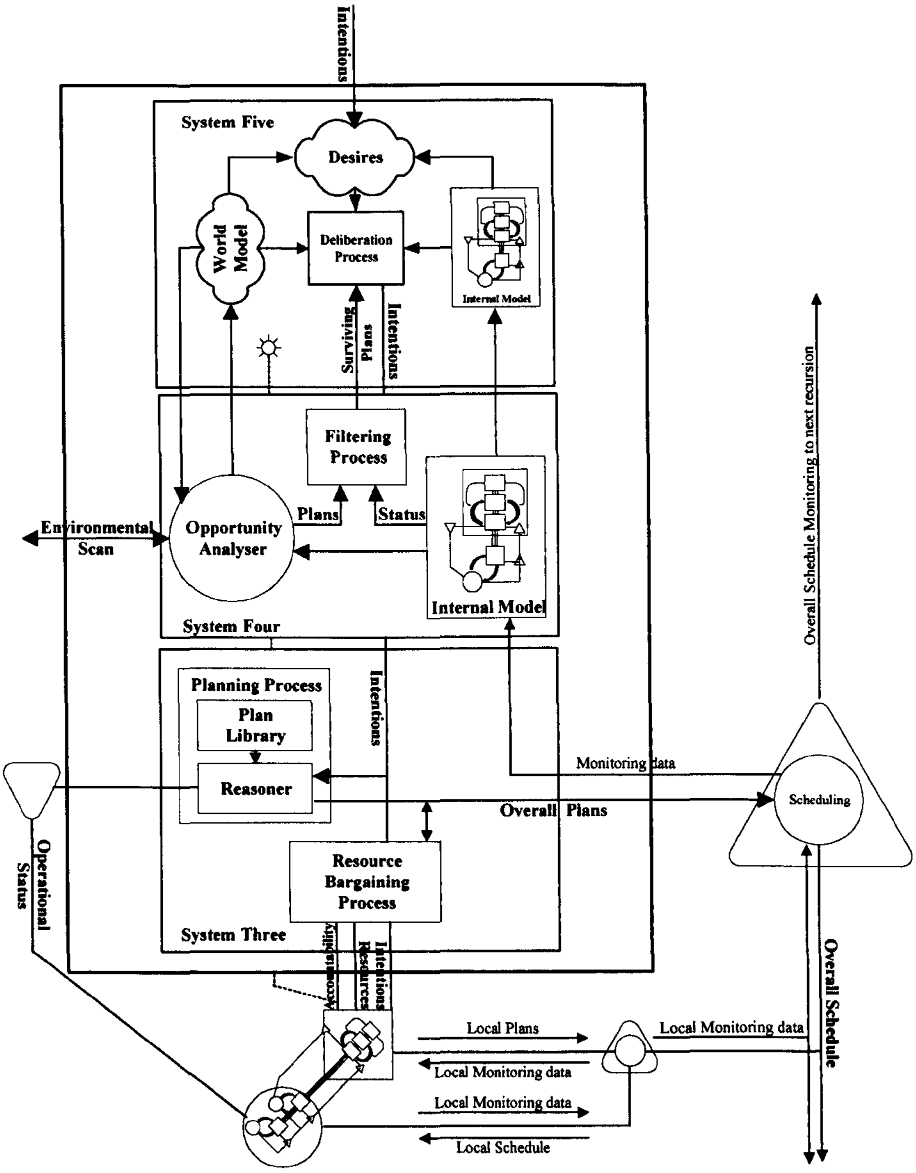


Figure 10 An Outline Viable Intelligent Agent Architecture

Drawing on the structure of the VSM and incorporating the Beliefs, Desires and Intentions (BDI) approach of the IRMA design (Bratman, Israel, & Pollack, 1988) this model uses an opportunity analyzer to scan the environment for events which may constitute either an opportunity or threat to the system. This is guided by the S5 desires model which holds the goal of the system at the current time. Information from the environment is passed to S5 which holds a model of the "World" and S3 which deal with current system capabilities and planning, although planning in this context is closer to plan selection and realisation based on the intentions determined by S5. Beer realised the difficulty of effective planning in that "if the plan when published is conceived out of a model that assumes complete information, it is mistaken" (Beer S. , 1991).

The interface with the environment is vital for maintaining a current view of the "world" to allow the system to adapt more successfully as

> *"The model identifies the necessary and sufficient communication and control systems that must exist for any organization to remain viable in a changing environment"*
>
> (Laws, Taleb-Bendiab, Wade, & Reilly, 2003).

There are a number of practical problems with the concept of the opportunity analyzer including:-

- How do we scan the environment?
- What is the model of our environment (world view) likely to contain?
- What are we scanning for?
- How is filtering to be applied to avoid information overload?
- Can we forecast future changes in the environment to enable our systems to become more predictive rather than strictly reactive which may improve survivability?
- How can we scan what is important to our system?

To overcome some of these challenges it seems reasonable to suggest that a learning approach may have a role to play. Feedback from the environment may be used in an adaptive software system in order to "learn" a model of the environment and therefore allow the system to make choices / decisions based on that model.

Using data received from detectors would be a potential method of modelling the environment and therefore allow the system to improve its environmental model and sensitivity over time.

Bousquet *et al.* (Bousquet, Boucheron, & Lugosi, 2004) roughly summarises the process as

- Observe a phenomenon
- Construct a model of that phenomenon
- Make predictions using this model

This can be considered as part of a standard learning model, if it included an end stage of looping back to the observation stage. They concede that the above definition is very general and that the goal of machine learning should be to automate this process and the goal of learning theory is to formalise it.

The goal of our system is to provide a basis for refining our models and predictions as more data is obtained in order to provide more optimal predictions, which our system can use as a basis for adaptive choices.

One of the key challenges to the development of effective adaptive software is the ability to observe and interpret the environment in which it exists.

> *"Detailed, dynamically gathered information is essential to ensure the composability, dependability and adaptability of software systems. Without such information, it will not be possible to determine whether a system should be changed, or to locate points of failure that need to be fixed"*
>
> (GIWG, 2001)

Dobson *et al.* suggests that the

> *"viability of environmental sensing – essential for effective science and policymaking – therefore depends on sensor systems' ability to self-manage in the face of a changing environment"*
>
> (Dobson, Sterritt, Nixon, & Hinchey, 2010).

We will need a system that can use detectors to collect observations from the environment and interpret these observations to provide the systems with the information requirements to subsequently make "good" decisions to improve performance and robustness.

3.5 Summary

The research challenges outlined in Chapter 2 and the analogy with the natural world as an inspiration has led to an investigation into the applicability of cybernetic principles as a framework for developing environmentally aware software systems. The field of cybernetics appears to have great applicability as an important piece of the self adaptive puzzle as its primary focus is based on understanding complex systems and the maintenance of homeostasis within those systems. Also, much of the fundamental underpinning behind cybernetics comes from the natural world and the complex systems within it. If we are to accept the marvel of the natural world in producing robust and fit solutions and overcome the problems associated with environmental change then we should seek to use the millions of years of experience that nature has provided us with.

The Viable System Model provides a framework for designing complex yet inherently robust systems and appears to have applicability to the design of software systems as outlined by Laws *et al.* (Laws, Taleb-Bendiab, Wade, & Reilly, 2003). System 4 of the Viable System Model is of most interest to this particular research problem and fundamentally requires that a system holds both a model of the environment and also a model of the internal capability of the system. This element will inform many of the requirements, of environmental management, of our system such as detecting a change in context or a change in needs but also knowing the space of adaptations that our system can effect.

By using cybernetic principles and conceptualising our view of the environment we can seek to develop a learning approach that will enable our prototype system to develop a current model of the environment and capability of the system and use this information to make “good” decisions in relation to environmental change.

Chapter 4: An Approach to Provide an Adaptive Model of the Environment

"Learning Is Not Compulsory...Neither is Survival"
William Edwards Deming

4.1 Learning From The Environment

It is virtually impossible to define a complete and accurate environmental model during design time for any but the most trivial of systems. Prolonged interactions with the "real-world" would render such a model increasingly inaccurate over any significant period of time. To embrace a truly cybernetic approach to designing our systems then it seems reasonable to design a solution with the capability to construct and maintain a current model of the environment.

To achieve this capability, the approach needs to be able to perform not only online learning but also continuous learning to enable environmental change to be reflected in the model to achieve, at least, a strongly homomorphic model of the environment. This requirement for "active" learning is supported by Epshteyn *et al.* who suggest that the "optimal policy computed offline in an imperfectly modelled world may turn out to be suboptimal when executed in the actual environment" (Epshteyn, Vogel, & DeJong, 2008).

Reinforcement learning is a well established machine learning technique and seems to have applicability in the context of this particular research. If we can interact with our environment through the combined use of detectors and effectors and can measure the impact of our actions through appropriate metrics then we have the fundamental building blocks necessary.

These building blocks potentially give the ability to interact with the environment by ascertaining the effectiveness of actions on the environment but a further technique is required for effective exploration of the environmental search space. This exploration will enable a current model of the operational environment to be developed where a system can respond appropriately to current environmental conditions. This exploration will be an ongoing process whereby the model is adjusted to maintain its currency and therefore value to the controlling system.

A re-occurring theme within this thesis has been taking inspiration from nature for the solution to the problem of designing robust software. It seems appropriate to continue this approach when considering potential approaches as to how this prototype will actually be developed.

4.1.1 Online Learning

Online learning as an approach means that the system is "created" with no prior knowledge about its environment and is forced to "learn" about it whilst making operational decisions. Off-line training would mean that a system is given "training data" with which to build up its knowledge before it is introduced into such a "live environment".

Das *et al.* found that the off-line model building approach was only applicable in "low-dimensional state and action spaces" and that it would not be suitable for more complex and larger scales applications (Das, Kephart, Lefurgy, Tesauro, Levine, & Chan, 2008). Schulenburg *et al.* stated that continual re-training may not be the answer in that it may be expensive and there is a potential problem about when to switch from the old model to a new one (Schulenburg & Ross, 2002).

Therefore a desirable property of a modelling prototype system existing in complex environments would seem to be the ability to build an effective model using on-line and continuous learning rather off-line training as a general approach and also include that of applicability and suitability for large-scale complex systems.

4.1.2 Reinforcement Learning

"Reinforcement learning is the study of how animals and artificial systems can learn to optimise their behaviour in the face of rewards and punishments."
(Dayan & Watkins, 2001)

The basic premise is that by using a system of reward and punishment in response to an agents interaction with the environment, reinforcement learning can be utilised in terms of exploring unknown and potentially complex environments. A definition used by Yamada (Yamada & Yamaguchi, 2010) is that

"an agent selects and engages in behaviours with respect to sensory inputs obtained from sensors, As a result, learning is performed by repeating a cycle in which a reward from the environment and sensory input for the next state are given."

Kaelbling *et al.* found that reinforcement learning differed from supervised learning in that:-

i) *"After choosing an action the agent is told the immediate reward and the subsequent state, but it is not told which action would have been in its best long term interests"*

ii) *"It is necessary for the agent to gather useful experience about the possible system states, actions, transitions and rewards actively to act optimally"*

iii) *"on-line performance is important: the evaluation of the system is often concurrent with learning"*

(Kaelbling, Littman, & Moore, 1996)

According the Langlois *et al.* the goal of reinforcement learning is to

"choose the best action for the current state. More precisely, the task of reinforcement learning is to use observed rewards to learn an optimal (or almost optimal) policy for the environment"
(Langlois & Sloan, 2010).

Das *et al.* suggested that the solution to the problem of scalability and complexity outlined later in Section 4.2 could be in the use of model based reinforcement learning or a form of hybrid reinforcement learning which has shown "promising initial results in learning policies in high-dimensional state spaces" (Das, Kephart, Lefurgy, Tesauro, Levine, & Chan, 2008).

Bull *et al.* found that evolutionary algorithms and reinforcement learning are increasingly being used in the design of complex systems as more traditional learning algorithms require "detailed knowledge of and control over the computing substrate involved" (Bull, Budd, Stone, Uroukov, Costello, & Adamatzky, 2008).

4.1.3 Self-Adaptive Learning

A key component of a successful learning approach is the notion of reinforcement learning. This element relies on the notion of feedback from the environment to ascertain whether an action taken by the system has had a subsequent beneficial or detrimental effect on the environment. Reinforcement learning should allow the system to improve its response by using previously successful approaches when subsequently facing the same environmental situation. An example of a reinforcement feedback loop can be observed in Figure 11.

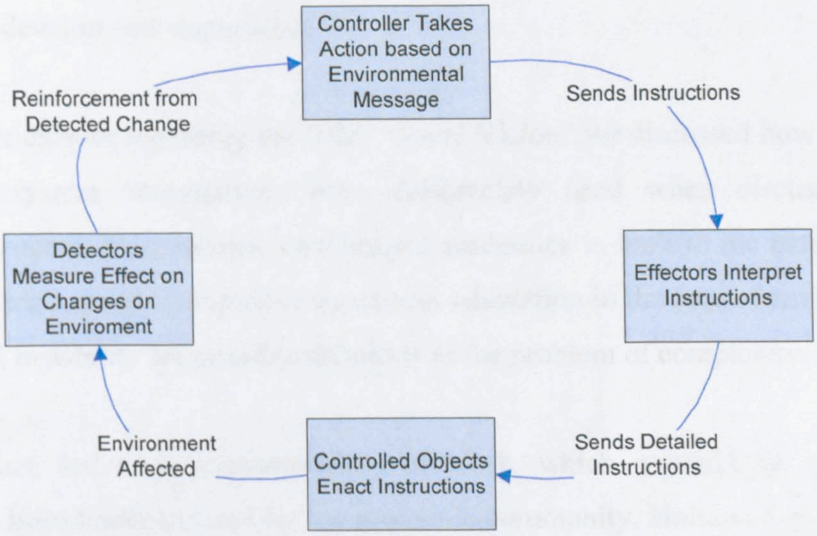

Controller Takes Action based on Environmental Message
Sends Instructions
Effectors Interpret Instructions
Sends Detailed Instructions
Controlled Objects Enact Instructions
Environment Affected
Detectors Measure Effect on Changes on Enviroment
Reinforcement from Detected Change

Figure 11 Reinforcement Feedback Loop

This identifies a number of initial requirements for our design. We will therefore require the following components:

i) An appropriate test-case system for the prototype Environmental Modelling, Monitoring and Adaptive System (EMMA) to model and attempt to control.

ii) The development of an appropriate performance metric that can be obtained from the test case system environment.

iii) A detector to provide environmental data to our system.

iv) An effecter that will enable our system to influence the environment of the test case system.

v) The system must be able to evaluate the performance of its actions and learn from these actions to provide improved performance over time.

vi) The system should be able to dynamically adjust its "model" of the environment so it always holds an appropriate model of the environment.

vii) The system should be able to hold a view of its own capability and therefore its ability to enact influence on the environment it is seeking to monitor and control.

viii) The system should include an element of historical learning in order to inform future developments.

ix) The system needs to be able to explore its environment and be able to develop new approaches.

In an earlier chapter regarding the IBM "Grand Vision" we discussed how biological regulation system connotations were deliberately used when discussing truly adaptive systems. This, in turn, encouraged academics to look to the natural world, which provided many examples of successful adaptation in the face of environmental change and instability for possible solutions to the problem of complexity.

This, in fact, led to a re-examination of work, which it could be argued had previously been under-utilised by the academic community. Holland had provided a seminal body of work on the use of genetic algorithms to provide a robust search in complex spaces (Holland J. , 1975). Genetic algorithms are search algorithms that are based on the

> *"mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomised information exchange to form a search algorithm with some of the innovative flair of human search"*
>
> (Goldberg, 1989)

This work was supplemented in 1977 when Holland presented the first implementation of his learning classifier system (Holland & Reitman, 1978). A classifier system has been defined as a *"machine learning system that learns syntactically simple string rules (called classifiers) to guide its performance in an arbitrary environment"* (Goldberg, 1989). There have been further revisions and refinements to this approach but perhaps it is only recently that the true value of this approach is being realised in our goal of providing adaptive software that can respond effectively to environmental change and instability.

Bull suggests that the reason for the waning interest in learning classifier systems was that Holland's full system *"was somewhat complex and practical experience found it difficult to realise the envisaged behaviour / performance"* (Bull & Lanzi, 2009). One approach to developing learning systems has been to train the system using offline batch training data in order to provide the "learning" that the system requires. However, there is a difficulty in obtaining this data in turbulent and changeable environments where sufficient data cannot be obtained in advance. Gao states that

> *"learning systems need to deal with the incrementally coming cases and the feedbacks from the environments while such requirements are not necessary in batch learning"*
>
> (Gao, Zhexue Huang, & Wu, 2007).

4.2 Possible approaches to solving the problem

Whilst recognising that there are a number of evolutionary approaches that could be used to solve the research problem of developing an environmental model, it would appear that an approach using Learning Classifier Systems (LCS) and Genetic Algorithms (GA) is a valid experimental approach for the purposes of this thesis. As

Heylighen commented, when talking about cybernetics, *"many of the core ideas of cybernetics have been assimilated by other disciplines, where they continue to influence scientific developments"* (Heylighen F. , 2001). Asaro identified that genetic algorithms were one of the many significant technical developments to have been inspired by the field of cybernetics (Asaro, 2006).

LCS and GAs have similarities to Ashby's cybernetic based definition of the ultra-stable system in that his inner control loop could be considered to be the LCS whilst the outer loop (providing adaptive behavioural change) could easily be represented by the genetic algorithm element.

The rationale for this decision is based on the obvious links between cybernetics and the complex adaptive systems movement, of which John Holland, who first devised the notion of learning classifier systems (Holland J. , 1975) was a key contributor.

4.3 Genetic Algorithms

At this point in the thesis it is useful to provide an overview of genetic algorithms and their development over the past decades. The fundamental essence of genetic algorithms is rooted in the Darwinian view of the "survival of the fittest" and the fact that genetic algorithms try to *"refine a population of a problem solution through experience with the trained data"* (Sarkar & Sana, 2009). This effectively means that searches of the complex space are not purely random but targeted as "stronger" string structures are bred to provide targeted search points and improve performance. According to Goldberg they work because *"they combine survival of the fittest among string structures with a structured yet randomised information exchange to form a search algorithm with some of the innovative flair of human search"* (Goldberg, 1989). Genetic algorithms have been applied in applications such as the finance based scheduling of large-sized projects activities where it was found to perform robustly in terms of its effectiveness to search for optimal solutions (Abido & Elazouni, 2010).

According to Goldberg (Goldberg, 1989) genetic algorithms are more robust than traditional optimisation and search procedures because

- They work with a coding of the parameter set, not the parameters themselves.
- They search from a population of points, not a single point.
- They use payoff (objective function) information, not derivatives or other auxiliary knowledge.
- The use probabilistic transition rules, not deterministic rules.

4.3.1 How Do Genetic Algorithms Work?

The fundamental basis of the strength of genetic algorithms lies in the reproduction of increasingly fit structures in response to a particular problem domain. A "solution" to the problem domain is represented as a set of "chromosomes", where each chromosome is composed of alleles (one allele can equal 0 or 1), which represent one characteristic. Each set of chromosomes represent one "putative solution" and a population of such "solutions" are initially randomly generated and then tested for effectiveness. They are rewarded or punished accordingly and such rewards contribute to the fitness or otherwise of each solution. In the breeding process, weak solutions are discarded in favour of replacement "bred" from stronger solutions, and then the process beings again.

This is achieved by three basic operators which are:-

i) **Selection.** The role of the selection operator is to select high strength chromosomes to act as parents in the reproduction stage of the process. The strength of the chromosome is determined by a reinforcement learning process where good chromosomes have been suitably rewarded and poorly performing chromosomes have been punished.

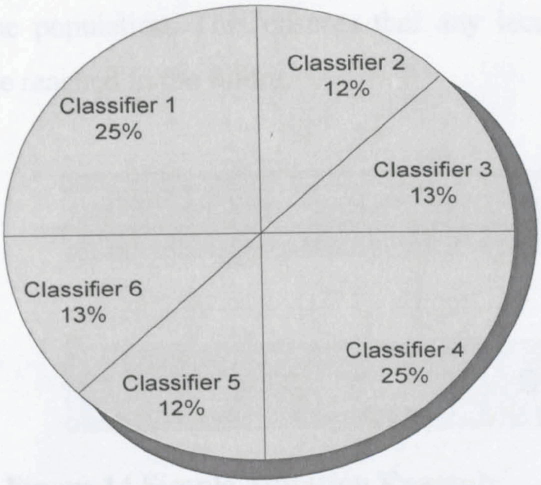


Figure 12 Selection Using A Roulette Wheel

ii) **Crossover.** Crossover involves exchanging a randomly selected segment between the selected parents to produce "child" chromosomes. Booker et al. suggests this is the key to understanding the strength of genetic algorithms it that "a good building block is a building block that occurs in good rules. The GA biases future constructions toward the use of good building blocks" (Booker, Goldberg, & Holland, 1989).

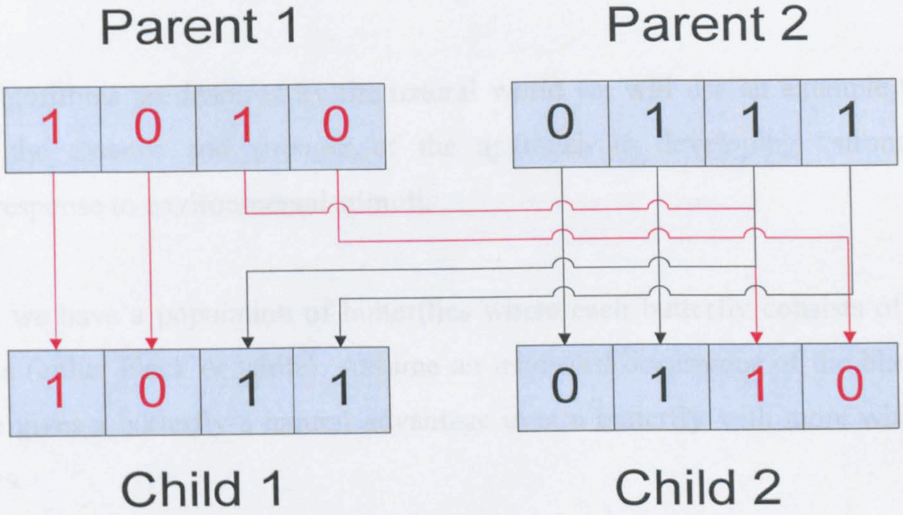


Figure 13 Simple Crossover Example

iii) **Mutation.** The role of mutation is to prevent premature convergence and to allow genetic material which may have been bred out of a population, or indeed never existed in the population, to be introduced or reintroduced back into the population. This ensures that any location in the search space may be reached in the future.

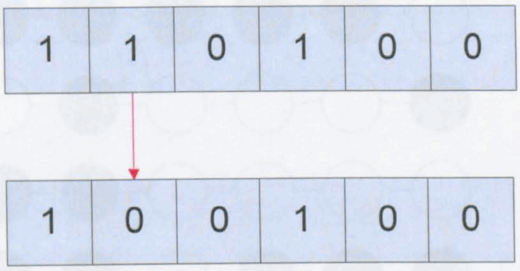


Figure 14 Simple Mutation Example

A proportion of the existing rule-set is replaced by "child" chromosomes during each run of the genetic algorithm process.

These fundamentally simple building blocks produce a powerful mechanism for the searching of complex spaces and producing fit rules to exploit the problem domain at that particular moment in time.

4.3.2 A Simple example of the Genetic Algorithm using Crossover

As genetic algorithms are inspired by the natural world we will use an example to demonstrate the essence and strength of the approach to developing "strong" solutions in response to environmental stimuli.

Assume that we have a population of butterflies where each butterfly consists of 8 chromosomes (either black or white). Assume an increased occurrence of the black chromosome gives a butterfly a natural advantage over a butterfly with more white chromosomes.

Therefore, butterflies with increased number of the black chromosome are more likely to survive into subsequent generations. The initial population consists of eight butterflies with an overall equal number of white and black chromosomes (i.e. randomly generated)

1	●–●–●–●–○–○–●–●	Strength 6
2	○–○–●–●–●–●–●–○	Strength 5
3	○–○–○–●–○–○–○–●	Strength 2
4	●–●–●–●–○–○–○–○	Strength 4
5	●–○–●–●–○–●–●–●	Strength 6
6	●–●–●–○–●–●–○–○	Strength 5
7	○–○–○–●–○–○–●–○	Strength 2
8	○–○–●–●–○–○–○–○	Strength 2

Figure 15 Generation 1 of the Butterfly Population

The strength of the butterfly is calculated by the total of black alleles which exist in its genetic make-up.

The total strength (in terms of black alleles) of this population of butterflies is 32.

At the end of a population generation the four weakest butterflies (in terms of black alleles) are lost giving the population outlined in Figure 16.

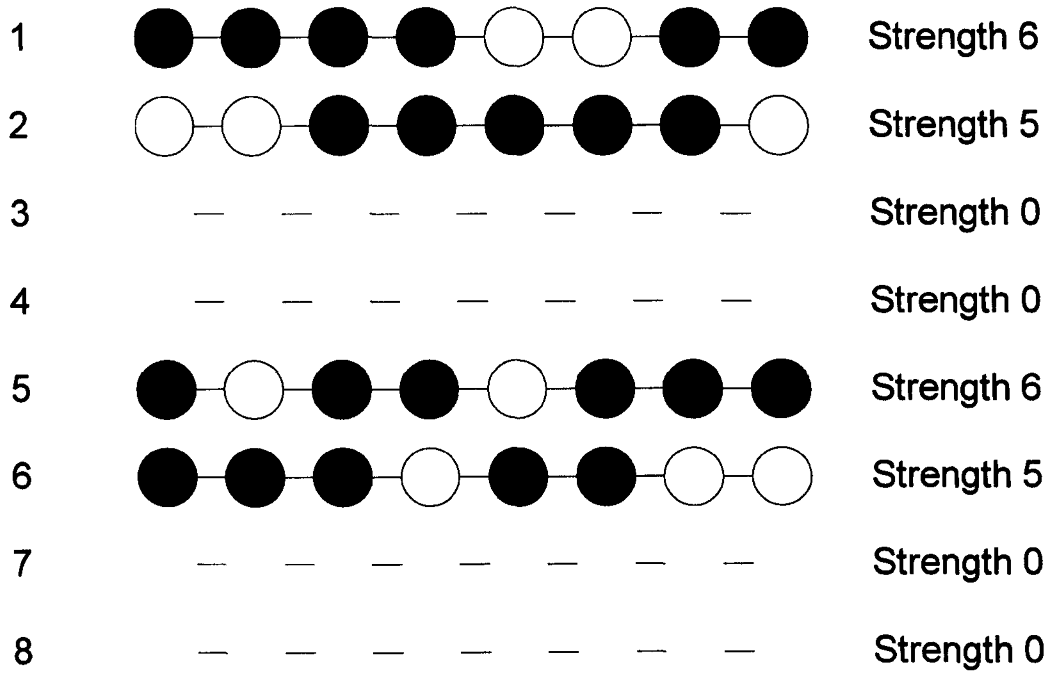


Figure 16 Generation 1 after weakest butterflies are lost

The remaining strong butterflies breed to produce four offspring to replace the "lost" butterflies.

Any random crossover point would produce similar results but for the purposes of this explanation we will use a standard crossover point of 3 for this particular generation.

At this point we

1. Breed butterflies 1 and 2 to create new offspring at position 3 and 4 (using crossover point at Position 3)

2. Breed butterflies 5 and 6 to create new offspring at position 7 and 8 (using crossover point at Position 3)

The 2nd generation butterfly population can be observed in Figure 17.

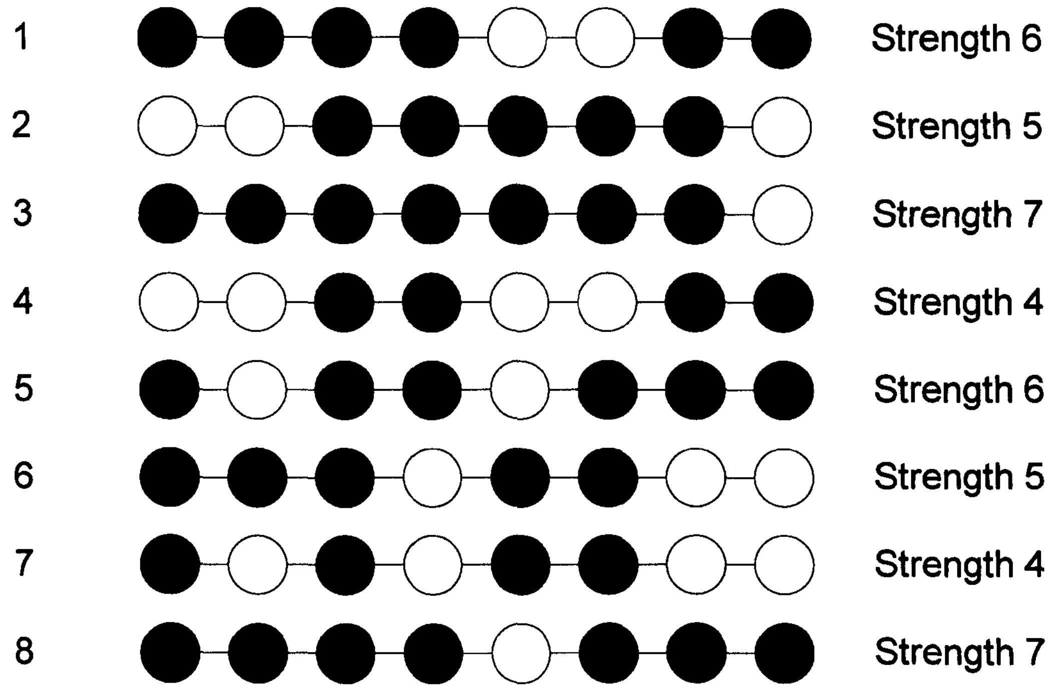


Figure 17 Generation 2 of the Butterfly Population

The strength of the population has increased from 32 to 44. This is due to the parent butterflies "passing on" their black alleles to their children.

At this point it can be observed that the four "strongest" butterflies are 1, 3, 5 and 7. These are the butterflies which will survive into the 3rd generation whereas butterflies 2, 4, 6 and 7 will be lost during this generation.

For the 3rd generation we will

1. Breed butterflies 1 and 3 to create new offspring at position 2 and 4 (using crossover point at Position 4)

2. Breed butterflies 5 and 7 to create new offspring at position 6 and 8 (using crossover point at Position 4)

The 3rd generation butterfly population can be observed in Figure 18.

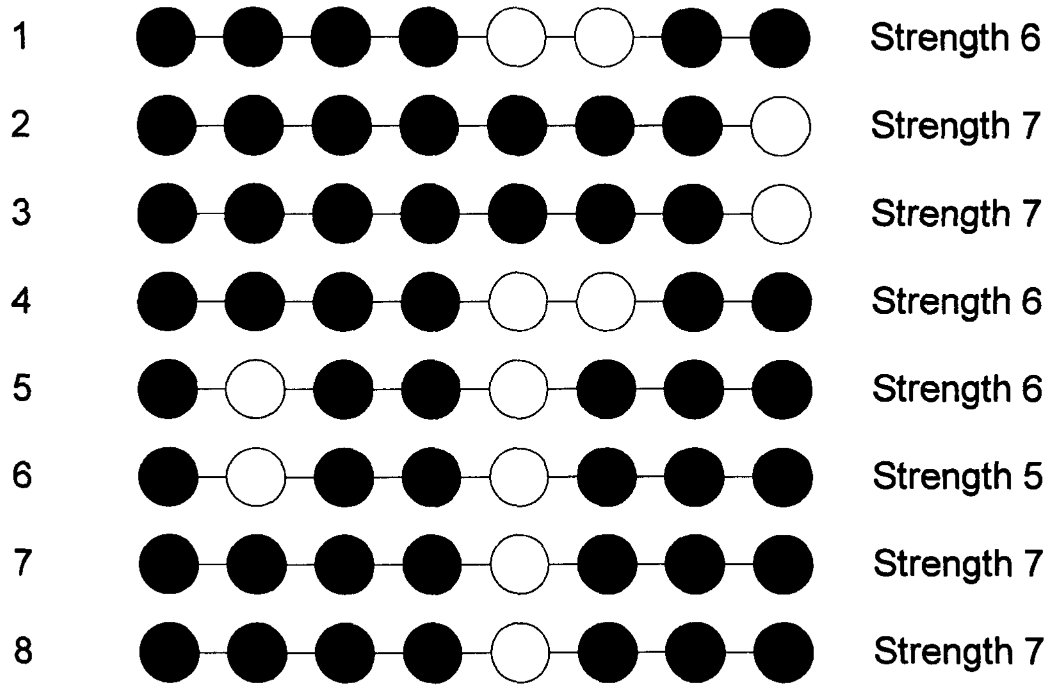


Figure 18 Generation 3 of the Butterfly Population

The strength of the population has increased from 44 to 51. This is again due to the parent butterflies "passing on" their black alleles to their children. If this experiment was continued the white alleles would be totally bred out of the butterfly population in future generations (we are assuming there is no mutation in this particular example).

If the environmental balance changed to favour butterflies having white alleles in their genetic profile we would see a reversal of this process and black chromosomes would ultimately be bred out of the population. For this to hold true we would require at least one white chromosome left in the butterfly population to enable them to "re-establish" themselves in the population.

If the white chromosome had been totally bred out of the population only the process of mutation would allow the white allele to be reintroduced into the butterfly population.

Therefore the strength of genetic algorithms lies in their ability to combine "good building blocks" with respect to the current environment, in order to allow the strength of the population to increase over subsequent generations by breeding stronger off-spring.

4.4 Learning Classifier Systems

To fully understand the value of classifier systems and their applicability to this particular thesis research area we must look to Bookers *et al.* definition of the particular the type of system that classifiers system were designed to address

They were

i) For environments where a "perpetually novel stream of data constitutes an extremely complex and uncertain problem solving environment"

ii) When a system must "dynamically construct and modify the representation of the problem itself"

iii) For complex environments that "will contain concepts that cannot be specified easily or precisely even with a powerful logic"

(Booker, Goldberg, & Holland, 1989)

Indeed, Holland himself states that "each of the mechanisms used by the classifier system has been designed to enable the system to continue to adapt to its environment, while using its capabilities to respond instant-by-instant to that environment" (Holland J. , 2000).

In 1978 Holland presented the first implementation of a Learning Classifier System (LCS) which was based on his work on genetic algorithms and was a ruled based system. The rules used by the Holland's LCS were in the form of

IF [Condition] THEN [Action]

Therefore when the condition was met then the action was taken by the system. This could be compared with the development of expert systems, which used a similar method but suffered from the requirement to have their knowledge (and therefore rules) captured during the development of the system. Therefore a major issue is that the rules envisaged at the design stage may not capture all of the domain space. This was increasingly problematic as the complexity of systems and their environments increased which may lead to a corresponding change in the domain space.

Learning Classifier Systems offered a potential solution to this issue because they used two main elements:-

- Genetic Algorithms which sought out new rules that were not based on true random generation but rather used a Darwinian "survival of the fittest" approach to suggest likely successful areas of the complex space to search for fitter rule strings. Such natural selection means that *"those organisms best able to acquire limited resources and convert them to offspring will leave the most descendents and the genes controlling their behaviour will increase in frequency"* (Zimmerman, 2009). In terms of our work, such organisms are represented by initial randomised responses (binary strings) to the environment. These are evaluated on their ability to respond successfully to environmental change allowing "stronger strings" to be identified and subsequently "bred" to embed their strength in the population.
- Reinforcement Learning seeks to reward or punish rules depending on their ability to achieve a desired response, such as beneficial change, from the environment. Therefore, rules that are highly fit (in terms of their net worth gained by the reward mechanism) are subsequently given preference over other rules. The reinforced learning would subsequently guide successive targeted searches for "better" rules.

Kovacs saw the interleaving of these two components as driving the population toward a "minimal, fit, non-overlapping population" as a central element of Learning Classifier Systems (Kovacs & Kerber, 2006). Holland suggested that these key mechanisms made possible performance and learning without the characteristic of brittleness associated with most expert systems in AI (Holland J. , 1986).

Designing decision making systems of this nature was often difficult and more and more work began focusing on automated techniques able to "tune efficient decision making systems while requiring less design effort from human experts" (Landau & Sigaud, 2008).

The Classifier System as envisaged by Holland (Holland J. , 1975) is illustrated in Figure 19.

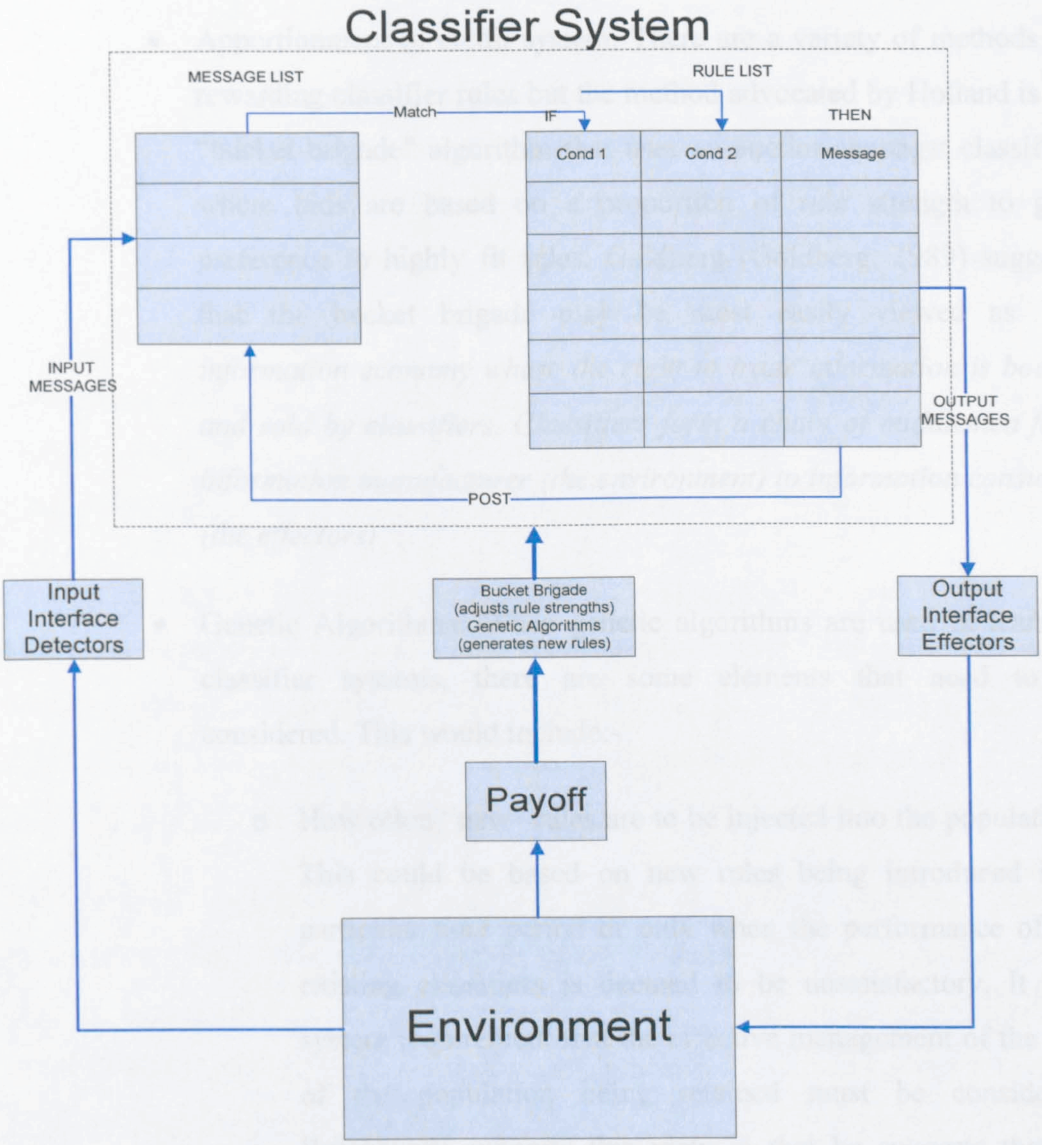

Classifier System
MESSAGE LIST
RULE LIST
Match
IF
THEN
Cond 1
Cond 2
Message
INPUT MESSAGES
OUTPUT MESSAGES
POST
Input Interface Detectors
Bucket Brigade (adjusts rule strengths) Genetic Algorithms (generates new rules)
Output Interface Effectors
Payoff
Environment

Figure 19 Holland's Classifier System

The three main components of a classifier system are:-

- Rule and Message system: Information is received from the environment via detectors that activate rules (or classifiers) to cause an action to be taken in response to environmental stimulus through the system's action triggers (or effectors). Goldberg (Goldberg, 1989) captures the essence of this in that classifiers *"combine environmental cues and internal thoughts to determine what the system should do and think next. In a sense it coordinates the flow of information from where it is sensed to where it is processed to where it is called to an action"*.

- Apportionment of credit system: There are a variety of methods for rewarding classifier rules but the method advocated by Holland is the "bucket brigade" algorithm that uses an auction amongst classifiers where bids are based on a proportion of rule strength to give preference to highly fit rules. Goldberg (Goldberg, 1989) suggests that the bucket brigade may be most easily viewed as *"an information economy where the right to trade information is bought and sold by classifiers. Classifiers form a chain of middlemen from information manufacturer (the environment) to information consumer (the effectors)"*.

- Genetic Algorithms: When genetic algorithms are used in learning classifier systems, there are some elements that need to be considered. This would include:-

    - How often "new" rules are to be injected into the population? This could be based on new rules being introduced in a particular time period or only when the performance of the existing classifiers is deemed to be unsatisfactory. It is a system requirement that the effective management of the size of the population being retained must be considered. Rojanavasu supports this view in that he suggests that *"a large population size has a dramatic impact on the speed for processing incoming data traffic"* (Rojanavasu, Dam, Abbass, Lokan, & Pinngern, 2009) as each time a data instance arrives, the population needs to be scanned to find those rules that match the incoming data instance. Consequently, he suggests that using LCSs for systems requiring a large population size (such as real time stream data mining) is problematic.

- What proportion of the existing population of classifier rules are to be retained? In our earlier discussion on genetic algorithms there was no concept of existing "fit" strings "surviving" into the next generation of the population. However, it seems reasonable to suggest that if we are looking to develop robust systems where we need to maintain a high level of performance then we might want to maintain a proportion of these fit strings in subsequent generations to improve the chance of continued robust performance.

Whilst there were applications of Holland's LCS to "real" problems including gas pipeline control, space vessel power management and letter recognition, interest in LCS waned perhaps due to the complexity of implementing Holland's full system.

One of the earliest "parting of the ways" in LCS research approaches involved the distinction between Michigan LCS and Pittsburgh LCS. The fundamental difference between these two approaches is that in a Michigan LCS, the strongest rule in a population of rules is the solution of the problem, whereas Pittsburgh LCS uses multiples rule-sets where each population rule-set together represent the solution.

Bacardit *et al.* suggested that Michigan style LCS were a more appropriate solution for online learning problems as Pittsburgh LCS *"require a longer evaluation period until the next generation evolve, since the fitness of the whole population rather than of individual classifiers needs to be assessed"* whereas Michigan LCS can usually be *"continuously evaluated and evolved by steady-state GA techniques"* (Bacardit, Mansilla, & Butz, 2008).

Further modifications have been proposed that have sought to increase understanding and performance of learning classifier systems. Wilson continued to work with LCS and proposed the Zeroth-level Classifier System (ZCS), which removed the message list and rule bidding element of Holland's LCS (Wilson S. , 1994). Wilson continued to develop the classifier system and in 1995 presented his eXtended Classifier System (XCS) (Wilson S. W., 1995). The major difference between XCS and earlier classifier systems is that rule fitness is based on the accuracy of its predicted payoff rather than the payoff actually received from the environment and the bucket brigade approach was replaced by a form of Q-learning. There has been research focused on the scalability and learning capacity of XCS systems. Stalph *et al.* suggests that the major factor impacting on learning capacity in XCS systems was in its population size and therefore sought to measure the "minimal population size necessary to solve a learning task" (Stalph, Butz, & Goldberg, 2009) and showed that a linear increase in difficulty results in a polynomial increase in population size.

Bull suggests that only recently is the ability of LCS to solve complex real world problems becoming clear and that *"future work must apply LCS to a wide range of problems and identify characteristics which make the task suitable to solution with Learning Classifier Systems"* (Bull, 2004).

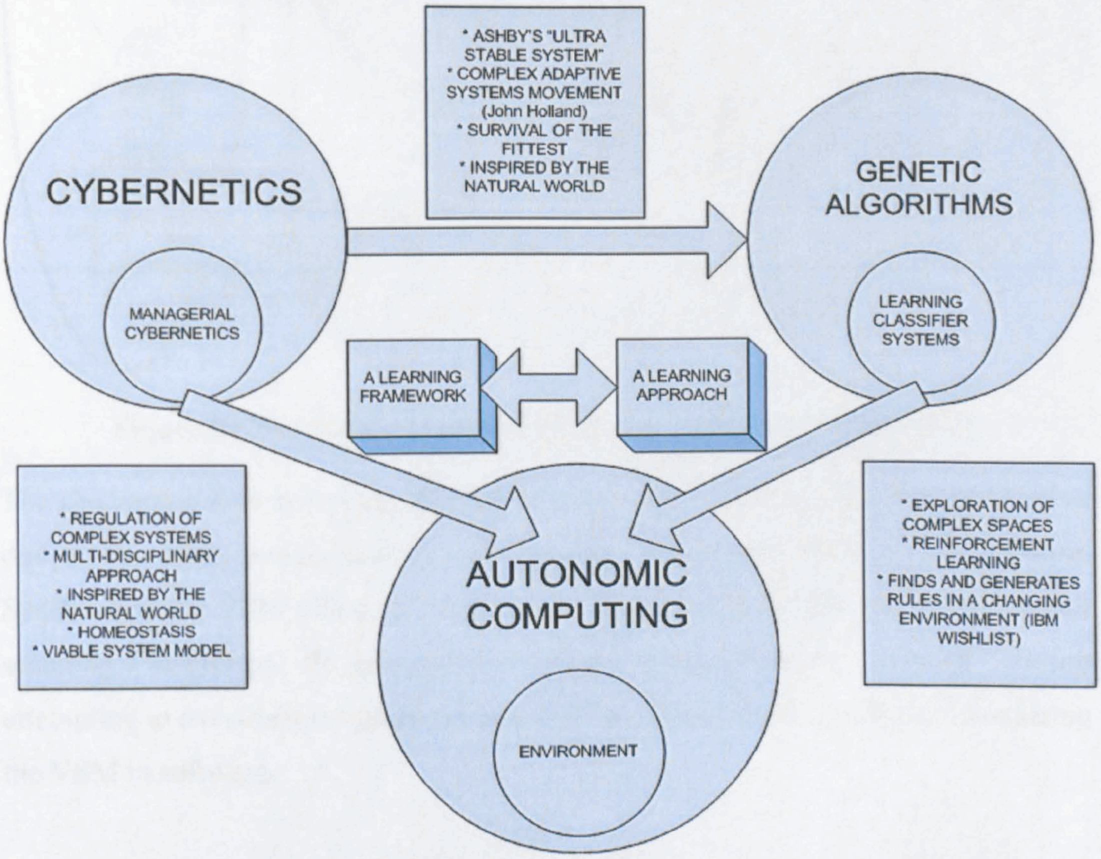
Which form of LCS is most appropriate for which type of problem also needs to be established, along with continued refinement of the architectures and improved theoretical understanding. Hurst and Bull amongst others have continued to experiment on applying Learning Classifier Systems in problem domains such as robotic environments (Hurst & Bull, 2006) whilst the "father of LCS" John Holland has also continued to work on practical uses for learning classifiers (Holland J. , 2003).

Our work suggests that Learning Classifier Systems have a potentially important role to play in realising the goal of autonomic software systems that can respond to environment stimuli in an appropriate manner. This is achieved using genetic algorithms to explore the complex space and the utilisation of reinforced learning to allow for evaluation of the rules of "most promise" to be identified. The rule set generated by this process may well provide the most appropriate adaptive homomorphic model of both the external visible environment as well as a current model of internal capability which allows our system to continue to operate robustly, by making decisions based on appropriate models, in complex and turbulent environments.

4.5 Prototype Proposal

The prototype will seek to use cybernetics and more specifically Beer's Viable System Model as a "learning framework" to provide the heuristics for the overall system design. Learning Classifier Systems and Genetic Algorithms will be used to provide a reinforcement learning mechanism approach in which to realise aspects of System 4 of the VSM in software.

Figure 20 illustrates the some of the theoretical links between cybernetics and Genetic Algorithms / Learning Classifier Systems. The obvious links include inspiration from the natural world and the regulation of complex systems. It is proposed that this application of a learning framework (provided by the VSM) and a learning approach (provided by GA/LCS) provides a potential solution to the challenge of providing environmental modelling during run-time.



• ASHBY'S "ULTRA STABLE SYSTEM"
• COMPLEX ADAPTIVE SYSTEMS MOVEMENT (John Holland)
• SURVIVAL OF THE FITTEST
• INSPIRED BY THE NATURAL WORLD
CYBERNETICS
MANAGERIAL CYBERNETICS
GENETIC ALGORITHMS
LEARNING CLASSIFIER SYSTEMS
A LEARNING FRAMEWORK
A LEARNING APPROACH
• REGULATION OF COMPLEX SYSTEMS
• MULTI-DISCIPLINARY APPROACH
• INSPIRED BY THE NATURAL WORLD
• HOMEOSTASIS
• VIABLE SYSTEM MODEL
AUTONOMIC COMPUTING
ENVIRONMENT
• EXPLORATION OF COMPLEX SPACES
• REINFORCEMENT LEARNING
• FINDS AND GENERATES RULES IN A CHANGING ENVIRONMENT (IBM WISHLIST)

Figure 20 Theoretical Underpinning for the Prototype System

Figure 21 illustrates why the VSM may provide a broader learning framework, from which autonomic systems may be developed in the future. The IBM wish-list can potentially be mapped across to the VSM. This thesis is primarily concerned with System 4 (environmental awareness) but this should not distract from the further value potentially held by releasing the full VSM in software.

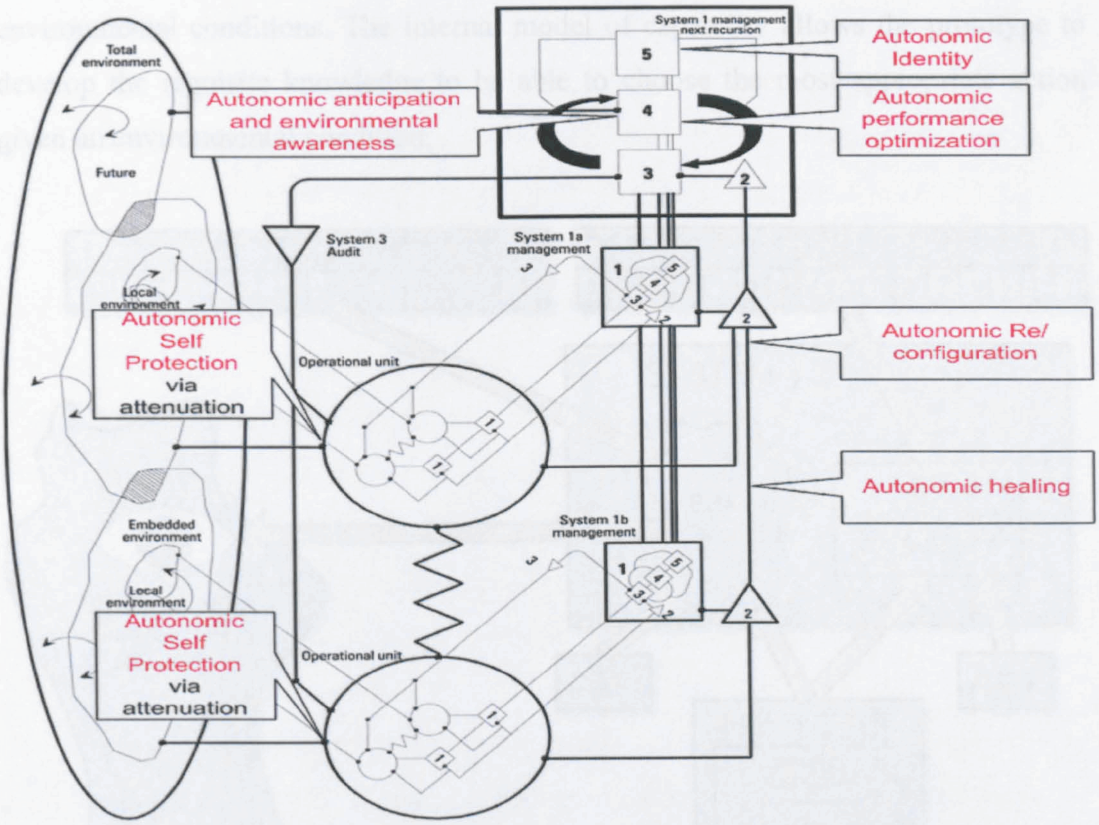

System 1 management next recursion
5
4
3
2
Autonomic Identity
Autonomic performance optimization
Total environment
Autonomic anticipation and environmental awareness
Future
System 3 Audit
System 1a management
1
Local environment
Autonomic Self Protection via attenuation
Operational unit
Autonomic Re/ configuration
Autonomic healing
Embedded environment
System 1b management
Local environment
Autonomic Self Protection via attenuation
Operational unit

Figure 21 The Viable System Model and Autonomic Computing

The challenges with unlocking this potential are many, but this thesis seeks to solve one of the main issues, namely, how to replicate human creativity in software. System 4 of the VSM relies on human creativity to achieve effective environmental scanning / awareness. By using GA / LCS to emulate "human creativity" we are attempting to overcome a significant barrier in the considerable challenge of realising the VSM in software.

This will enable further development of autonomic systems based on the VSM to be developed and implemented in the future.

Figure 22 illustrates in further detail how the prototype will seek to use GA / LCS to realise System 4 of the VSM.

System 4 relies on two models to achieve environmental awareness. These models consist of the external model of the environment and an internal model of capability. The external model of the environment allows the prototype to develop the requisite variety required, in cybernetic terms, to be able provide a response to varying environmental conditions. The internal model of capability allows the prototype to develop the requisite knowledge to be able to choose the most appropriate action given an environmental condition.

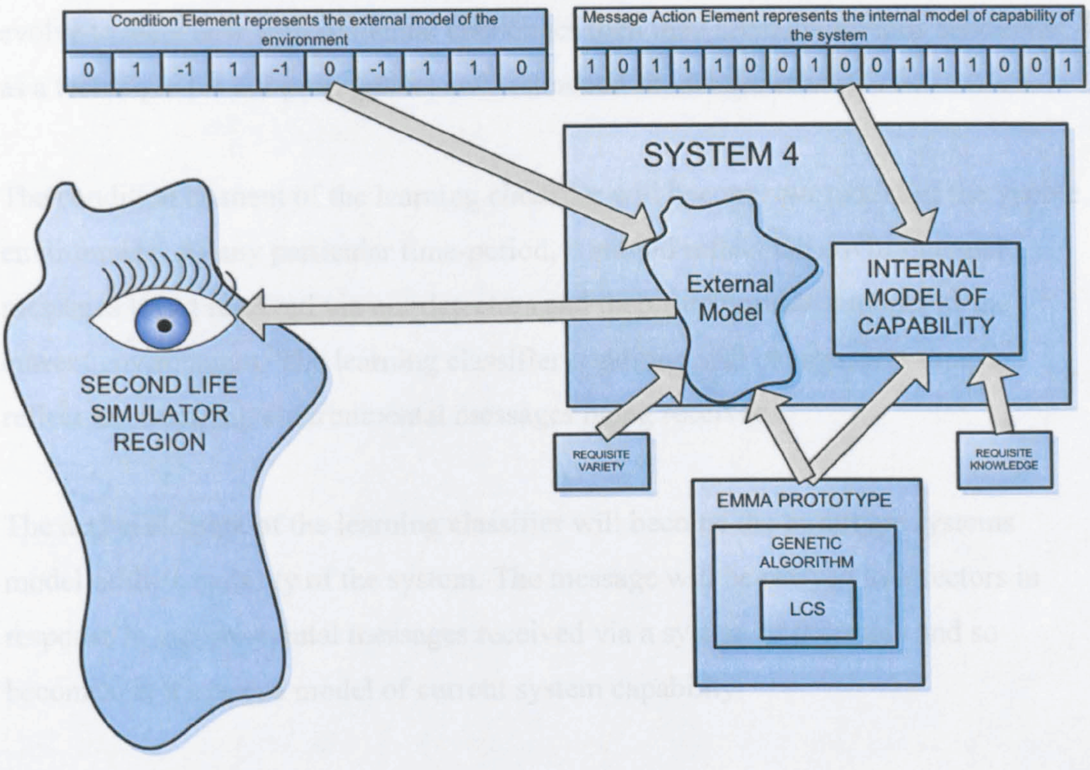

Condition Element represents the external model of the environment
0 | 1 | -1 | 1 | -1 | 0 | -1 | 1 | 1 | 0
Message Action Element represents the internal model of capability of the system
1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1
SYSTEM 4
External Model
INTERNAL MODEL OF CAPABILITY
SECOND LIFE SIMULATOR REGION
REQUISITE VARIETY
REQUISITE KNOWLEDGE
EMMA PROTOTYPE
GENETIC ALGORITHM
LCS

Figure 22 The Prototype System, VSM's System 4 and GA/LCS

The condition element of the learning classifier will represent the external model of the environment and the message action element will represent the internal model of capability of the system. The ability to evolve these models will be provided by the genetic algorithm which will enable new conditions and actions to be generated to maintain the currency of these models.

4.6 Summary

The prototype system will be developed using Learning Classifier Systems as a fundamental guiding design principle. It is not the only potential solution available but matched the autonomic principles advocated by cybernetic theory. It contains many of the building blocks required, such as reinforcement learning as a fundamental underpinning, and the obvious biological and Darwinian principles of survival of the fittest.

If we accept that genetic algorithms are extremely efficient at exploring complex search spaces and have the ability through crossover and mutation to continuously evolve to meet new environmental challenges then they seem an obvious candidate as a technique for our prototypical environmental model system.

The condition element of the learning classifier will become our model of the visible environment. At any particular time-period, it should reflect the environmental messages being received via our detectors and therefore provide a model of the current environment. The learning classifier condition will change over time to reflect the changing environmental messages being received.

The action element of the learning classifier will become the prototype systems model of the capability of the system. The message will be relayed to effectors in response to environmental messages received via a system of detectors and so becomes, in a sense, a model of current system capability.

This combination of models means that the prototype will perform some of the functionality of system 4 of the Viable System Model, a key component in maintaining the robustness of the system and therefore helping to realise the potential of autonomic software. There are a number of development challenges remaining including the requirement for an appropriate environmental application to be used and modelled by the prototype system. The next chapter outlines the application used by the prototype system and discusses its particular suitability.

Chapter 5: Design & Evaluation of Initial Experimental Approaches to Controlling and Modelling the Environment

"We Make The World We Live In And Shape Our Own Environment"
Orison Swett Marden

5.1 The Experimental Platform

Virtual worlds in computing are not a particularly new concept as some form of these "worlds" have existed since the late 1970s with the development of the first text based virtual worlds called Multi-User Dungeons (or MUDs). Such worlds were often based on the Dungeons and Dragons gaming movement, which was popular during this period.

Sanchez (Sanchez, 2009) suggests that whilst many breakthroughs have occurred since 1979, five milestones can be used to summarise the history of virtual worlds:

- Multi-User Dungeons (MUDs)
- TinyMUDs
- MOOs (Multi-User Dungeons Object Orientated)
- Massively Multi-Player Online Role-Playing Games (MMORPGs)
- 3-D social Virtual Worlds

Due to the quantum leap in computing technology over the period (including processor capacity, graphics capability and broadband internetworking), virtual worlds have developed from relatively simplistic text based adventure games into highly complex environments that allow new levels of freedom and opportunity for development. One major change with the growth of 3-D social Virtual Worlds such as Second Life is that there is no inherent game or quest as in MMPORPGs such as World of Warcraft and Sony's Everquest. These virtual worlds are increasingly seen as the next phase of the Internet or "the new web" (Bell, Pope, Peters, & Galik, 2007).

Despite the plethora of virtual worlds that now exist, Smart *et al.* (Smart, Cascio, & Paffenduff, 2007) suggest that there are features that are shared between such worlds including:

- A shared space allowing multiple users to participate simultaneously
- Virtual embodiment in the form of an avatar (a 3D representation of the self)
- Interactions that occur between users and objects in a 3-D environment
- An immediacy of action such that interactions occur in real time
- Similarities to the real world such as topography, movement and physics that provide the illusion of being there.

The most popular of these 3D Social Virtual Worlds is currently Second Life. Second Life was developed by Linden Laboratories and launched in June 2003 and now has over 16 million registered users. Nearly everything seen within the Second Life universe is created by users rather than by software developers at Linden Labs. Second Life has been described as having "the most powerful object creation toolset of any 3D MUVE" (Salmon, 2009), which may account for its leading role in terms of both numbers of users and innovative use by developers. This means that users can to a large extent create and modify their own virtual environment unconstrained by the laws of nature and physics. This offers a potentially valuable environment for relatively complex simulations to be modelled more efficiently. Foss (Foss, 2009) suggests that this offers a number of advantages for using virtual environments in this manner, including:

- Simulation of an expensive real world resource (such as a virtual set for a film studio)
- Collaborative tasks (such as group meetings eliminating the requirement for physical travel)
- Risk assessment (assessing risks in a virtual world prior to the real event)

From a technical viewpoint, a client viewer running on a user PC is connected to Linden Labs servers to access and use the Second Life simulation. This "server array" now consists of several thousand servers as the hardware supporting the virtual world has required rapid expansion to service the rapid increase in users. There are a number of software servers used to provide basic services such as login, instant messaging, agent database, central database, the rendering of the overall world map etc but we are primarily concerned with the simulator service provided by the virtual world. The interactions between the varying servers used to provide the services required by Second Life are outlined in Figure 23.

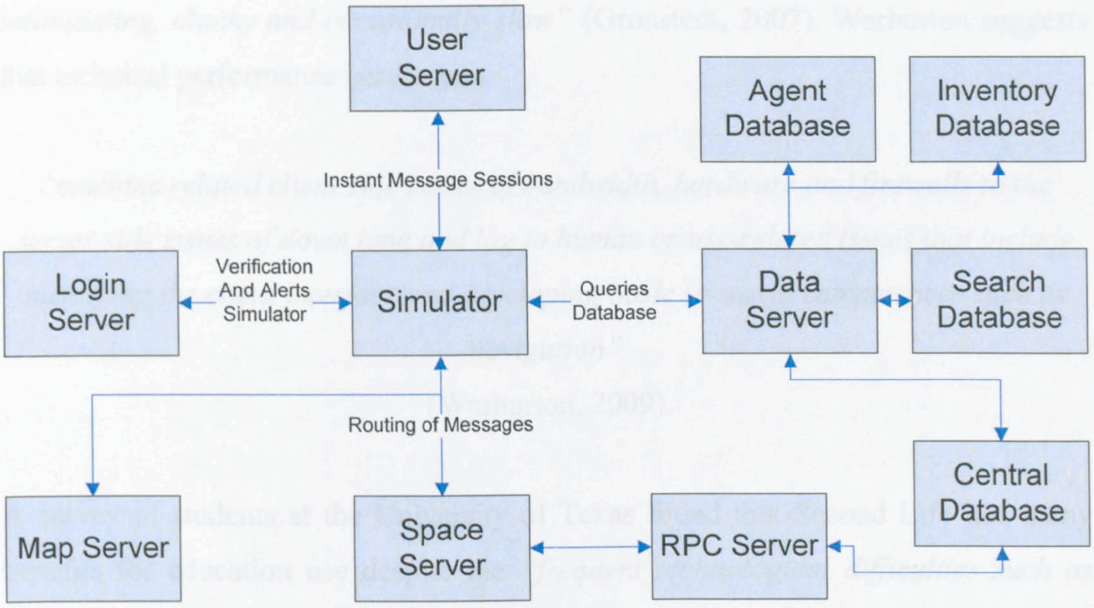


Figure 23 Second Life Server Services

Virtual land in Second Life consists of regions that can be purchased by residents (users) and subsequently used for their own purposes such as e-commerce, educational use, private use etc. with the owner having considerable control over their region. A region consists of 65536m² (256m x 256m) of virtual land which equates to approximately 16 acres.

The simulator provides a range of services to each of these regions including sending data to the clients of avatars within the region including textures and sounds, handling the physics engine of the region, networking requirements such as communication with adjacent simulators and the running of scripts (using Linden Scripting Language) within the region. Each simulator runs on a single core of a multi-core server with proprietary software on Debian Linux. As a user moves from one region to another within the virtual world they become part of the processing load of the simulator handling that particular region.

One of the main issues facing virtual worlds such as Second Life has been in the performance of the simulators (or regions) and the subsequent impact on the experience of the users. Grondstedt suggested that whilst Second Life is *"mesmerizing, immersive, and inherently social, it's also perplexing and intimidating, clunky and occasionally slow"* (Gronstedt, 2007). Warburton suggests that technical performance issues span

> *"machine related client side issues of bandwidth, hardware and firewalls to the server side issues of down time and lag to human or use-related issues that include managing the client interface and developing basic in-world competences such as navigation"*
>
> (Warburton, 2009).

A survey of students at the University of Texas found that Second Life had many benefits for education use despite the *"frequent technological difficulties such as glitches, lags, frequent updates and crashes"* (Jarmon, Traphagan, Mayrath, & Trivendi, 2009).

If virtual worlds such as Second Life are to provide a genuine 3D internet experience then this virtual experience needs to be *"substantially better than existing web capabilities"* (Anthes, 2007) in order to make it worthwhile. One of the key benefits of owning land in the Second Life grid is the opportunity to create new and exciting uses of the technology, for example, innovative uses in education and training as diverse as creating and studying 3D fractals (Bourke, 2009) to the use of virtual worlds in order to provide psychologists with a *"multitude of new and exciting opportunities"* (Jarrett, 2009).

Developing such virtual world applications requires the maintenance of a high quality of service level by the region simulator if they are to be used to their full potential. If the Gartner Inc. prediction that 80% of all active Internet users will have an avatar and would be registered in one or more virtual worlds becomes reality (Gartner, 2007) then increasing "pressure" on the hardware and software systems providing the technical backbone to these applications can be expected. At this point, it is useful to consider, in more technical detail, how a region in controlled by the simulator.

5.2 The technical aspects of performance on a virtual world simulator

The first component is the simulator frame rate (often described as Sim FPS). The simulator frame rates equate to the number of times per second that the simulator is able to process all the components (scripts, collision detection, updates etc) that are required within the virtual world. A Second Life region running at optimum performance seeks to maintain a frame rate of 45 Frames Per Second (FPS). If this is to be achieved the calculation required to process each frame "time slice" must not take more than approximately 22.22 milliseconds (ms). This figure is calculated by dividing one second by 45 (frame time slices) which equals 0.2222222222 ms.

This means that for a simulator to achieve 45 frames per second the sum of the calculations and processing required must be less than 22.22 ms.

If a simulator is required to process 40 milliseconds of "load" then it will only be able to achieve 25 fps. If the simulator is effectively only processing 25 fps (in terms of updating physics, textures etc) then the users on the region will experience a slowdown (often described as lag) where their avatar will appear not to respond effectively in terms of movement etc which significantly diminishes their experience of the region. A key indicator used to measure simulator performance is dilation which represents the performance of the physics element of the region as a figure between 0 and 1. A time dilation of 1.00 means that the simulator is performing at optimum level in processing all of its various functions. If dilation is equal to 0.5 it would mean that the simulator was only able to perform at 50% of its optimum performance level (this would indicate that the frame time was probably at a level of 44.4ms which is twice what the simulator can handle and still perform at the optimum level).

One method used by Second Life to attempt to maintain its performance at 45 frames per second is throttling the time allocated to the scripts running within a region.

Scripted objects often take a significant portion of the 22.22ms frame time element. Therefore, the simulator will try to maintain the other elements required by the simulator at the expense of scripts. While this approach does help to keep basic functions at an appropriate level it is the scripted elements within a region that provide much of the functionality available to users and therefore there is a requirement to manage this effectively to provide robust quality of service while maximising script functionality.

In terms of scripted objects in Second Life (using Linden Scripting Language), there are two main elements to consider, namely the quantity and efficiency of the scripts being used in the region. However, in terms of efficiency it may be appropriate to actively manage scripts (rather than purely using throttling) to provide a more appropriate and rich experience to users.

Below is a script example of an "intelligent" light in Second Life provided to show the syntax of the language. This script would be embedded within an object (acting as a light) within the region but it should be remembered that all scripts have a "cost", which would have a potential implication for a region if the script is particularly "expensive" in terms of the execution time. Therefore, a monitor that is "scanning" some element of the environment every one tenth of a second is more expensive than an environmental scan which occurs every five seconds.

This script uses a proximity monitor script (see script example 1). The script runs every second and detects any avatars within a radius of 3 metres. If the proximity monitor senses an avatar within a specified metre range it broadcasts "light On" across the region on a communication channel which in this instance is channel 5.

Script Example 1 (Monitor Script)

```
default
{
state_entry()
    {
    llSetTimerEvent(1.0); //Repeat every 1.0 second
    }
timer()
```

```
    {
    llSensor("", NULL_KEY, AGENT, 3, PI);
    }
sensor(integer total_number)
    {
    llRegionSay(5, "light on");
    }
no_sensor()
    {
    llRegionSay (5. "light off");
    }
}
```

Figure 24 Example LSL Code

Any intelligent light (or any other object within the region) “listening“ on channel 5 will be able to detect a message broadcast on that channel and act accordingly. In this example if “light on” is detected then the script is instructed to switch the light on (light parameter of object is set to TRUE) . Once an avatar moves out of range then a “light off” message will be broadcast region-wide and all lights will be switched off (light parameter of the object is set to FALSE).

Script Example 2 (Intelligent Light Script)

```
default
{
    state_entry()
    {
    // listen on channel five for any chat spoken by
    the object owner.
    llListen(5,"",NULL_KEY,"");
    }

    listen(integer channel, string name, key id, string
    message)
    {
    // check if the message corresponds to a predefined
    string.
    // llToLower converts the message to lowercase.
    // This way, "LIGHT ON", "Light on" or "LiGhT On"
    will all work the same way.
    if (llToLower(message) == "light on")
        {
llSetPrimitiveParams([PRIM_POINT_LIGHT,TRUE,<2.0,0.7,1.0>
,1.0,10.0,0.6]);
        }
        if (llToLower(message) == "light off")
        {
llSetPrimitiveParams([PRIM_FULLBRIGHT,ALL_SIDES,FALSE]);
            llSetPrimitiveParams([PRIM_POINT_LIGHT,
FALSE,<0.0,1.0,0.0>,1.0, 10.0, 0.5]);
        }
    }
}
```

Figure 25 Example LSL Code

While the scripting language provided with Second Life provides the basis for much of the content and functionality provided within regions, it also has the potential to create significant performance issues for simulators running regions.

There is a plethora of virtual world simulator environments currently being developed and whilst the social networking elements of these systems are the primary focus, they also provide interesting platforms for research purposes in a number of areas.

The "Second Life" platform is an ideal test platform for the environmental modelling prototype developed in this research for a number of reasons, including:

i) The flexibility and functionality of Linden Scripting Language (LSL) that allows for a system of effectors and detectors to be developed for experimental purposes.

ii) A wide range of controls over the simulator region that allows the environment to be closed or relatively open, which will prove useful in terms of the experiments.

iii) A range of communication channels including HTTP and XML-RPC that enables the system to communicate with the prototype systems in terms of reporting metric data and communicating actions into the region.

iv) The relative ease with which virtually every aspect of a region can be metricated.

v) The ability to be able to "affect" performance in a controlled manner i.e. to easily determine the effect of actions on performance.

vi) The dynamic nature of the region i.e. scripts beginning / ending and avatar action, therefore giving a noisy and unpredictable environment.

The choice of the virtual world simulator has proved illuminating in terms of the broader requirements of tethering software systems to the real-world, or indeed the virtual world, and has illustrated some of the pre-requisite features related to communication channels, usable metrics and the utilisation of effectors and detectors.

Before the development of the prototype it seemed appropriate to develop a series of experiments based purely in the virtual world to provide some benchmark metrics. This would illustrate the strengths and weaknesses of a variety of approaches related to managing environmental change using randomised approaches, static models, self-management and hybrid learning models. The rationale behind and the results of these initial experiments are discussed in the following section of the thesis.

5.3 Description and Rationale of the initial experiments

The support for the development of a prototype using a method of real-time and continuous learning seemed compelling but if we were to critically evaluate the performance of such a system it seemed appropriate to develop a suite of alternative experiments with which we could compare and contrast our results. LSL provided sufficient functionality to design and evaluate a number of alternative approaches to managing environmental performance within the simulator. The rationale, design and evaluation of these experiments are presented within this chapter.

5.3.1 The Scripted Objects

The scripted objects are represented as a simple primitive object on the simulator region. They were given object numbers in order to allow easy identification and for messaging purposes. They contain two main elements:-

i) The Listener Script. The function of this script was to scan the environment for messages significant to that particular object and take appropriate action in response to any such message. The messages to which the objects could respond were

   a. Stop All. This was a broadcast message that stopped all of the scripted objects together. It was typically used by the system when establishing a baseline measurement of performance.

   b. Start All. This was a broadcast message that started all of the scripted objects together. It was typically used by the system when establishing a maximum measurement of performance.

   c. Start [Object Number]. This was an object specific message to start all scripts within that particular scripted object. For example "Start 1" would start all of the additional scripts within scripted object 1.

   d. Stop [Object Number]. The stop object command would stop all of the additional scripts within the scripted object.

   It should be noted that it was necessary to leave the listener script running at all times within the object but it was the only script which could not be stopped from running.

ii) A number of additional scripts in each scripted object, each of which contained LSL code of a simple infinite While loop. It was found that this was one of the simplest and effective ways of imposing a significant computation load on the simulator region.

When a scripted object is running all additional scripts, the object is Green whereas when scripts were disabled in that particular object it was Red. This can be observed in Figure 26.

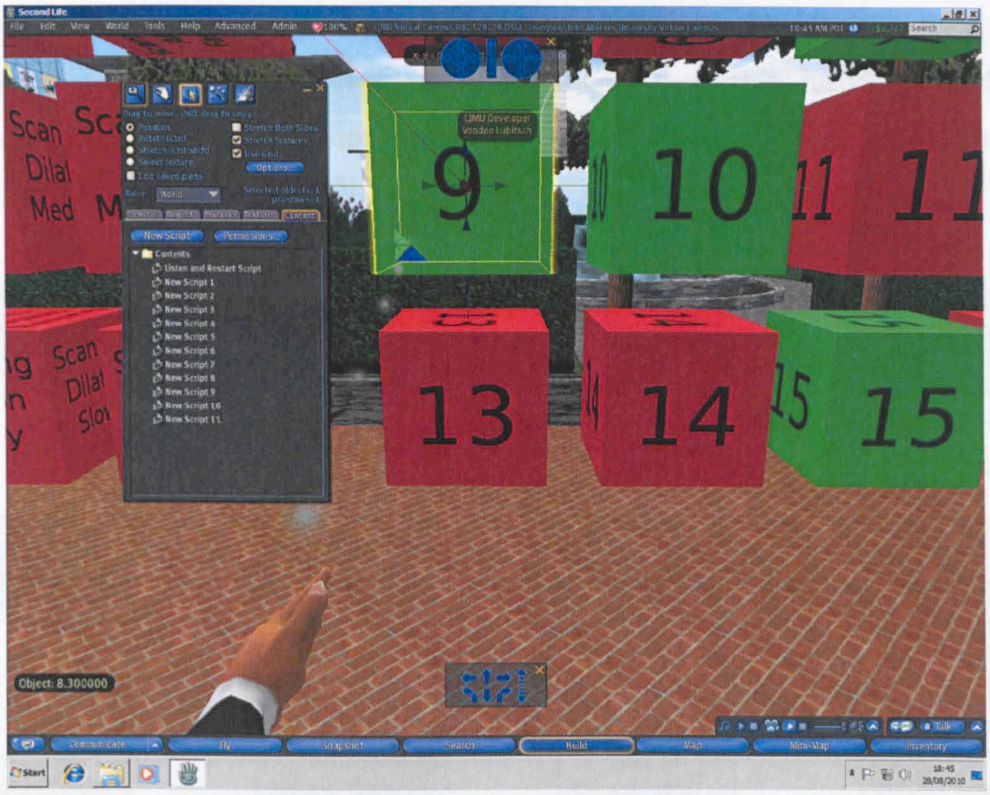
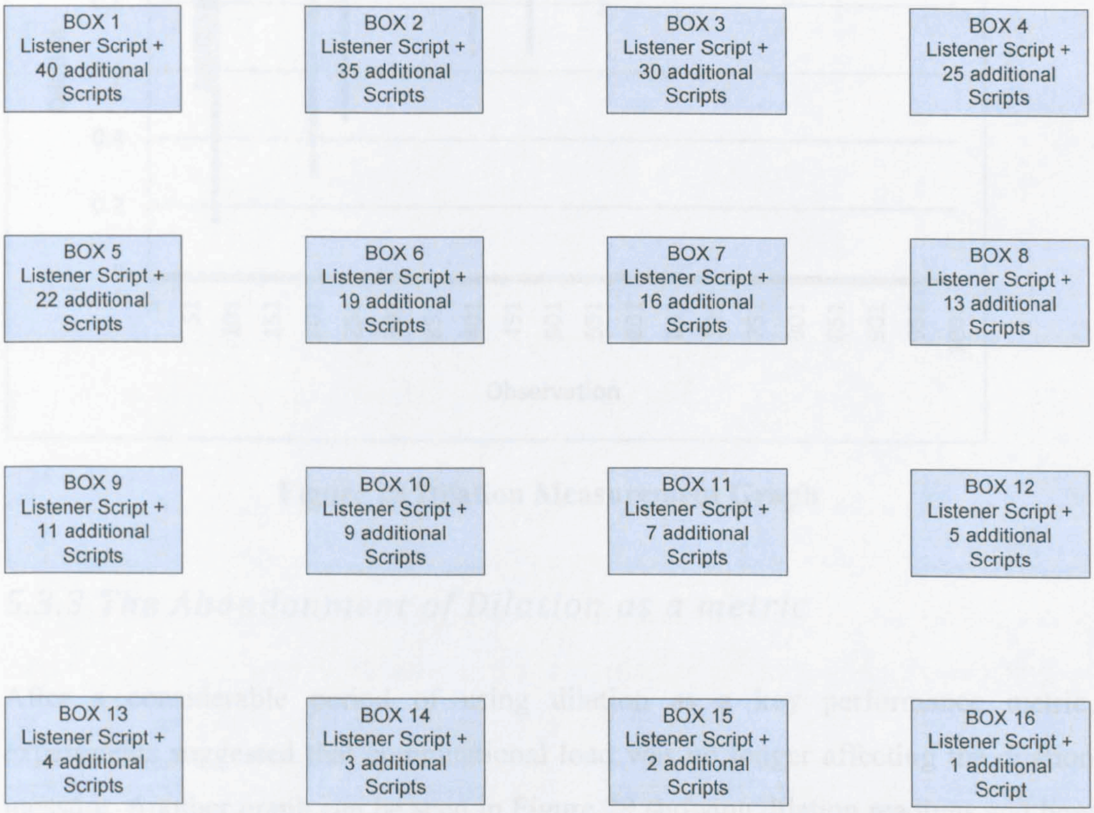


Figure 26 Scripted Objects on the Simulator

The scripted objects and the number of additional scripts contained within them can be seen in Figure 27. Box 1 (with 40 additional scripts) induced the highest computational load on the simulator region whereas Box 16 (with just a single additional script) was the lowest.



BOX 1 Listener Script + 40 additional Scripts	BOX 2 Listener Script + 35 additional Scripts	BOX 3 Listener Script + 30 additional Scripts	BOX 4 Listener Script + 25 additional Scripts
BOX 5 Listener Script + 22 additional Scripts	BOX 6 Listener Script + 19 additional Scripts	BOX 7 Listener Script + 16 additional Scripts	BOX 8 Listener Script + 13 additional Scripts
BOX 9 Listener Script + 11 additional Scripts	BOX 10 Listener Script + 9 additional Scripts	BOX 11 Listener Script + 7 additional Scripts	BOX 12 Listener Script + 5 additional Scripts
BOX 13 Listener Script + 4 additional Scripts	BOX 14 Listener Script + 3 additional Scripts	BOX 15 Listener Script + 2 additional Scripts	BOX 16 Listener Script + 1 additional Script

Figure 27 The Contents of the 16 Scripted Objects

5.3.2 Boundary Counts, Dilation Metrics and FTime

The dilation measure is the physics simulation rate relative to real-time where 1.0 means that the simulator is running at full speed. In seeking an appropriate performance metric that would allow the system to measure the beneficial or detrimental effect of its action then dilation was original envisaged to be the key performance metric for this purpose.

Below is a graph showing performance variability as measured by the dilation metric. It can be seen that as increased "computational load" is placed on the simulator then this is reflected in the dilation metric. Therefore, it seemed highly appropriate to use dilation and many of the early experimentation were based on this.

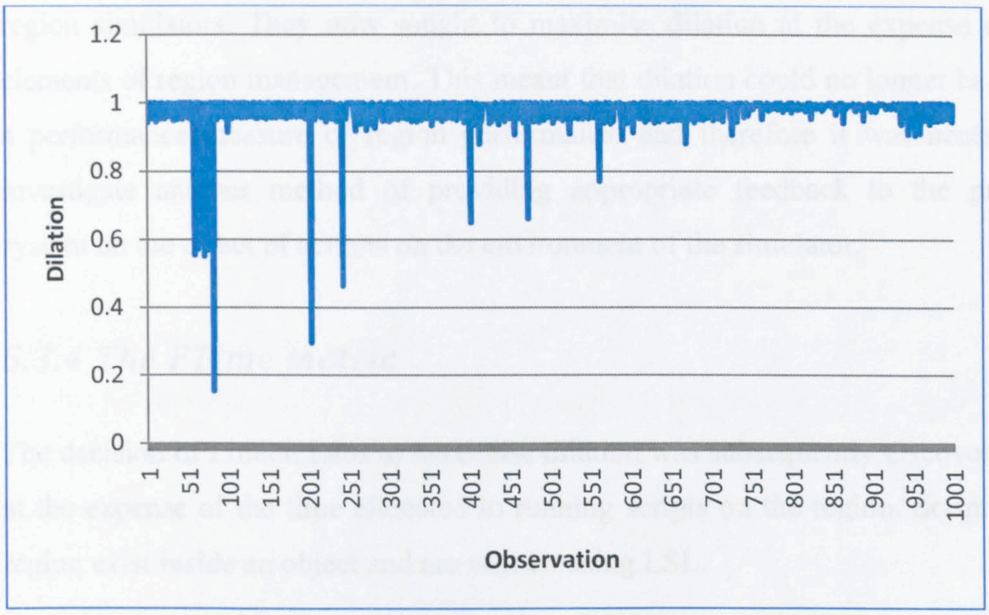


Figure 28 Dilation Measurement Graph

5.3.3 *The Abandonment of Dilation as a metric*

After a considerable period of using dilation as a key performance metric, experiments suggested that computational load was no longer affecting the dilation measure. Another graph can be seen in Figure 29 showing dilation readings and how stable they had now become even when extreme loading was placed on the region.

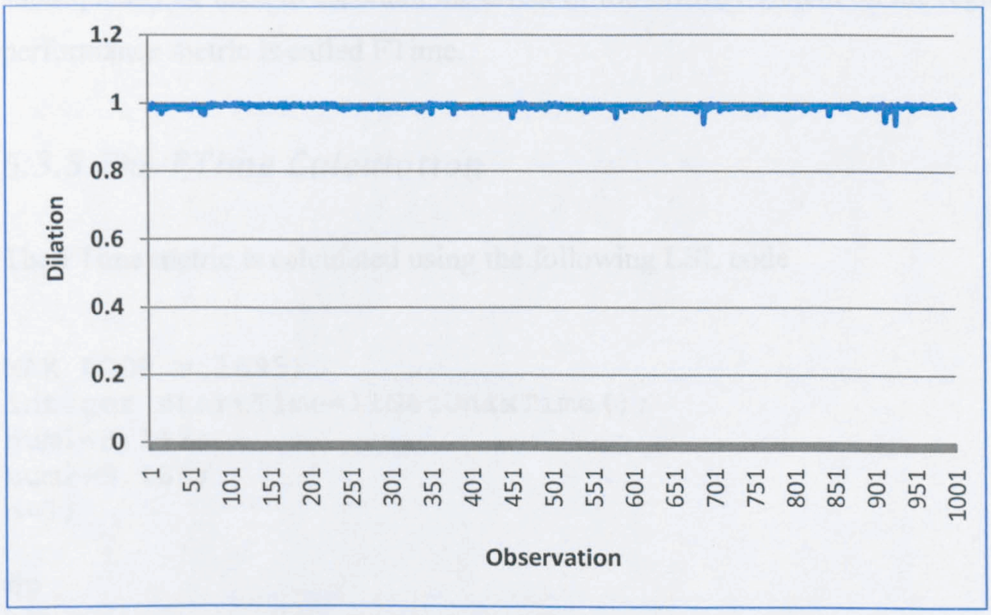


Figure 29 Dilation Measurement Graph After Change

An investigation of this issue showed that Linden Labs (developers of the Second Life platform) had changed the way servers handled computational load on their region simulators. They now sought to maximise dilation at the expense of other elements of region management. This meant that dilation could no longer be used as a performance measure of region performance and therefore it was necessary to investigate another method of providing appropriate feedback to the prototype system on the effect of actions on the environment of the simulator.

5.3.4 The FTime metric

The decision of Linden Labs to maximise dilation was subsequently discovered to be at the expense of the time allocated to running scripts on the region. Scripts on the region exist inside an object and are written using LSL.

All scripts running on the region have an associated computational overhead. Therefore, if the time allocated to scripts were throttled as a consequence of Linden Labs decision to maximise dilation we would expect to observe that scripts would subsequently take longer to perform their particular actions when the region was beginning to suffer from performance problems.

This provided an opportunity to develop an alternative performance metric that could subsequently be used to ascertain the effect of the EMMA system on the region. This performance metric is called FTime.

5.3.5 The FTime Calculation

The FTime metric is calculated using the following LSL code

```
MAX_LOOP = 1695;
integer startTime=llGetUnixTime();
num1=3.142;
num2=9.189;
n=1;

do
      {
      num3=num1*num2;
```

```
    num4=num3/num1;
    n=n+1;
    }
while(n<=(10*MAX_LOOP));

integer endTime=llGetUnixTime();
integer timeTaken=endTime-startTime;
ftime=timeTaken/10.0;
```

Figure 30 Calculation of the FTime Metric

The rationale behind the use of the FTime calculation is that when scripts on a region are being throttled, the time taken to perform the calculation required in the script will be increased. This can be measured by using llGetUnixTime() to measure the start and end time of the calculation and allowing the calculation of the difference between the two times. One limitation of this method is that llGetUnixTime() only measures to the nearest second. Therefore, it would be necessary to run the FTime calculation over a longer period to gain meaningful results in terms of accuracy.

Linden Scripting Language does have its own time measurement function (llGetTime) that would enable more precise measurements, but these could not be used as they themselves were subject to being throttled and therefore render any subsequent timing results as unreliable. The llGetUnixTime() function was not affected by this factor as it took the time from the server and therefore was much more appropriate as an accurate measure despite the limitations outlined above.

An average run of the script on the simulator without any artificially induced computational load can be seen in Figure 31. We were able to induce load that could increase this metric and therefore give an indication of the effect of a particular single or accumulative actions on the environment of the simulator.

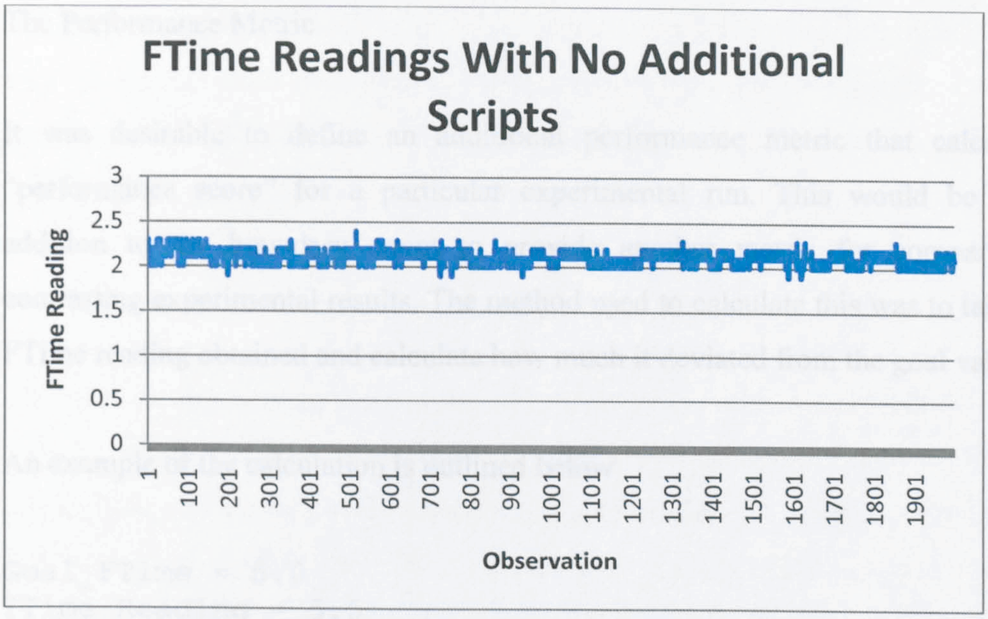


Figure 31 Background FTime Readings

A method for comparing and contrasting the relative performance of the various experiments was required, so the following metrics were used on the simulator experiments.

Boundary Counts

Each of the experiments kept a total of the FTime readings within specific performance boundaries (Figure 32). This would allow analysis within these performance boundaries

Boundary 1	Within plus or minus 0.4 from the goal FTime value
Boundary 2	Within plus or minus 0.8 from the goal FTime value
Boundary 3	Within plus or minus 1.2 from the goal FTime value
Boundary 4	Within plus or minus 1.6 from the goal FTime value
Boundary 5	Within plus or minus 2.0 from the goal FTime value
Out of Bounds (OoB)	Greater than plus or minus 2.0 from the goal FTime value

Figure 32 The Values of the Experimental Boundaries

The Performance Metric

It was desirable to define an additional performance metric that calculated a "performance score" for a particular experimental run. This would be used in addition to the boundary count to provide another metric for comparing and contrasting experimental results. The method used to calculate this was to take every FTime reading obtained and calculate how much it deviated from the goal value.

An example of the calculation is outlined below

```
Goal_FTime = 5.0
FTime_Reading = 5.5
Performance = (FTime Reading - Goal FTime) * 10 = (0.5 *
10) = 50
```

This performance metric was totalled at the end of each experiment to give an overall performance. A lower score would indicate better performance in terms of obtaining FTime readings closer to the target goal.

5.4 The Random Experiment

Given the fact that Learning Classifier Systems and Genetic Algorithms have their fundamental basis in random searches of the space for potential solutions it seemed reasonable to begin our experiments with an approach using a purely random search. There would be no learning element involved in this approach and this would be a purely unguided search of the space.

This experiment would initially calculate a goal FTime, using the mid-point between the baseline and maximum readings, and attempt to experiment with random combinations of the scripted boxes to reach and then maintain that goal metric.

If the FTime reading was not equal to the goal that a random re-configuration of the scripted objects would taken place to try and reach the goal value.

The random model contained no prior experimental learning, nor was there any concept on "on-line" learning. It was purely an attempt to ascertain benchmark performance metrics for other future experiments. The outline design used by the experiment can be observed in Appendix A, Figure 137.

5.4.1 Results from the Random Experiments

The experiments were run as a series of individual experiments to limit the effect of random environmental noise on any one particular experiment. Figure 33 shows the results from a single run of the random experiment. There were 100 observations per experiment and the graph reflects the distribution of FTime readings in relation to the boundaries discussed earlier in the chapter. As shown in Figure 33, OofB indicates readings "Out of Bounds" which would equate to a margin of +- 2.0 away from the FTime goal requested.

The distribution towards the goal (boundary 1) reflects the fact that the goal has been set at the mid-point between the minimum and maximum readings. The randomised nature of the experiment means that on average, 8 boxes would be switched on and 8 boxes switched off for each iteration of the experiment and therefore hit the mid-point more frequently than other boundaries.

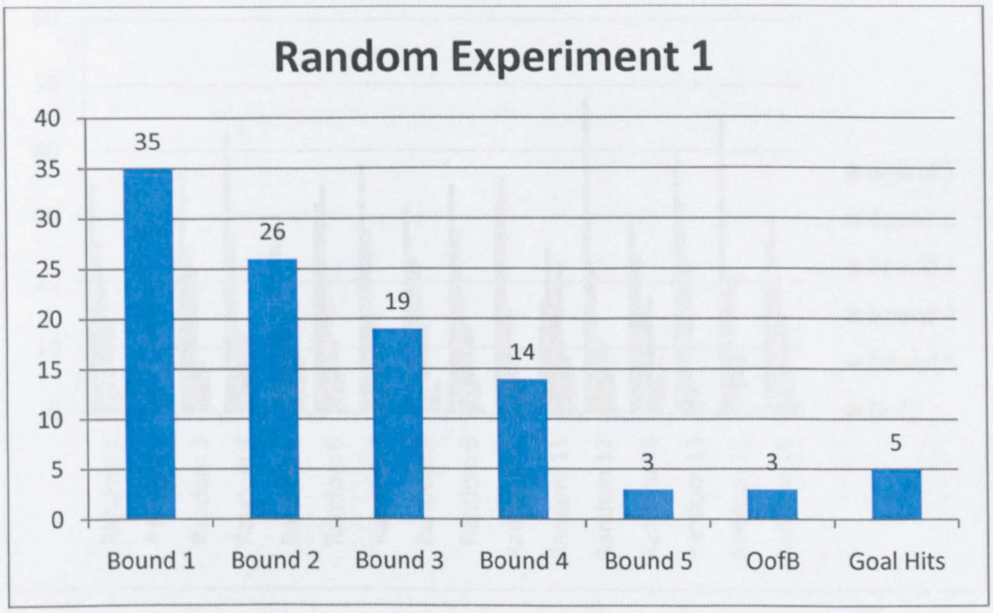


Figure 33 The Results of Random Experiment 1

Figure 34 shows the results obtained from random experiments 1 - 7. The overall results show that the results of Random Experiment 1 have been broadly replicated in subsequent experiments in terms of distribution in the defined boundaries.

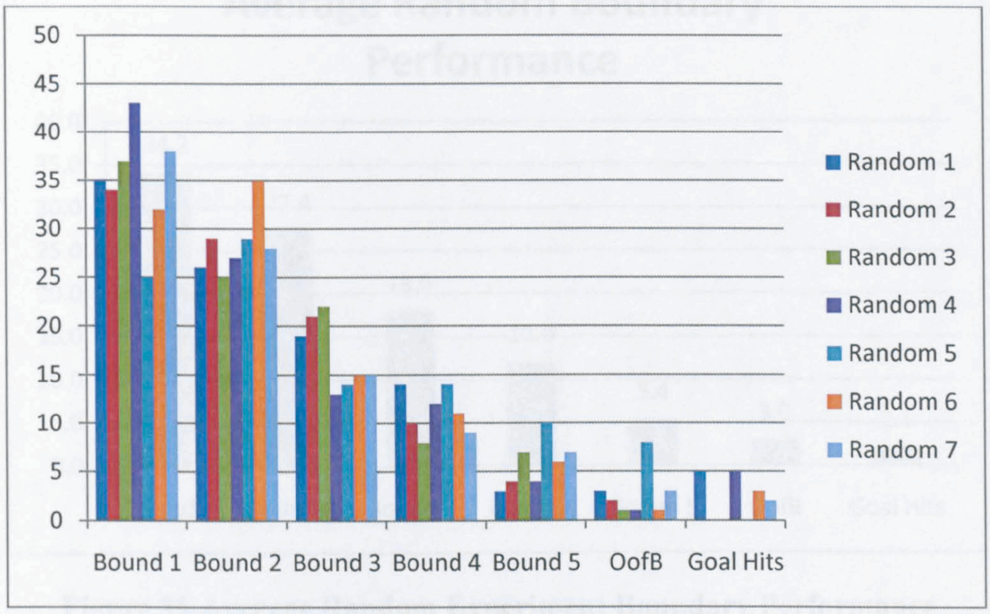


Figure 34 The Results of Random Experiment 1 – 7

Figure 35 illustrates the results obtained from random experiments 1 – 15. The results are grouped by experimental result.

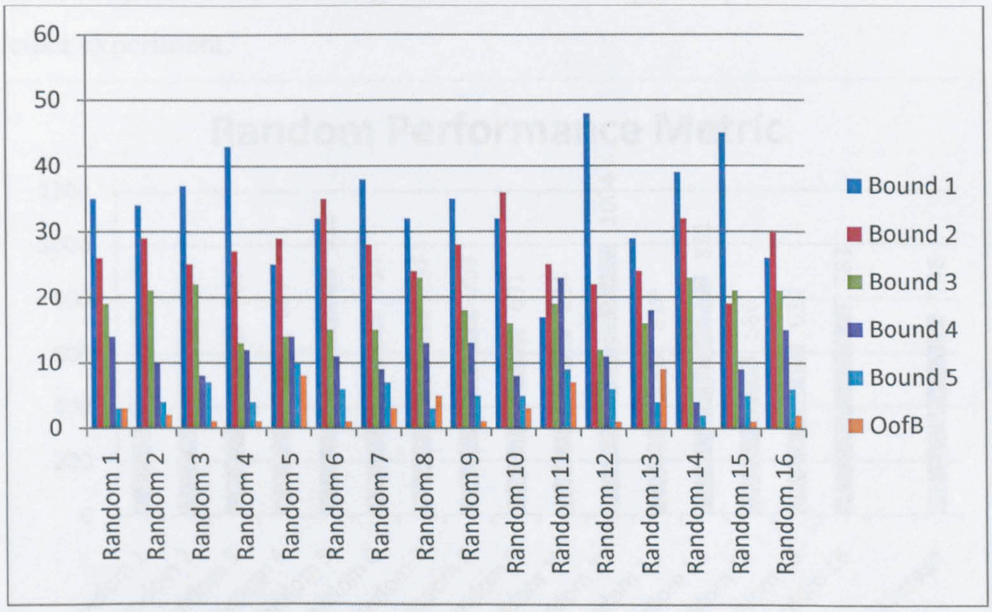


Figure 35 The Results of Random Experiments 1-16

Figure 36 shows the summary of significant experiments related to the random experiment. We can see that the boundary distribution is broadly equivalent to earlier single experiments and will serve as a useful comparator against subsequent experimental results obtained from the "on-world" experiments.

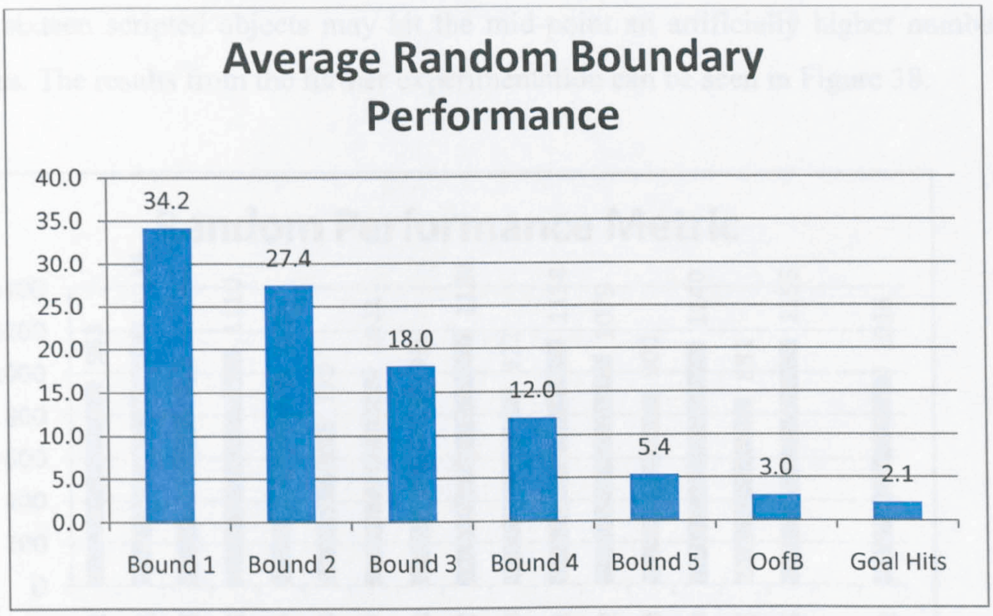


Figure 36 Average Random Experiment Boundary Performance

Figure 37 shows the performance metrics obtained from the random experiments. A score of zero would reflect that the goal metric had been obtained on every single FTime reading and therefore a higher score indicates poorer performance in this particular experiment.

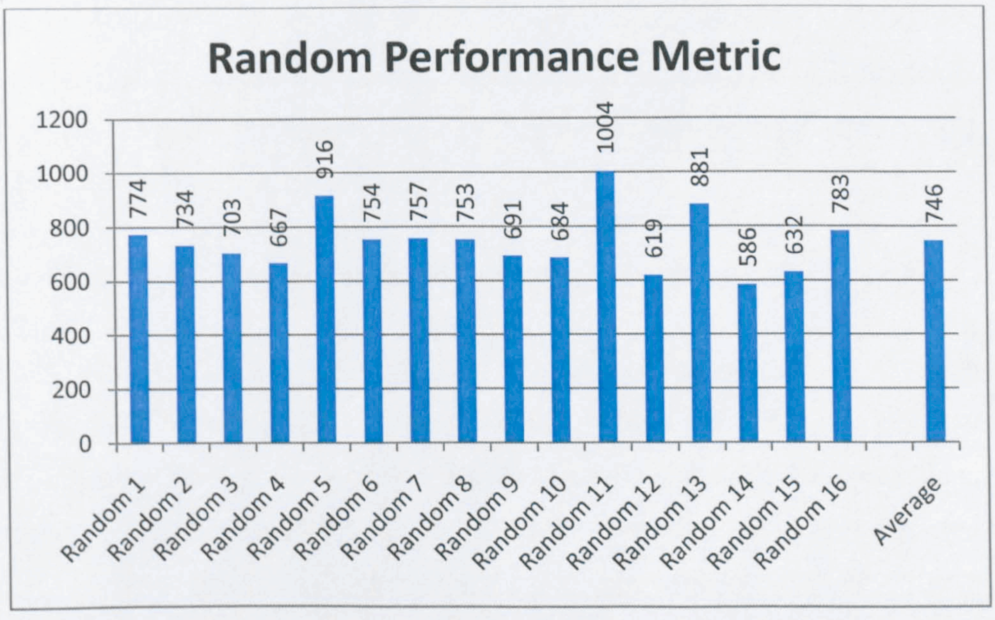


Figure 37 Random Experiment Performance Metric Results

It was decided that it would be useful to re-test the random model with an FTime goal which would be calculated as the baseline reading + 0.5. The reason for this further experimentation is that it was thought that the random experiment had scored higher as the goal had been set at the mid-point and therefore a random selection of the sixteen scripted objects may hit the mid-point an artificially higher number of times. The results from the further experimentation can be seen in Figure 38.

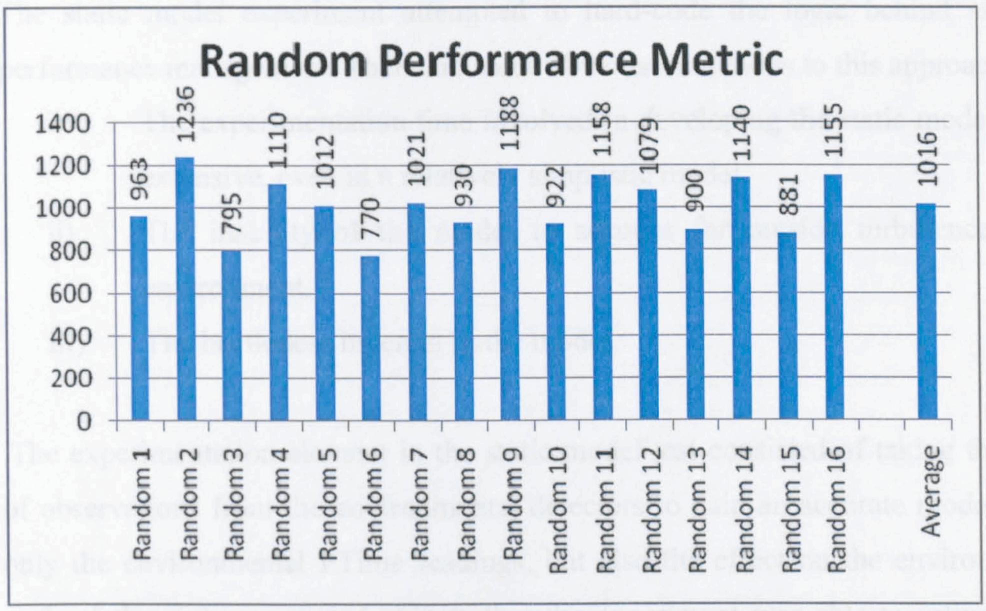


Figure 38 Random Experiment Performance using Baseline + 0.5 Goal

5.5 The Static Model Experiment

A more traditional approach to this type of problem may have been to perform significant "off-line" learning to ascertain through experiment, the average contribution of each scripted object and then build a static model based on these results.

The static model experiment attempted to hard-code the logic behind simulator performance management. There are some obvious limitations to this approach

i) The experimentation time involved in developing the static model can be extensive, even in a relatively simplistic model.

ii) The inability of the model to account for outside turbulence in the environment.

iii) The brittleness inherent in the model.

The experimentation element in the static model test consisted of taking thousands of observations from the environmental detectors to gain an accurate model of, not only the environmental FTime readings, but also the effect on the environment of each of the sixteen scripted objects that the experiment was able to influence and control.

The experimental approach was to calculate by extensive repeated experiments values for the target goal but also for the impact of each individual scripted object on the environment. It was hoped that the extensive calculation would provide the system the "expert" knowledge with which to adequately exert control over the simulator region and its performance.

The outline design for the static model experiment was as follows:-

The first stage of this process was to develop the experimental data with which to build our static model. This was developed using in excess of ten thousand environmental observations to obtain an average goal and contribution of the sixteen individual scripted objects. Once this data had been obtained it was "hard-coded" into the scripted controller with the hope that that observations would allow the system to make "good" decisions when attempting to manage FTime.

5.5.1 Results from the Static Model Experiments

Results from the experimentation phase that informed the static logic model can be observed in Figure 39. This was achieved by measuring the average goal and individual scripted object FTime contribution in order to allow an effective static model to be developed.

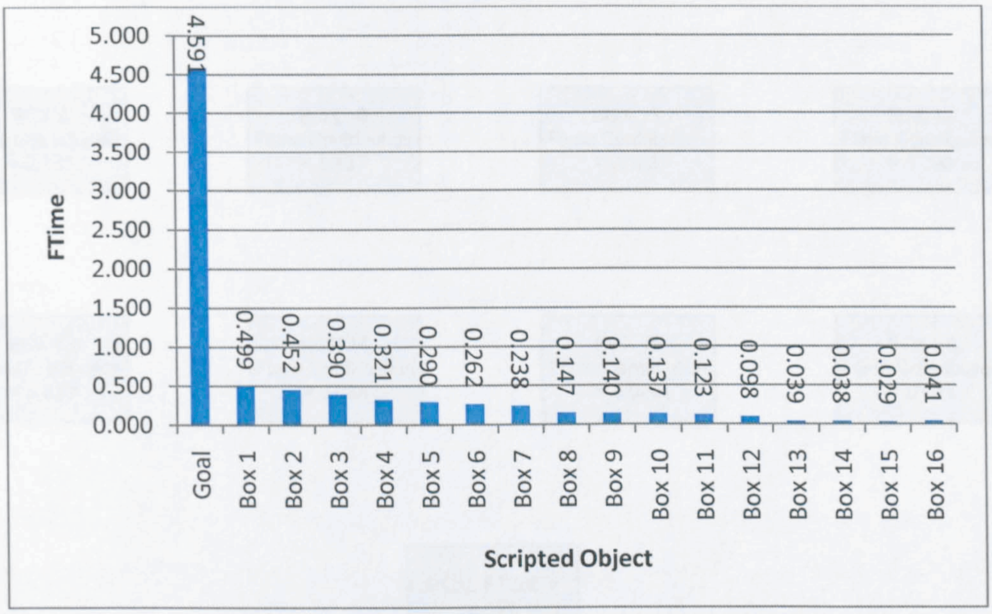


Figure 39 Average FTime observed from 10,000 environmental readings

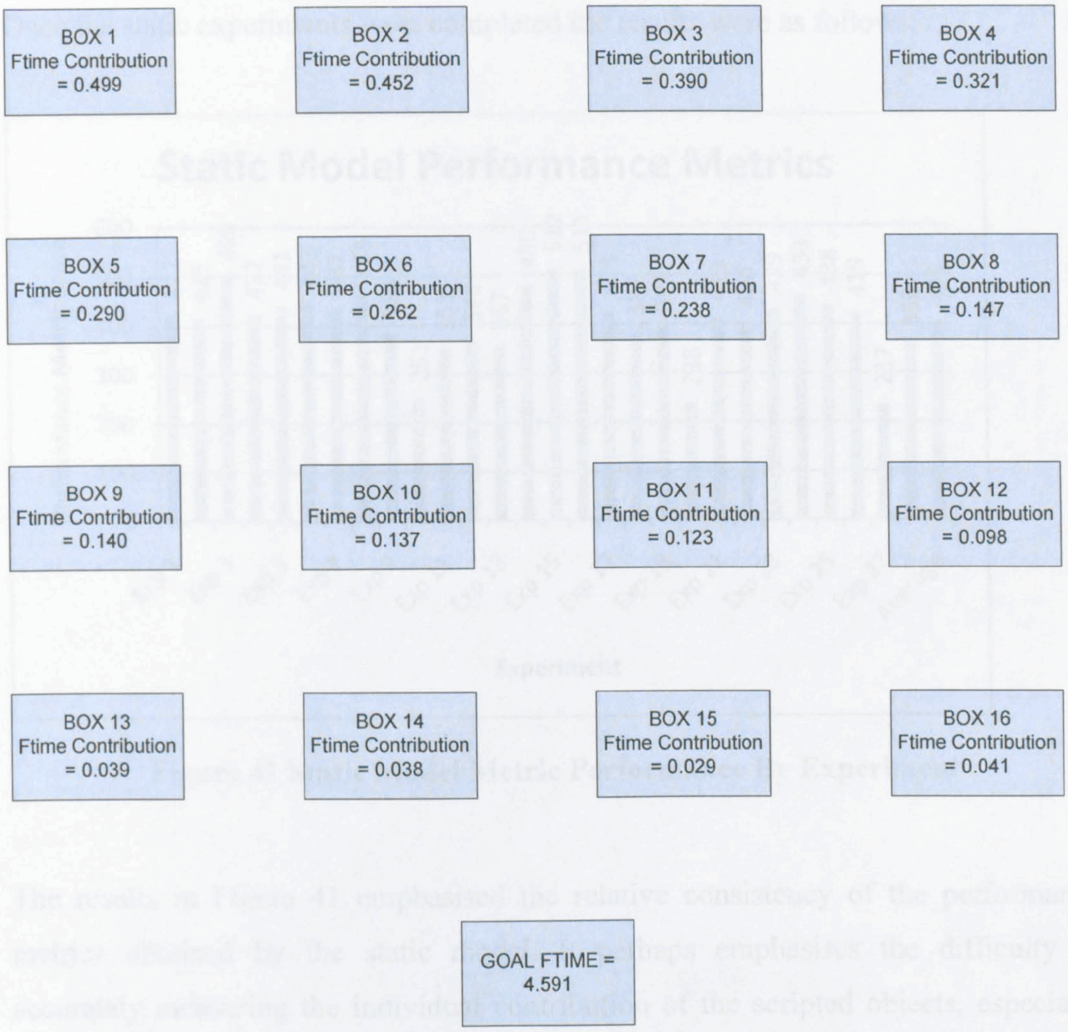


Figure 40 Results of Static Model Experimentation Phase

One obvious limitation of this approach was that each individual box was measured in isolation. To obtain a complete model of the effect of actions on the environment it would have been necessary to measure the effect of multiple boxes being switched on or off and measuring their combined effect to add to the richness of the model. To achieve this would take required 65536 individual experiments (all possible combination of the sixteen boxes being switched either on or off) to gain this knowledge. This would have proved a time consuming process and therefore the approach was to "chain" combinations of boxes based on their expected individual contribution.

Once the static experiments were completed the results were as follows:

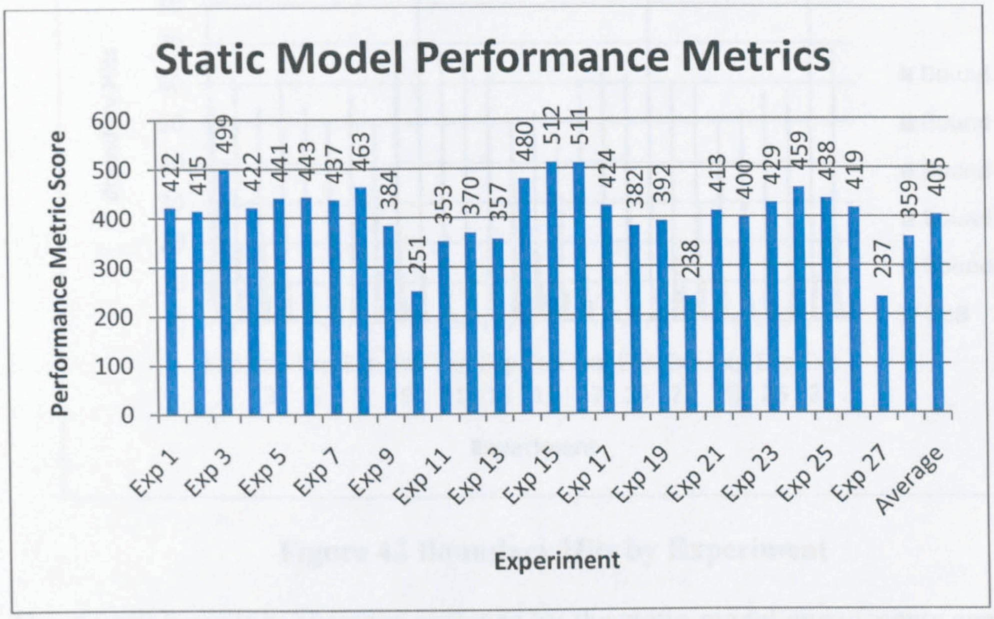


Figure 41 Static Model Metric Performance By Experiment

The results in Figure 41 emphasised the relative consistency of the performance metrics obtained by the static model. It perhaps emphasises the difficulty of accurately measuring the individual contribution of the scripted objects, especially for the objects containing fewer additional scripts, when readings are increasingly affected by external environmental noise.

Figure 42 details the individual experiments results in terms of boundary hits. Again the results obtained are broadly consistent across the experiments.

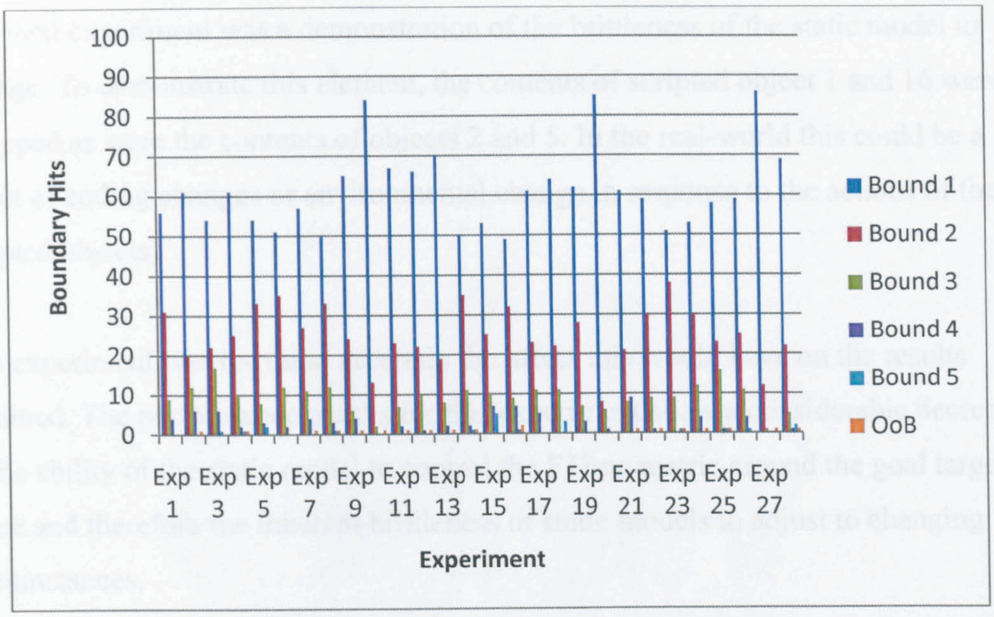


Figure 42 Boundary Hits by Experiment

The overall boundary averages obtained by the static model experiments can be seen in Figure 43. The performance on average saw 61.29% boundary 1 hits (within +/- 0.4 of the goal FTime).

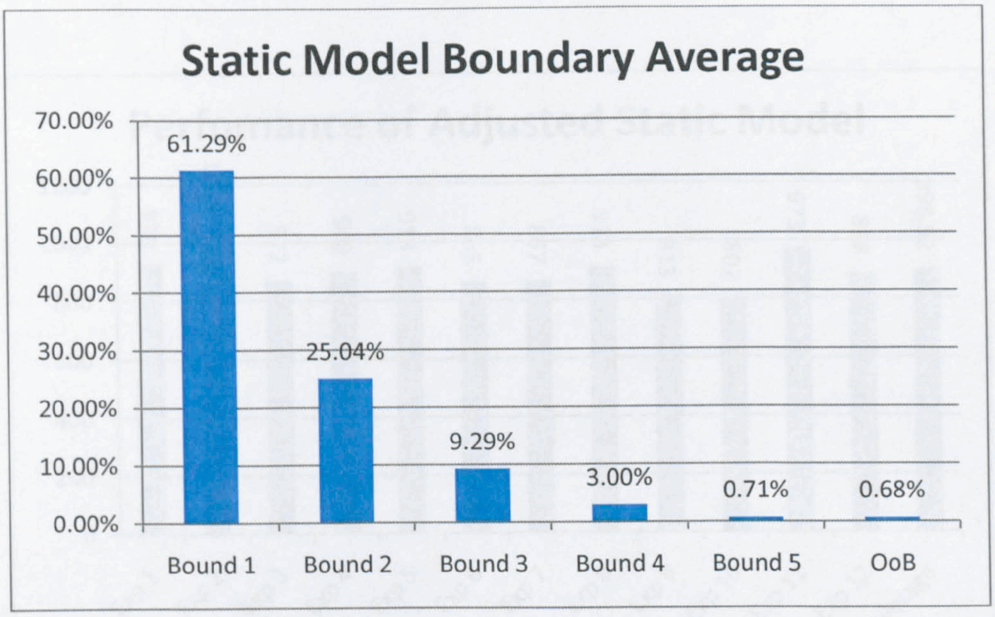


Figure 43 Static Model Boundary Average

The static model performs relatively well while the contents of the scripted boxes do not alter, or indeed outside environmental turbulence does not render the goal unobtainable.

The next experiment was a demonstration of the brittleness of the static model to change. To demonstrate this element, the contents of scripted object 1 and 16 were swapped as were the contents of objects 2 and 5. In the real-world this could be a result of coding changes or environmental change in response to the actions of the scripted objects.

The experiment was re-run to ascertain the affect this would have on the results obtained. The performance graphs for this experiment show a considerable decrease in the ability of the static model to control the FTime metric around the goal target value and therefore the inherent brittleness of static models to adjust to changing circumstances.

The average performance of the adjusted static model was 899 compared to the 405 obtained in the original experiments. These results are not unexpected but demonstrate the problem of any static model in that they can take considerable experimentation to develop but can be rendered obsolete by changes within the system or in the environment itself.

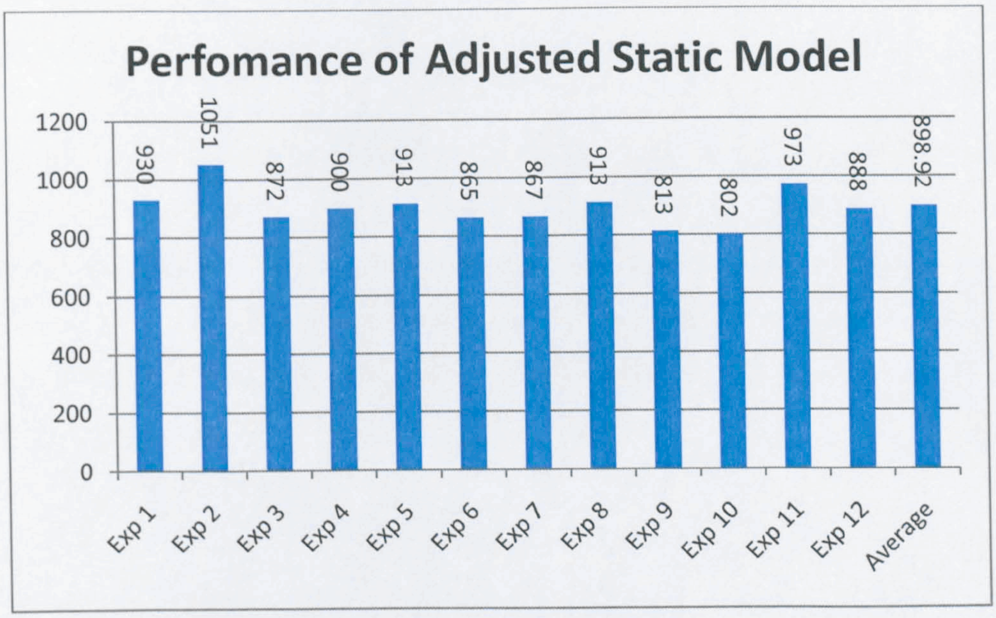


Figure 44 Performance of Adjusted Static Model

This decreased performance levels are also emphasised by the boundary performance metrics where it can be seen that the ability to control the environment has reduced in terms of effectiveness.

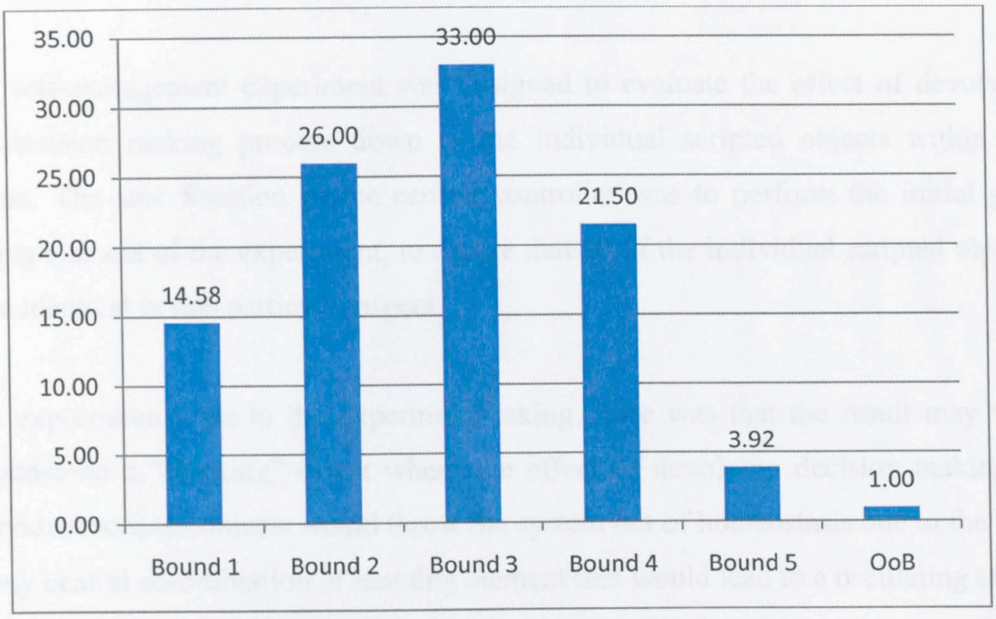


Figure 45 Boundary Performance of Adjusted Static Model

The overall conclusion from these particular set of experiments is that the prototype system must be able to perform continuous evaluation of its own performance in order to react and evolve to meet changing requirements and environmental turbulence.

5.6 The Self-Management Experiment

The self-management experiment was designed to evaluate the effect of devolving the decision making process down to the individual scripted objects within the region. The sole function of the central controller was to perform the initial goal setting element of the experiment, to ensure that all of the individual scripted objects were identical in this particular aspect.

The expectation prior to the experiment taking place was that the result may well demonstrate a "flocking" effect where the effect of devolving decision making to individual scripted objects would throw the system out of homeostasis due to the lack of any central co-ordination or learning element that would lead to a oscillating effect of scripted objects being turned on and then off.

The program outline design is outlined in Appendix A, Figure 140.

5.6.1 Results from the Self Managing Experiments

The results obtained from the experiment were as follows:

Figure 46 shows the average performance of the self-managing scripted object in managing FTime. It is clear that the results are relatively poor compared to the other on-world experiments. The results of the metric were consistent within an average performance of 1113. This emphasises the issue of a lack of centralised planning and control which leads to a violent swaying of decision making as the boxes "lurch" from one state to another.

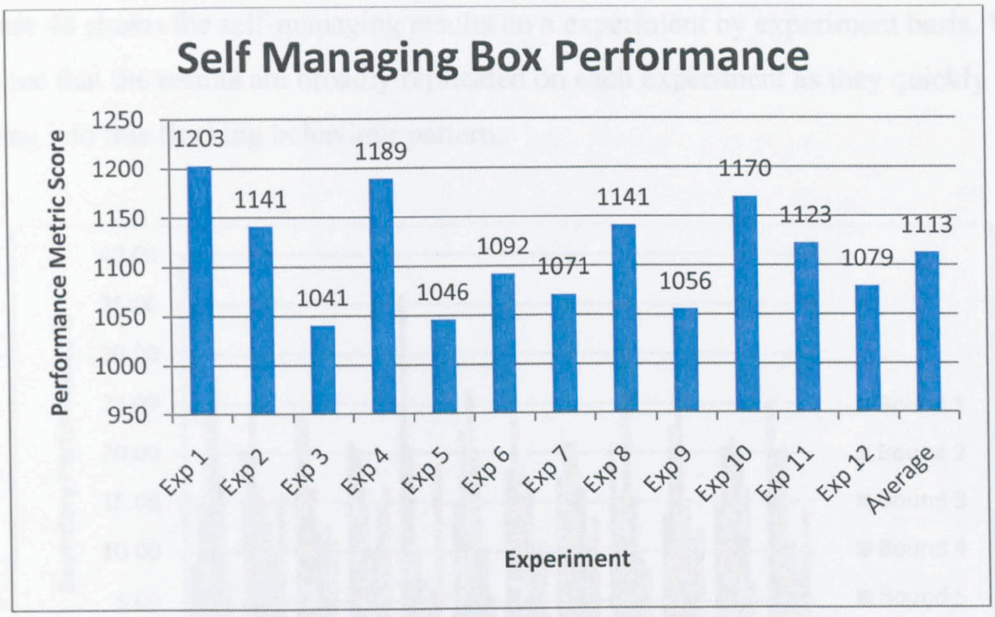


Figure 46 Self Managing Box Performance

In terms of boundary performance it can be seen that the outer ranges are hit far more often than the other experiments utilised.

This again emphasises that the FTime readings, on which the scripted objects are basing their decision, to switch additional scripts on or off, often lay on the outer boundaries. The effect of this factor is that local decisions are being made that do not contribute to the system maintaining the goal state because of a lack of co-ordination between the "local" decision making scripts.

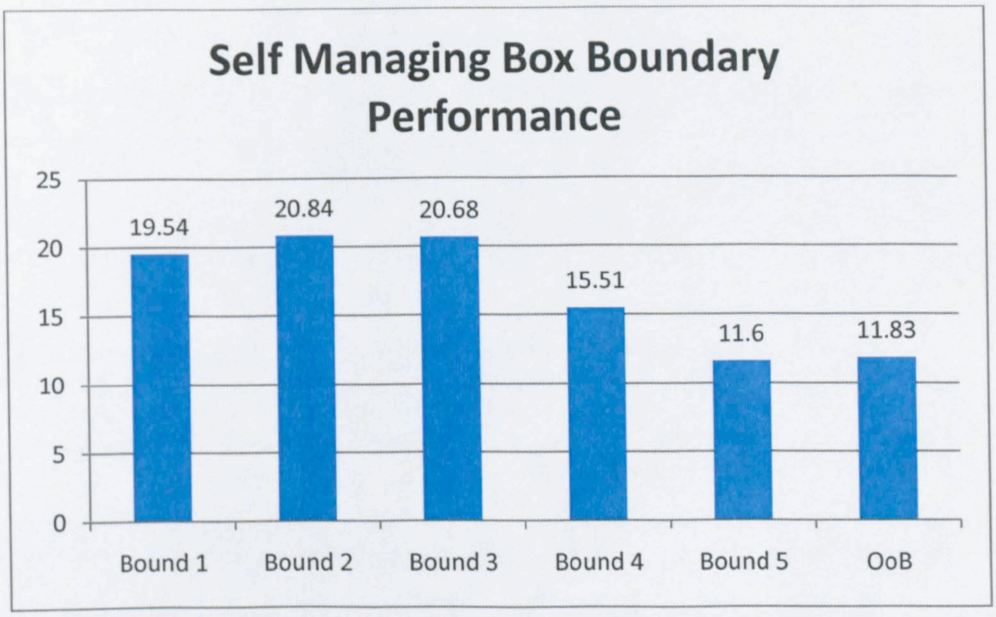


Figure 47 Self-Managing Box Boundary Performance

Figure 48 shows the self-managing results on a experiment by experiment basis. We can see that the results are broadly replicated on each experiment as they quickly falling into this flocking behaviour pattern.

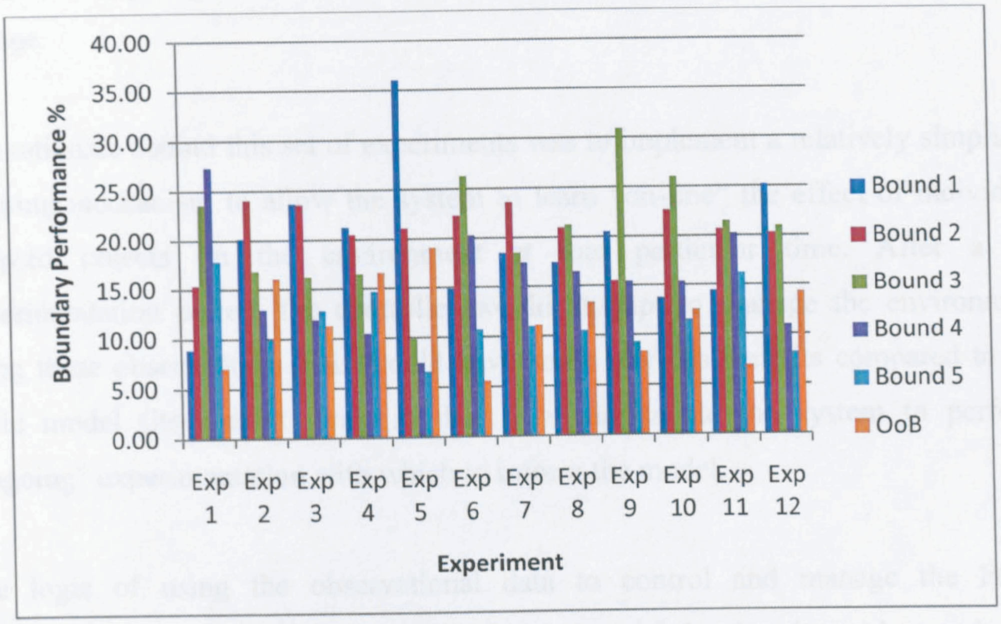


Figure 48 Self-Managing Performance by Experiment

5.7 The Hybrid Experiment

None of the experiments undertaken so far had attempted to give the controller any online learning capability to allow better responses with respect to environmental change.

The rationale behind this set of experiments was to implement a relatively simplistic learning mechanism to allow the system to learn "on-line" the effect of individual scripted objects on the environment at that particular time. After a set experimentation period, the controller would attempt to manage the environment using these observations. This would have some obvious benefits compared to the static model (its closest rival) in that it would enable the system to perform "ongoing" experimentation with which to inform the model.

The logic of using the observational data to control and manage the Ftime environmental messages replicated the static model in that it used a series of *If...Then* statements to chain together individual expected contributions to manage the simulator region.

One problem with this approach is that the online experimentation element would limit its applicability as the system effectively "abandons" control of the environmental FTime whilst the online training is being performed. Therefore, this approach would not be suitable for any system where it is undesirable to perform this type of learning (e.g. safety control system).

The outline of the experiment design is can be observed in Appendix A, Figure 141.

5.7.1 Results from the Hybrid Model Experiments

The results from the experiment were as follows:

The average performance of the hybrid model was slightly worse than the static model. This was mainly down to the performance of one particular experiment

(experiment 3). The reasons behind the problems with this particular experiment will be discussed later in this section.

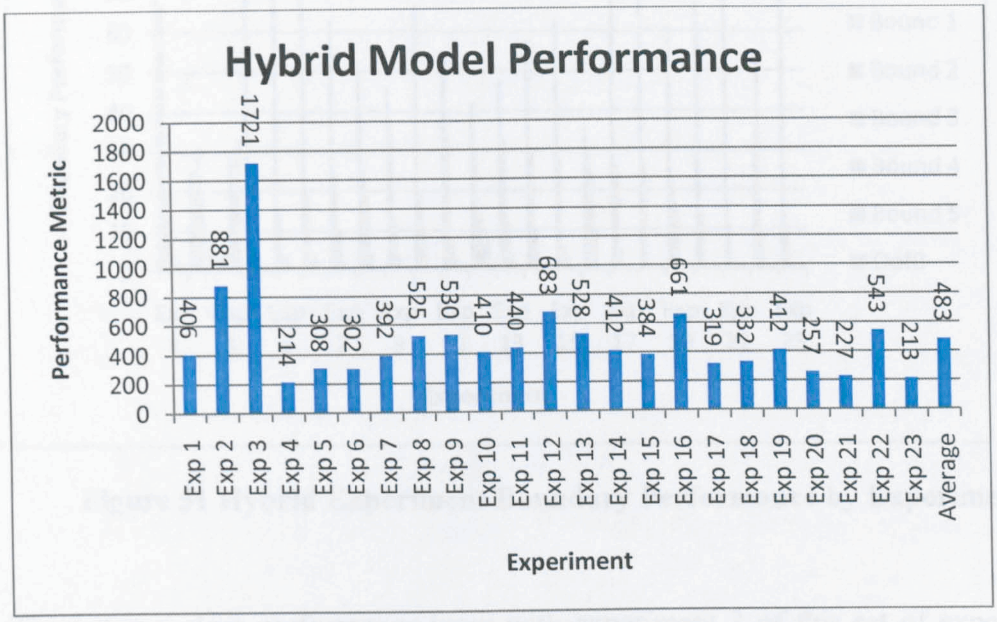


Figure 49 Performance of the Hybrid Model

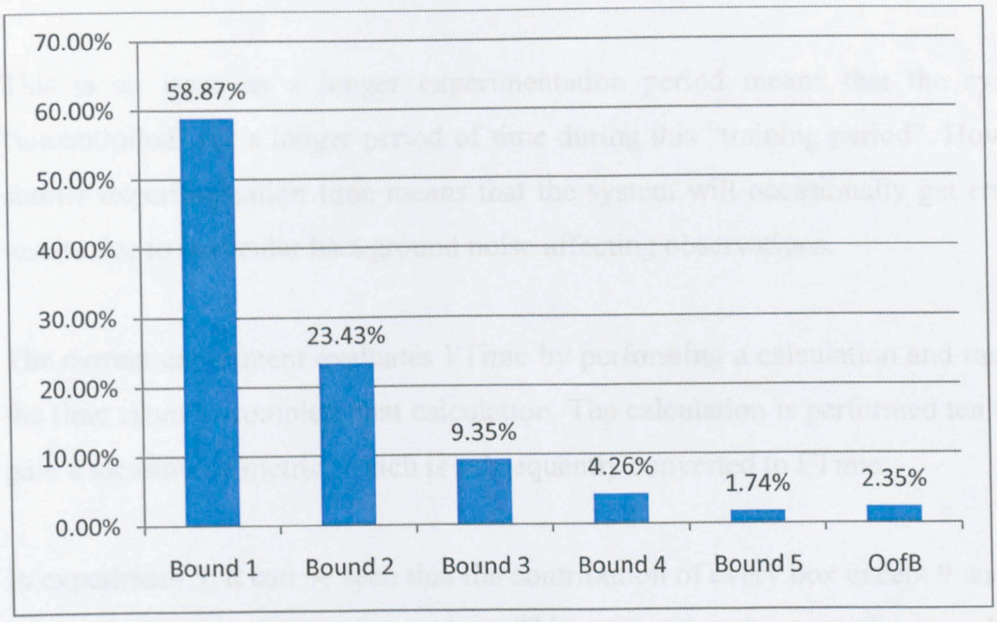


Figure 50 Average Boundary Performance of the Hybrid Model

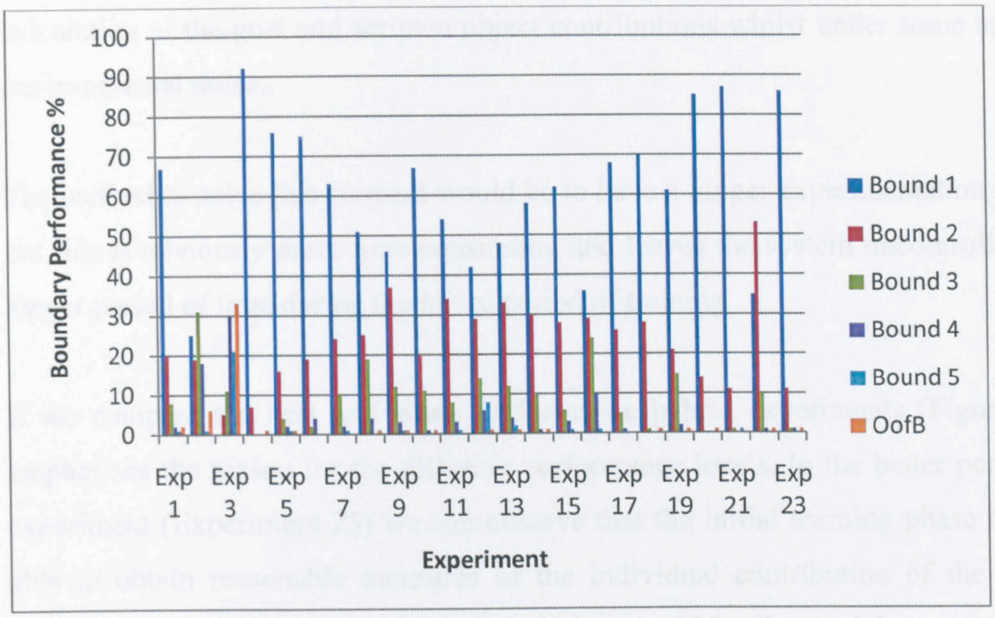


Figure 51 Hybrid Experiment Boundary Performance by Experiment

There was a clear performance issue with experiment 3 of this set of experiments. The issue can be seen in Figure 51 but fundamentally relates to the brevity of the experimental period for ascertaining the individual contribution of the boxes.

This is an issue as a longer experimentation period means that the system is "uncontrolled" for a longer period of time during this "training period". However, a shorter experimentation time means that the system will occasionally get erroneous results due to particular background noise affecting observations.

The current experiment evaluates FTime by performing a calculation and measuring the time taken to complete that calculation. The calculation is performed ten times to gain a measurable metric, which is subsequently converted to FTime.

In experiment 3, it can be seen that the contribution of every box except 9 and 10 has been calculated as being zero or less. This means that the controller can detect no benefit of switching those boxes off to lower the FTime readings. Therefore, the system can only see a measurable benefit of turning off boxes 9 and 10. It has switched both boxes off for the entire experimental period but this obviously has not been enough to control the FTime effectively. The cause of this problem is the calculation of the goal and scripted object contributions whilst under some unknown environmental noise.

The method to solve this element would be to have a longer experimentation element but this is obviously more time consuming and leaves the system uncontrolled for a longer period of time during the initial period of training.

If we compare the best and worst performance hybrid experiments (Figure 52) it emphasises the reason for the differing performance levels. In the better performing experiment (Experiment 23) we can observe that the initial learning phase has been able to obtain reasonable measures of the individual contribution of the scripted objects. This information has subsequently been used by the model to make “good” decisions in terms of the likely results of switching these objects on or off.

Therefore it demonstrates the potential brittleness of using purely an initial training (or learning) approach to inform the model.

	Experiment 3		Experiment 23	
Box	Measured At	Switched Off	Measured At	Switched Off
1	-0.30	0	0.90	87
2	-0.40	0	0.50	54
3	-0.70	0	0.60	100
4	-0.60	0	0.20	81
5	-0.50	0	0.30	76
6	-0.50	0	0.30	94
7	-0.70	0	0.30	24
8	-0.70	0	0.20	30
9	0.20	100	0.10	67
10	0.10	100	0.40	0
11	0.00	0	0.30	0
12	0.00	0	0.20	24
13	-0.10	0	0.00	0
14	-0.30	0	0.10	67
15	-0.20	0	-0.10	0
16	-0.80	0	0.00	0

Figure 52 Comparison of the Best and Worst of the Hybrid Experiments

5.8 Critical Evaluation of the Virtual World Experiments

It is perhaps unsurprising that the static model should show the best performance levels in a relatively stable environmental setting. The performance of the hybrid online learning model is perhaps unfairly negatively skewed by the exceptionally poor performance of experiment 3. If the results of experiment 3 are discounted the average metric performance would have been 426 (as opposed to 483, as outlined in the figure below).

The random experiment performed better than might have been expected but can be attributed mainly to the nature of the experiment where a goal at the mid-point was defined.

The self managing experiment performed more poorly than perhaps would have been expected. It emphasised the requirement for the role of a central controller that could co-ordinate actions with respect to an overall goal.

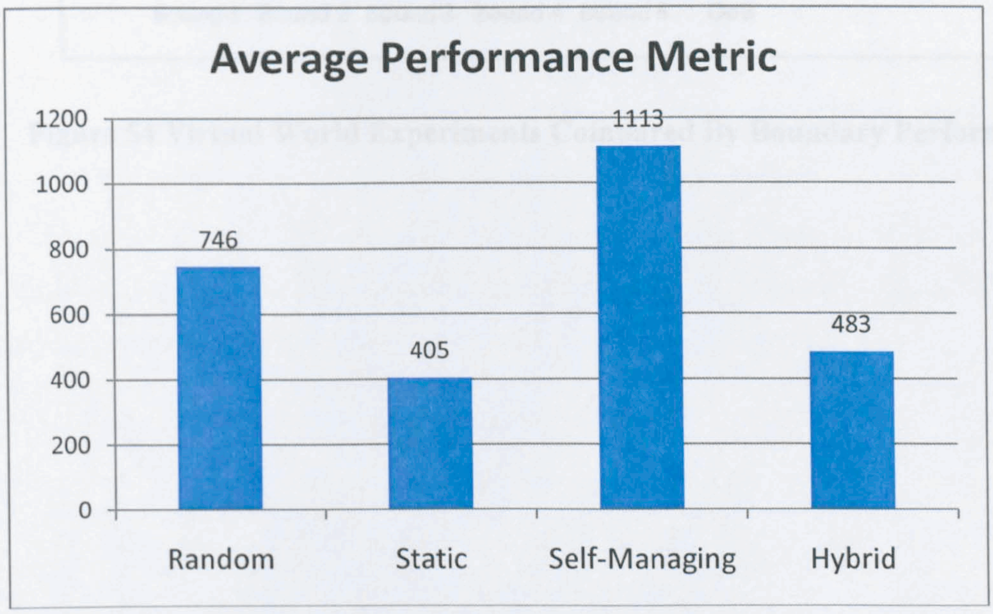


Figure 53 Virtual World Experiments Compared by Metric Performance

Figure 54 illustrates the comparison of the virtual world experiments in terms of boundary hits and emphasises the closeness of the hybrid and static model experiments in terms of achieving boundary 1 hits (with +- 0.2 of the goal ftime).

Boundary 2 hits were relatively equal across the experimental range and we can see the poorer performing experiments (especially the self managing experiment) as seeing increasing Boundary 5 and Out of Bounds (OoB) hits in their results profile.

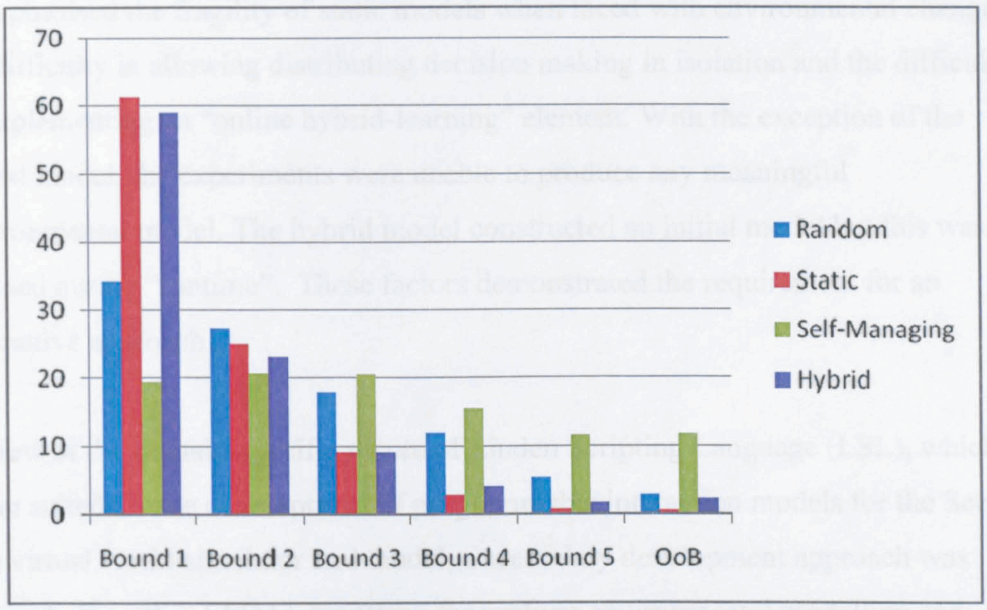


Figure 54 Virtual World Experiments Compared By Boundary Performance

5.9 Summary

The virtual world based experiments were extremely interesting and useful in providing an underpinning and understanding for the likely requirements for the development of the full prototype model. The metrics record would provide a method of comparison between different approaches and the relative strength and inherent weaknesses between approaches.

It emphasised the fragility of static models when faced with environmental change, the difficulty in allowing distributing decision making in isolation and the difficulty in implementing an "online hybrid-learning" element. With the exception of the hybrid model, the experiments were unable to produce any meaningful environmental model. The hybrid model constructed an initial model but this was not updated during "runtime". These factors demonstrated the requirement for an alternative approach.

In view of the domain-specific nature of Linden Scripting Language (LSL), which is better suited for the development of programmable interaction models for the Second Life virtual world simulator and model, a secondary development approach was required. The other EMMA functions for runtime environmental modelling and control system were developed using C / PHP and deployed on a web server. It is also worth considering the fact that the proposed control system runs "off-world", which in some respects changes the experiments slightly. These experiments had ran "in-world" and would consequently impact performance, whereas "off-world" experiments export some of that overhead.

Chapter 6: EMMA Testbed Design to provide a Model of the External Environment and Internal Capability

6.1 Introduction

In the following sections an overview of the experiments will be discussed in terms of the development towards the final prototype system development.

The goal of the EMMA prototype system is to monitor, model and adaptively control a Second Life simulator (or island region) to maintain performance levels close to a target goal. To achieve this goal, EMMA should hold a current model of the variety that is evident in the environment and also an appropriate response (i.e appropriate actions given a particular environmental state). There were a number of building blocks required before we could attempt to develop an experiment to investigate this research problem.

The following elements needed to be developed:

i) A metric for accurately measuring performance levels.

ii) A method of exporting this metric "off-world" to the control system.

iii) A method of passing messages from our prototype into the virtual world environment.

iv) A method of detecting these messages and enacting change on the virtual environment as a consequence of these instructions.

To explain how genetic algorithms work it is perhaps best to illustrate how they will be used in our virtual world experimental platform. Goldberg's Simple LCS in Pascal was used as a design guide in developing the some of the building blocks of the prototype system (Goldberg, 1989).

Experimentation will be performed using sixteen functions (represented visually as boxes) on the virtual world simulator. Each of these boxes contain LSL (Linden Scripting Language) code. Initially these will considered as "black boxes", in that, the computational cost of running any individual agent (ie script) on the global performance of the simulated world is unknown.

For the purpose of the experiment, each of the sixteen scripts can be turned on or off (represented by 1 or 0) remotely via the Environmental Modelling, Monitoring and Adaptive (EMMA) system. The mechanics of this will be discussed in detail later within this chapter. The goal of the control centre will be to maintain simulator performance at an acceptable level by reacting to key simulator performance indicators (dilation and script throttling). This will be achieved by adjusting the combination of scripts that are running at any one time.

Therefore, if none of the sixteen scripts are currently running, this will be represented by 16 zeroes (0000000000000000) whereas if all scripts are running this will be represented by 16 ones (1111111111111111). Therefore, there are 65536 possible permutations. If one more script were added to this set then the potential permutations would increase to 131072. Therefore, if we are considering systems of increasing complexity and environmental awareness then targeted searching of the environmental "space" becomes increasingly important and vital in providing timely and robust responses to environmental change and turbulence. Bull identified a potential problem in that some environments that may not be completely observable by the learning entity (limited sensory input) and that this may subsequently lead to the same sensory input for different environmental states, which presents a challenge where a learning system can not perform optimally (Bull & Hurst, ZCS Redux, 2002). This has implications when designers are deciding on the level of environmental sensitivity that a system is likely to require.

As genetic algorithms begin to search from an initial population of strings, one of the first tasks is to generate the initial population. This will be done by randomly generating a number of initial strings (of 16 bits in this example) using a notional random coin toss. So, for simplicity we might generate an initial population of 10 strings (as illustrated below)

0101010001110001

1001000001000010

1110001110101001

0110110010111111

0001110011100100

0101011010101011

0111101010101111

0010101110100101

1111110000110010

1010110000000000

For the virtual world simulator, these strings will represent whether a script function is enabled (1 is on and 0 is off). There will be a "fitness" or strength value associated with each of these strings. In this example, this will be represented by the effect that the string action has of improving or degrading simulator performance.

At this point, we have a population of 10 strings (the initial population) each of which has a strength value associated with it. Then genetic algorithms use probabilistic transition rules to guide the next element of the search, which is to generate subsequent generations of the population.

This is achieved by using selection, reproduction, crossover and mutation operations in the hope of combining the best features of "parent" strings and thereby generating "stronger" string populations. Indeed, Holland suggests that the goal of exploring our search space is not one of "finding the best individual" but rather it is to locate a succession of improvements in the search for individuals of "ever higher fitness" (Holland J. , 2000). There are various ways to implement reproduction but in simple terms we reproduce (or copy) strong strings (as determined by their strength or "fitness") into the pool for the next generation of strings. The strings are selected using a random approach, organised to favour stronger strings (although other selection schemes are possible e.g. "alpha male". These strings are then "bred" in the hope that the string produced will be stronger than the two parent strings and so on. Over subsequent generations of strings, weaker stings are "bred out" of the population.

Breeding of strings occurs by a process called crossover and is achieved by randomly "mating" strings from the pool for the next generation of strings. This mating occurs by randomly generating a break point and swapping over the contents of the two strings at that break point therefore creating new strings consisting of building blocks of the "parent" strings.

For example, using the initial population of strings, two strings are selected from the next generation pool because of their strength (fitness value). This concept is equivalent to Darwinian "survival of the fittest". If the strings are:

0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	1

0	1	1	1	1	0	1	0	1	0	1	0	1	1	1	1

A random "break point value" is generated between 1 and 15 (which is string length – 1). If this random value was 5 then we would break the strings at position 5 and swap (crossover) the remainder of the string value so

0	1	0	1	0	Break-Point	1	1	0	1	0	1	0	1	0	1	1

0	1	1	1	1	Break-Point	0	1	0	1	0	1	0	1	1	1	1

will then become a "bred" child string consisting of the following values

01010 (from Parent 1) plus 01010101111 (from Parent 2)

and therefore a value of

0	1	0	1	0	0	1	0	1	0	1	0	1	1	1	1

These new members of the population will subsequently be tested for fitness (payoff value) and the process is replicated over subsequent generations. This breeding of stronger string values then provides the basis for a more targeted search of the

complex space for an increasingly "fit" solution to the goal. Therefore, while the genetic algorithm search gives the impression of randomness it is actually a form of guided randomness.
The concept of mutation is explained by Goldberg (Goldberg, 1989) as being required to guard against some potentially useful genetic material (a 0 or 1 at a particular location in the string value) being bred out by the genetic algorithm process. To guard against this, a random mutation will be introduced into the population at a specified frequency. This is achieved by flipping a random element (either in the condition or message) to another value. In the prototype system this would mean 0 to a 1, or vice versa, in the message element or -1 (representing a wild-card value), 0 or 1 in the condition element. Mutation provides a route back from the search becoming "stuck" on a local maxima as well as the possibility of serendipitous innovations, whilst noting that mutation may have a detrimental effect on a classifier. As Goldberg says mutation, is itself, a random walk through the search space (Goldberg, 1989).

6.2 Metrics, Detectors, Effectors and Communication Channels

The first element in providing this functionality is the ability to both detect messages from the environment (of the simulator) and be able to provide a method of enacting change through the use of effectors. A further requirement, given that the simulator would be controlled externally to the Second Life platform, would be the ability to open a communication channel to facilitate the transfer of data "off-world" to our EMMA prototype system.

One of the fundamental reasons that Second Life was chosen as a platform was that it contained many of the envisaged building blocks required for development of the prototype.

Second Life simulators produce a number of metrics which prove useful as metrics of performance of the simulator. Some of the metrics reported by the system included dilation, Simulator FPS (Frames per Second), Physics FPS, agent updates per second, total frame time and Sim-Time.

6.2.1 Scripting of Detectors

LSL provides the functionality required to detect states from the environment and can subsequently take actions based on this information. For example,

```
llSensor("", NULL_KEY, AGENT, 3, PI);

llListen(5,"",NULL_KEY,"");
```

The first of these script lines will detect all primitive objects and agents within a 3 metre radius of the sensor while the second script line will listen on a specified communication channel (5 in this instance) for any messages broadcast on that particular channel.

For the purposes of this experiment, detectors were required to monitor FTime readings and subsequently report them "off-world" to the EMMA prototype system. This would enable EMMA to detect whether actions taken were beneficial or detrimental to the system. LSL allows the use of http requests through its llHTTPRequest function. This function allows the posting of FTime readings using the following code

```
string url = "http://henry-res.cms.livjm.ac.uk/getlcsreading.php?";
 url += "ftime="+(string)ftime;
 llHTTPRequest(url, [HTTP_METHOD, "POST"], "");
```

This enabled the detectors to report FTime readings using the http function to a PhP script residing on a web server. Therefore, a method of reporting FTime readings out of Second Life in a format accessible by the prototype system was now achieved.

6.2.1 The representation of controllable functionality on the region

It was necessary to develop some functionality on the simulator region that EMMA could control through the use of effectors, to learn from and develop strategies depending on their relative impact on the environment.

The strategy developed for this element was to create sixteen scripted objects (represented by number boxes from 1 to 16). The array of boxes on the region is shown below

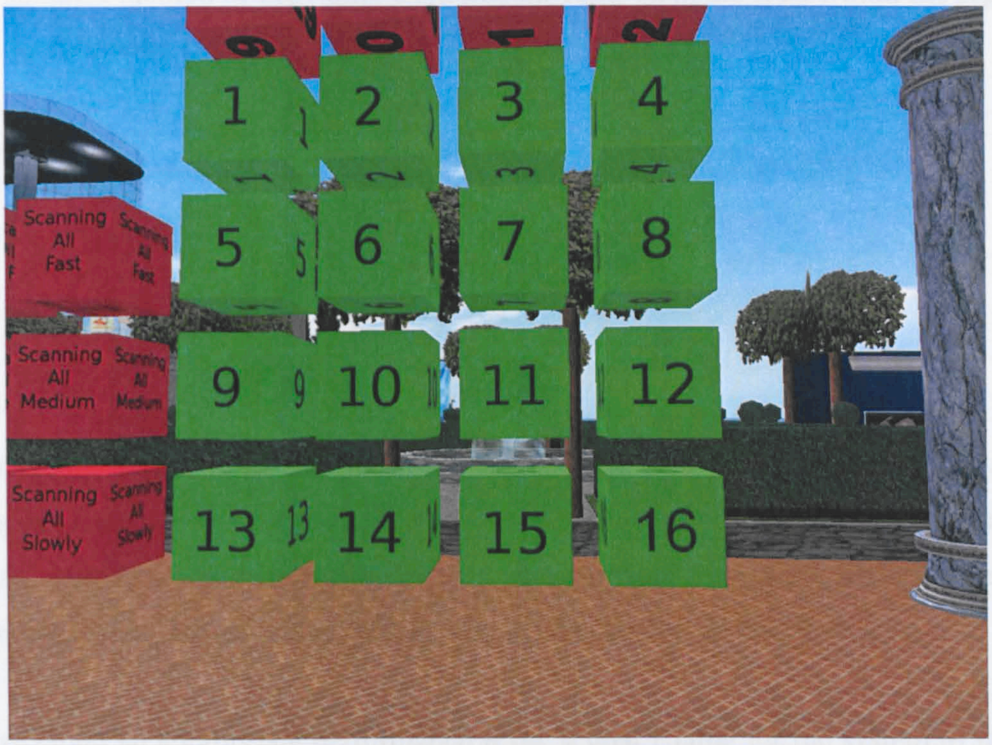


Figure 55 Scripted Objects on the Region Simulator

Each of these boxes consisted of two major elements.

i) A detector script that used the LSL llListen functionality to continuously listen to the environment for instructions being broadcast into the region from the EMMA prototype system. These instructions consisted of either stopping all scripts (apart from the listener itself) in the object or indeed starting all scripts in the object.

ii) A number of scripts specifically used for their ability to induce a large computational load on the region that could subsequently be detected and measured by the FTime detector.

The approach taken was to give each object a different computational effect that varied from Box 1 (highest computational overhead) through to Box 16 (least computational overhead). The challenge of the modelling and monitoring system was to learn the effect of appropriate strategies for maintaining performance levels.

The simplest method of providing a significant computational load within the region was the following code

```
default
{
state_entry()
    {
    while (TRUE) {}
    }
}
```

This code was simply an infinite while loop that presented a significant computational overhead when activated. While one occurrence of this script does not induce significant and more importantly, measureable load once multiple scripts were used then their accumulative effect on the system could be determined.

To ascertain an appropriate performance level, a goal FTime for the system to strive to achieve was required. To find this target the goal detector measured the FTime metric when no boxes were active to measure the baseline performance and therefore the effect of normal environmental noise. All boxes were switched on to measure the maximum FTime reading. The goal was calculated by taking the mid point between the two measurements to obtain a goal that the system would seek to achieve.

6.2.3 The Effectors

In order to effect change on the simulator environment, it was necessary to be able to broadcast messages into the virtual effector system. These messages would be detected by the listeners which could then enable the relevant action to be taken.

Two main components were required to achieve this functionality.

i) The use of an XML-RPC broadcaster to send XML data over HTTP that can subsequently be used with the Second Life region

ii) The use of functions within Second Life including llRegionSay and llListen

An XML-RPC broadcaster was developed as a scripted object within Second Life. It opened a remote data channel that enabled the EMMA prototype system to communicate instructions to the region. This was achieved using both text file and PHP scripts to facilitate the transfer.

Once a message was received by the XML-RPC scripted object, the message was broadcast using the following syntax:

```
llRegionSay(5, sval);
```

The string "sval" would hold the value of the message to be broadcast by the effector.

The use of the llRegionSay command meant that all listening objects within the region would be able to detect the message and act on it if it was appropriate to that particular object.

6.3 The development of EMMA prototype

The development of the fundamental building blocks related to measurement of an appropriate metric, the ability to communicate effectively with the scripted effectors and detectors within the simulator region allowed for the development of the EMMA prototype system.

Linden Scripting Language did not contain the functionality required for the full implementation of the EMMA prototype and therefore it was necessary to choose a development platform. The chosen development language was C and this choice was purely due to previous development experience.

The outline design of the program can be seen in Appendix A, Figure 142.

6.4 The Structure of the Learning Classifier

Goldberg suggests that one of the main obstacles to learning in traditional rule based systems has been the complex rule syntax used in those particular systems. By only allowing a fixed string representation it "permits string operators of the genetic kind" and therefore allows a "genetic algorithm search of the space of permissible rules" (Goldberg, 1989).

A learning classifier rule is typically composed of two parts, a "condition" that is supposed to match the environmental message and an "action" representing the appropriate action for that condition. The condition element must be able to "absorb" the variety of environmental messages being detected by the prototype system and respond with an appropriate message. The message element should be able to represent the capability of the system to respond to particular environmental conditions.

The structure of a classifier record in the EMMA prototype can be observed in Figure 56.

CONDITION 10 Bit Signed Binary String	ACTION 16 Bit Unsigned Binary String	CLASSIFIER STRENGTH	BID (IF MATCH) (SPECIFICITY * STRENGTH)	PROTECTION (IF PREVIOUS BID WINNER)	MATCH FLAG (indicates match to environmental value)	SPECIFICITY (Number of 0s and 1s in Condition)	DUPLICATE FLAG (indicates matching winning bids)

Figure 56 Emma Prototype Classifier Structure

6.4.1 The structure of the condition in the EMMA prototype

The structure chosen for the prototype system for the condition element was a 20 bit representation (signed 2 bit value for each position).

This would allow an environmental message of 0 – 1024 to be represented within the condition element. This is demonstrated by the illustration below.

512	256	128	64	32	16	8	4	2	1
1	-1	0	0	1	1	-1	-1	0	1

Figure 57 Representation of Classifier Condition Element

Previous experimentation using the FTime metric had demonstrated that environmental messages would not occur outside of this range using the particular range of experiments and therefore this representation was appropriate.

However, it should be emphasised that if we had a higher range of environmental messages than this, the representation could be easily adapted to absorb that increased variety. For example, 0-2048 could be represented using a 22 bit representation for the condition element.

The condition "alphabet" for each of the ten positions in the condition representation consists of 0, 1 or -1. The -1 represents a "wild card" symbol in Goldberg's terminology (Goldberg, 1989), which means that the system can have an either 1 or 0, or "don't care" representation for particular positions in the condition set, thereby allowing each condition to match a greater number of environmental messages and allowing the system to operate with a smaller rule-set.

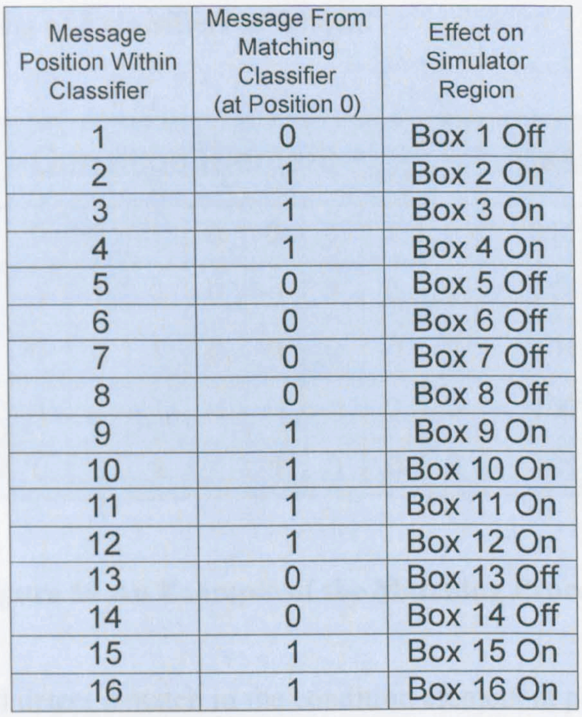
When the classifier set is initialised the condition element is randomly generated and is subsequently modded down to 0, 1 or 2. 1 is then subtracted from the value to give values of 0, 1 or -1, the wildcard. Each position in the condition has an equal probability of being initialised as either 0, 1 or -1.

6.4.2 The structure of the message in the EMMA prototype

The message element of the classifier in the prototype condition represents the capability of the system to effect change in response to environmental messages. In these particular experiments, the system has the capability to control sixteen boxes that represent varying functionality and hence computational load on the simulator region. Therefore, the structure of the message is defined as 16 X 2 bit numbers. These are left as unsigned as the only possible values they can hold are 0 (to represent the functionality being switched off and 1 (to represented the functionality being switched on).

As with the condition element discussed earlier, we are not limited to purely binary representations of messages but it is used for the particular purposes of this experiment.

The message element of the condition represents the switching on or off of those particular functions in the virtual world simulator. A "matching" classifier could send the following message <0111000011110011> to the effectors in the virtual world. This would be interpreted as follows:-



Message Position Within Classifier	Message From Matching Classifier (at Position 0)	Effect on Simulator Region
1	0	Box 1 Off
2	1	Box 2 On
3	1	Box 3 On
4	1	Box 4 On
5	0	Box 5 Off
6	0	Box 6 Off
7	0	Box 7 Off
8	0	Box 8 Off
9	1	Box 9 On
10	1	Box 10 On
11	1	Box 11 On
12	1	Box 12 On
13	0	Box 13 Off
14	0	Box 14 Off
15	1	Box 15 On
16	1	Box 16 On

Figure 58 Converting the Message to an Environmental Action

6.5 Matching Environment Messages to a Learning Classifier

The Environmental FTime reading, currently real values, are converted to binary strings, initially by converting them to integers i.e. multiply by 100 and subsequently converting to binary i.e. modulus and divide by 2.

An environmental message might be detected in the following format:-

FTime = 2.90 (as represented in the text file output by the detectors)

Digital Value = 290 (prototype system multiplies FTime by 100 for conversion purposes)

The conversion to binary (from our digital value) would give 0100100010 which then is the binary representation of our converted FTime value (290).

This binary representation of the environment message is then used to search our learning classifier conditions for potential matches.

So, a system consisting of 5 classifiers as follows

Index	Condition Element										Message Element
0	0	1	0	0	1	0	0	0	1	0	0111000011110011
1	-1	0	1	1	0	0	-1	1	0	0	1111000011110000
2	-1	-1	0	0	1	0	0	0	-1	0	1010101010101010
3	1	0	1	1	1	-1	-1	-1	0	0	1001110111111001
4	0	-1	0	0	1	1	1	0	0	0	1011000011001010

Figure 59 An Example of the Matching Process

The classifier would detect a match in the condition element at position 0 (which is a fully specified match) but also position 2 because of the wildcard used in that particular position.

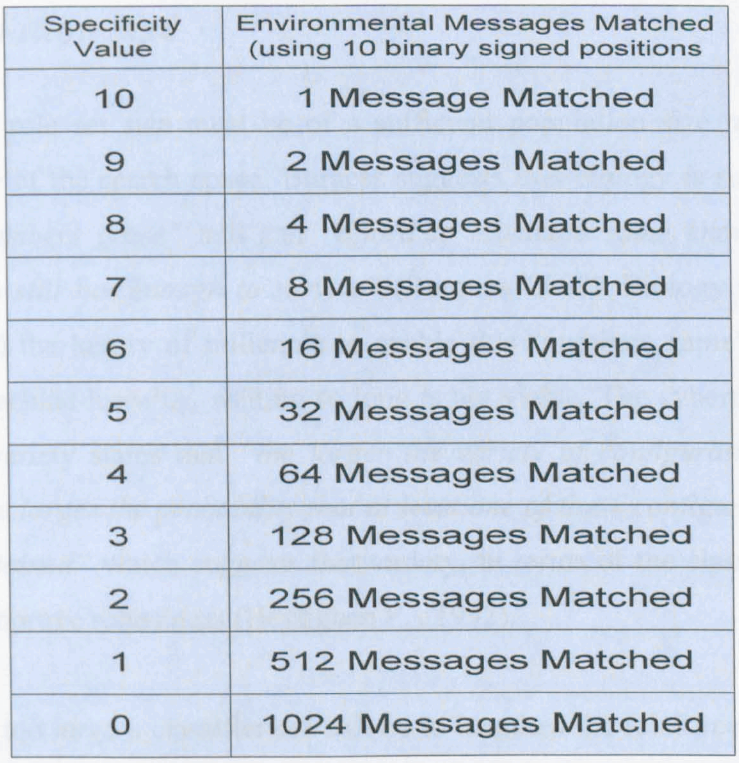
The specificity of each classifier condition is a value that represents how many of the values in the condition are either 0 or 1 (but not the wildcard value of -1). This is calculated by a simple count of each classifier condition position.

Therefore

Index	Condition Element										Specificity Value
0	0	1	0	0	1	0	0	0	1	0	10 (No Wildcards)
1	-1	0	1	1	0	0	-1	1	0	0	8 (2 Wildcards)
2	-1	-1	0	0	1	0	0	0	-1	0	7 (3 Wildcards)
3	1	0	1	1	1	-1	-1	-1	0	0	7 (3 Wildcards)
4	0	-1	0	0	1	1	1	0	0	0	9 (1 Wildcard)

Figure 60 Calculation of the Classifier Specificity

The specificity reflects how many individual environmental messages that the particular classifier could "match" against. So that



Specificity Value	Environmental Messages Matched (using 10 binary signed positions
10	1 Message Matched
9	2 Messages Matched
8	4 Messages Matched
7	8 Messages Matched
6	16 Messages Matched
5	32 Messages Matched
4	64 Messages Matched
3	128 Messages Matched
2	256 Messages Matched
1	512 Messages Matched
0	1024 Messages Matched

Figure 61 Specificity Value to Environmental Messages Matched

Therefore, a "fully specified" classifier condition can only match 1 environmental message, whereas a classifier consisting of purely wildcards would match all 1024 potential environmental messages.

The possibility of multiple matching conditions (as in the example outlined previously) introduces the need for a strategy to decide which classifier is the "winner". The role of the strength and specificity of the classifier in this process is discussed later on in this chapter.

6.6 Initial Parameters

The literature would suggest that determining the optimum settings of the initial parameters is more of an art rather than an exact science. It often depends on the particular application the LCS is developed for and experimentation is often required to initialise the various elements.

The main parameters and rationale for their particular values are discussed in the following sections.

6.6.1 Population Size

The classifier rule set size must be of a sufficient population size relative to the perceived size of the search space. Burgess suggests that biology is successful as it plays *"the numbers game"* and can *"afford to eliminate some knowing that the larger species still has enough to survive"* (Burgess, 2007). Biology in the natural world has had the luxury of millennia to enable this "numbers game" to play out, whereas in machine learning, waiting so long is not viable. The cybernetic principle of selective variety states that *"the larger the variety of configurations a system undergoes, the larger the probability that at least one of these configurations will be selectively retained"* which suggests that variety, in terms of the classifier rule-set, will help to improve robustness (Heylighen F. , 1992).

The effect of too large a classifier set will be to lengthen the reinforcement learning process whereas too few classifiers may limit the effectiveness of the search of the space.

An analysis of the rule-set size in relation to the expected coverage of the environment produced the following results

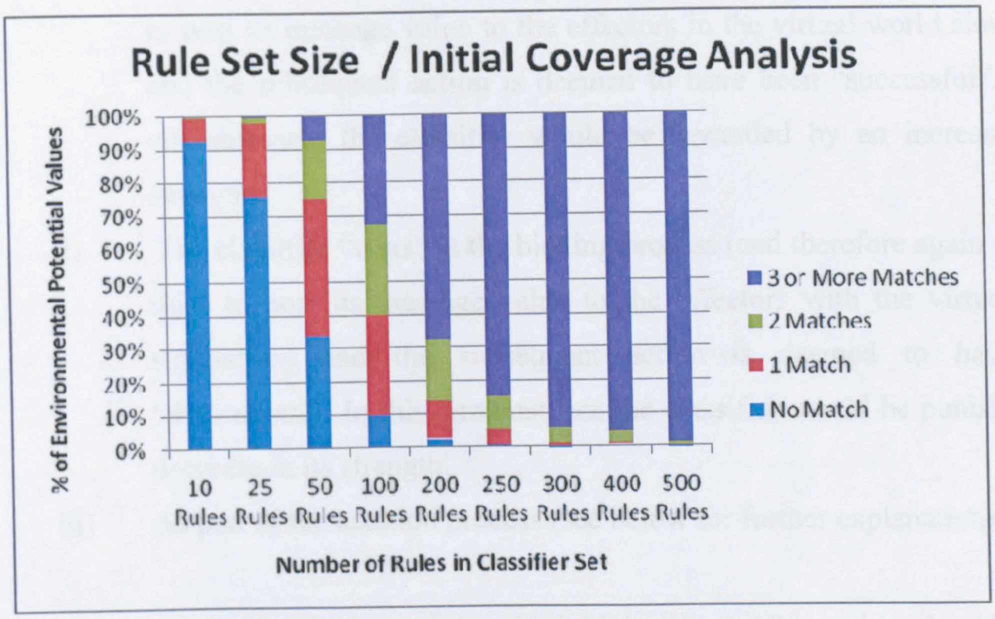


Figure 62 Rule Set Size and Expected Coverage Analysis

As a result of this experiment was that typically 200 – 250 rules were used in most of the experimental work. The next tuning parameters to consider are related to strength of the classifier and the tax system to be utilised by the prototype system.

6.6.2 Initial Fitness

Each classifier has an associated fitness / strength value. This value is of paramount importance to the classifier as it will be the significant element in deciding the following

i) The value of any bids that a classifier makes in the auction process. Therefore "weaker" classifiers are more likely to lose auctions to "stronger" classifiers.

ii) "Stronger" classifiers are proportionally more likely to be selected for breeding in the genetic algorithm process.

iii) "Weaker" classifiers are proportionally more likely to be selected for deletion from the population in the genetic algorithm process.

The strength of an individual classifier can be altered by the following circumstances

i) The classifier "wins" in the bidding process (and therefore wins the right to post its message value to the effectors in the virtual world simulation) and the subsequent action is deemed to have been "successful". In this circumstance, the classifier would be rewarded by an increase in its strength.

ii) The classifier "wins" in the bidding process (and therefore again wins the right to post its message value to the effectors with the virtual world simulation) and the subsequent action is deemed to have been "detrimental". In this circumstance the classifier would be punished by a decrease in its strength.

iii) As part of the taxation process (see below for further explanation)

A major consideration in determining the initial strength allocated to classifiers is to give each classifier sufficient strength to survive for a reasonable evaluation period. The initial strength needs to be considered in conjunction with the reward, punishment and taxation strategies used in the system.

To give a simple example of this, assume the following (ignoring taxation for the moment).

Initial_Strength = 10

Good_Action_Payoff = 10

Poor_ Action_Punishment = 10

In this particular example, a classifier that made a winning bid and made one "punishable" mistake will lose all its strength under the following coding logic

```
If <Classifier_Action> Moves FTime_Metric further from Goal_Value
        {
        Strength is diminished by Poor_Action_Punishment value
        }
```

A classifier without any strength will struggle to survive for any period, will not be selected for breeding and will not win an auction against any other classifier that has a strength value greater than zero.

Therefore the initial strength value (relative to reward, punishment and taxation strategies) must ensure the system the opportunity to evaluate classifiers sufficiently.

6.6.3 Bid Tax

The bid tax is a payment taken from all classifiers matching the environmental message that are subsequently invited to bid for the opportunity to post their message to the effectors. The impact of the bid tax is to adversely affect the strength of low specificity classifiers that are able to bid in response to a wide variety of environmental messages, as demonstrated in the following example.

A classifier has the following associated condition

-1	-1	-1	-1	1	0	-1	-1	-1	-1

This would mean the specificity of the classifier condition would be 2 (as there are 8 wildcard values out of 10 positions). This classifier would be able to match with 256 environmental messages.

This classifier is therefore potentially likely to enter the bidding process relatively frequently. The bid of a classifier is calculated using the specificity value as a multiplier and this means that this particular classifier is less likely to actually win a bid. The effect of the bid tax is to impose a penalty on over-general classifier conditions and therefore weaken their strength relative to other rules in the classifier set.

6.6.4 Life Tax

The life tax is a payment taken from all classifiers after each iteration of the program cycle responsible for receiving environmental messages and inviting classifiers to bid.

The reason for imposing a life tax is to penalise classifier rules that do not match environmental messages received from the environment and potentially never make a bid.

If the following classifier was generated

1	0	0	1	1	0	1	0	1	1

Specificity Value of 10

but subsequently failed to match an incoming environmental message (made more likely due to its high specificity) then a method of degrading its strength over time is needed, therefore making it less likely to survive and breed off-spring. A life tax taken from every classifier achieves this goal. As with all of the strength-affecting parameters it is vital to ensure that the correct balance is achieved.

If the tax life is set too high (relative to initial strength), potentially strong rules are lost as they have not had the chance to demonstrate their "fitness" yet. A life tax that is too low allows poor rules to survive longer than perhaps is desirable.

6.6.5 Proportion Select

The Proportion Select value relates to the proportion of classifiers that are replaced during the genetic algorithm function of the end of each learning cycle. The prototype system currently uses a proportion select rate of 0.20 (which equates to 20% of the classifiers being replaced at the end of each learning cycle).

By increasing the value of proportion select stronger classifiers are potentially bred more rapidly but at the risk of losing valuable classifier information that has not yet been able to demonstrate its "fitness". This may lead to the system becoming over-reliant on mutation to recover the lost genetic material.

An example of this element is explained below and considers only the condition element (in reality this would affect both the condition and message).

Index	Condition Element										Strength
0	0	1	0	0	1	0	0	0	1	0	300
1	-1	0	1	1	0	0	-1	1	0	0	250
2	-1	-1	0	0	1	0	0	0	-1	0	220
3	1	0	1	1	1	-1	-1	-1	0	0	400
4	0	-1	0	0	1	1	1	0	0	0	500
5	1	-1	1	0	1	1	1	0	0	0	450
6	0	1	1	0	0	1	1	1	1	1	200
7	-1	-1	1	1	0	0	-1	0	-1	1	190
8	1	1	1	0	0	-1	-1	-1	-1	1	210
9	1	1	0	0	1	1	1	0	0	1	240

Figure 63 Example of Proportion Select

Using a proportion select value of 0.5 (50% classifier replacement), it is perfectly possible (based on the selection process) that classifiers 2, 6, 7, 8 and 9 would be selected for replacement. Classifiers 6, 7, 8 and 9 are the only classifiers with a value of 1 in the final position of the condition element. The replacement classifiers are more likely to be bred from the stronger classifiers (none of which have 1 in the final position). Therefore, there is a chance that the 1 in the final classifier position is lost forever (unless mutation reintroduces a 1 in that position at some point in the future).

This loss may not have a long term adverse effect as perhaps it may indicate that it is not required to respond to the environment. However, this is a simple illustration of the potential impact of selecting a higher proportion of replacement classifiers and therefore should be carefully considered when selecting the proportion rate.

6.6.6 Crowding Factors and Mutation Rates

There is a danger in Learning Classifier Systems that the effect of the genetic algorithm leads to a convergence of similar condition and message sets as diversity is bred out of the population. This is primarily caused by stronger classifiers dominating the breeding process because of their relative strength in the rule set.

Miorandi *et al.* suggests that this is especially problematic in a dynamic environment as it leads to an inability to explore the search space effectively when the optimum changes (Miorandi, Yamamoto, & De Pellegrini, 2010). He further suggests that adaptive mutation changes are potentially a method to generate the required diversity after such a change.

The prototype system has some functionality that attempts to minimise the effect of premature convergence including:

Crowding is a technique based on work by De Jong to maintain diversity in the classifier population (De Jong, 1975). It works by selecting the most similar rule out of a sample of the population (which is then replaced). The effect of this is to maintain diversity as it enables relatively weaker rules that add diversity to the population to survive in the classifier set.

The earlier example, focussing entirely on the condition element is used again to demonstrate the concept of how crowding works in the prototype system. In the actual prototype system both condition and message are considered.

Index	Condition Element										Strength
0	0	1	0	0	1	0	0	0	1	0	300
1	-1	0	1	1	0	0	-1	1	0	0	250
2	-1	-1	0	0	1	0	0	0	-1	0	220
3	1	0	1	1	1	-1	-1	-1	0	0	400
4	0	-1	0	0	1	1	1	0	0	0	500
5	1	-1	1	0	1	1	1	0	0	0	450
6	0	1	1	0	0	1	1	1	1	1	200
7	-1	-1	1	1	0	0	-1	0	-1	1	190
8	1	1	1	0	0	-1	-1	-1	-1	1	210
9	1	1	0	0	1	1	1	0	0	1	240

If two classifiers are selected for breeding purposes (4 and 5) then a random crossover point is selected (e.g. 4).

Cross Point	0	1	2	3	4	5	6	7	8	9	Strength
4	0	-1	0	0	1	1	1	0	0	0	500
5	1	-1	1	0	1	1	1	0	0	0	450
Child 1	0	-1	0	0	1	1	1	0	0	0	475
Child 2	1	-1	1	0	1	1	1	0	0	0	475

Child 1 inherits the first 5 values of Classifier 4 and the final 5 values of Classifier 5. Child 2 inherits the first 5 values of Classifier 5 and the final 5 values of Classifier 4.

If only Child 1 is used for the purposes of this experiment, then three candidates are selected for replacement, if a crowding-factor value of 3 is used (for this example classifiers 7, 8 and 9 are used).

Each of the three candidates are compared to calculate which is the most similar to Child 1 (and is going to be introduced into the classifier population).

	0	1	2	3	4	5	6	7	8	9	Matches with Child 1
Child 1	0	-1	0	0	1	1	1	0	0	0	N/A
Classifier 6	0	1	1	0	0	1	1	1	1	1	4
Classifier 7	-1	-1	1	1	0	0	-1	0	-1	1	2
Classifier 8	1	1	1	0	0	-1	-1	-1	-1	1	1

Therefore, classifier 6 is the most similar to Child 1 and will be replaced by Child 1 in the population. The effect of this technique is to help to ensure an element of diversity remains in the population at all times. If crowding was not implemented the effect would be that over time one "species" of classifier would dominate the classifier rule-set.

The role of mutation in the success of genetic algorithms has often been seen as secondary to that of crossover. However, Kharbat states that it is *"well known that the mutation rate controls the diversity within the population which guides the search for the accurate solution"* (Kharbat, Bull, & Odeh, 2005).

As with much of the parameter setting in genetic algorithm approaches, mutation rate setting has been characterised as a trial and error approach to achieving an appropriate mutation rate. Lin suggested that probabilities of mutation are *"critical to the success of genetic algorithms"* and therefore the rates should themselves be automatically adjusted for optimum performance to be achieved (Lin, Lee, & Hong, 2003). This view of an adaptive mutation rate is also supported by Troc saying that *"only the adaptive mutation rate, which taken into account the current content of the classifier population, may support optimal search of the rule space"* (Troc & Unold, 2010).

The role of mutation in the prototype system is to potentially enable lost or non-existent population characteristics to be introduced into the classifier set. It also allows for serendipitous innovation, although this is balanced by the possibility of the fitness of a classifier being degraded. Mutation is the last resort of evolutionary backtracking from a blind alley. Mutation rates in the human DNA are typically 1 in

30,000,000 base pairs (Xue, et al., 2009). This equates to 100 – 200 new mutations from one generation to the next. Because of the size of the classifier chromosome strings it was obvious, as explained below, that a higher mutation rate would be required. The initial mutation rate for the prototype system was set as 0.001 for the initial experiments. Wilson's XCS system reported that higher mutation rates in the range of 0.01 – 0.05 are often used (Butz & Wilson, 2002) with some success.

In the prototype system, there is mutation of both the condition and the message elements of the classifier. Using 0.001 as the mutation rate and a rule set size of 100 classifiers would result in:

i) 10 (positions in condition) * 100 (rules) * 0.001 (mutation rate) = 1 mutation in the condition element

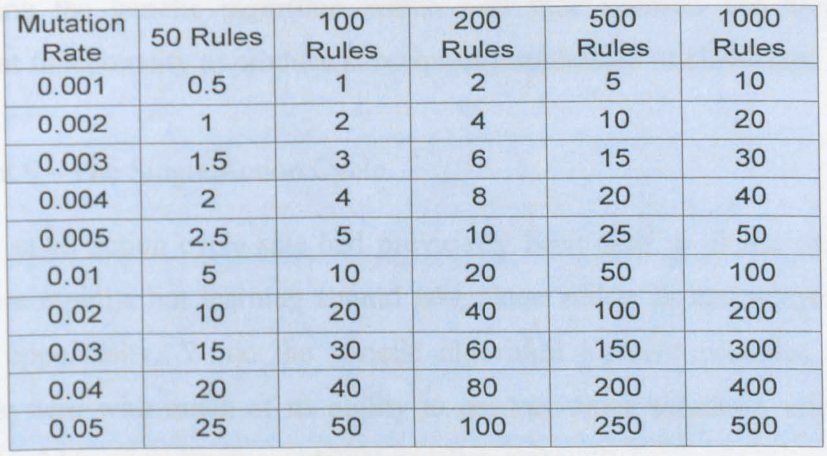
ii) 16 (positions in message) * 100 (rules) * 0.001 (mutation rate = 1.6 mutations in the message element

If we used 0.05 (as reported by Butz & Wilson) on the same rule set size of 100 classifiers we would see

i) 10 (positions in condition) * 100 (rules) * 0.05 (mutation rate) = 50 mutations in the condition element

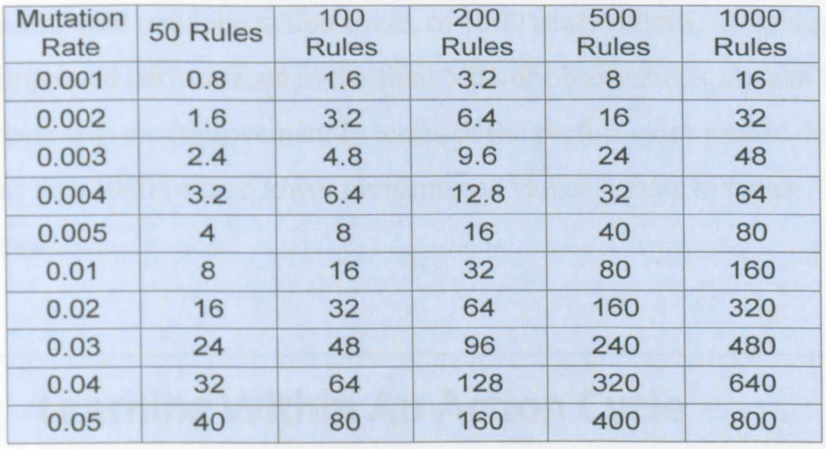
ii) 16 (positions in message) * 100 (rules) * 0.05 (mutation rate) = 80 mutations in the message element

A full table of classifier condition / rule-set mutations in relation to rule-set size and mutation rate are presented in Figure 64 and Figure 65.



Mutation Rate	50 Rules	100 Rules	200 Rules	500 Rules	1000 Rules
0.001	0.5	1	2	5	10
0.002	1	2	4	10	20
0.003	1.5	3	6	15	30
0.004	2	4	8	20	40
0.005	2.5	5	10	25	50
0.01	5	10	20	50	100
0.02	10	20	40	100	200
0.03	15	30	60	150	300
0.04	20	40	80	200	400
0.05	25	50	100	250	500

Figure 64 Mutation Rates / Rule Size in the Condition Element



Mutation Rate	50 Rules	100 Rules	200 Rules	500 Rules	1000 Rules
0.001	0.8	1.6	3.2	8	16
0.002	1.6	3.2	6.4	16	32
0.003	2.4	4.8	9.6	24	48
0.004	3.2	6.4	12.8	32	64
0.005	4	8	16	40	80
0.01	8	16	32	80	160
0.02	16	32	64	160	320
0.03	24	48	96	240	480
0.04	32	64	128	320	640
0.05	40	80	160	400	800

Figure 65 Mutation Rates / Rule Size in the Message Element

There is a requirement to choose a mutation rate that enables the classifier prototype to recover lost but potentially valuable alleles whilst guarding against a high mutation rate that may well mean that valuable characteristics are lost in the mutation process.

6.6.7 Actions Cycles and the Genetic Algorithm

ACTION_CYCLES refers to the number of FTime readings received and acted upon, which is effectively how the prototype learns the relative value of classifier rules, before firing the genetic algorithm element of the prototype system.

The number of ACTION_CYCLES has to be sufficient to allow classifiers to be given the opportunity to demonstrate their relative strength, weakness or applicability before firing the genetic algorithm which will then perform the breeding and replacement functionality to produce subsequent populations of classifiers.

Experiment 9 – The Single Action Cycle

Relatively small action cycle size had previously been used to obtain experimental results more rapidly, but learning should take place within an action cycle if given sufficient opportunity. While the genetic algorithm element provides a learning classifier system with much of its ability to produce fitter solutions, reinforcement learning should have an impact within the action cycle.

The experiment used one long action cycle, of 1000 observations, in the hope of observing improved performance in the final 50% of observations. As can be seen in Figure 66 there was an improvement in terms of the performance metric during observations 501-1000 (where lower performance values equate to better performance).

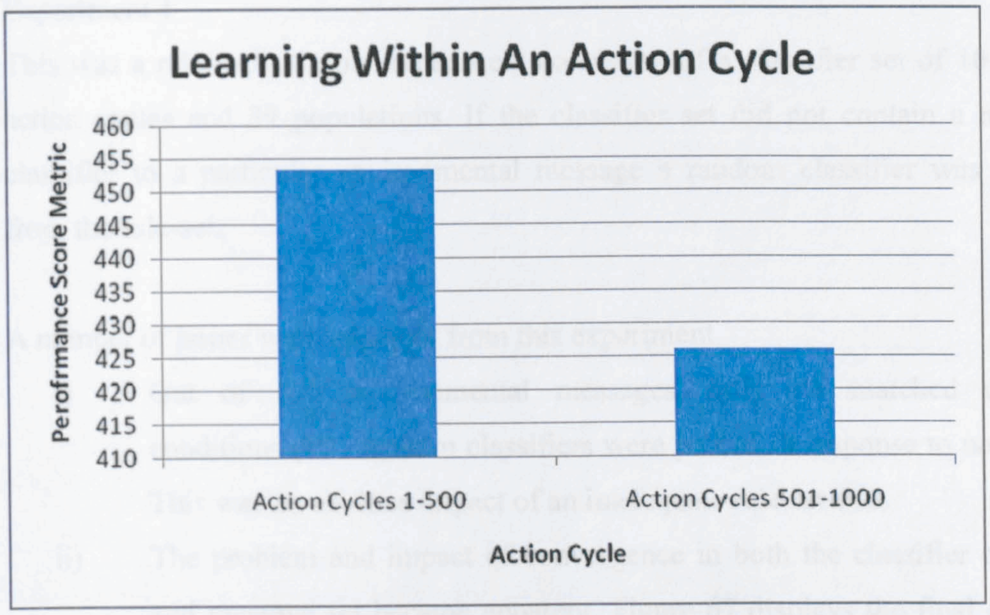


Figure 66 Experiment 9 Example of Learning Within A Single Action Cycle

It was apparent from this experiment that it would be necessary to give the classifier set an action cycle long enough to allow learning to take place before enacting the genetic algorithm function to select the classifiers to breed and delete from the population.

6.7 Strategies For No Match

The outline of the prototype assumes there will be a classifier to match all environmental messages. A conceptual challenge was the development of a robust approach to how the prototype system would handle a situation where an environmental message could not be matched to a classifier. In proposing adaptive systems as an approach to surviving in a changing and unknown environment, then it was obvious that the prototype system would require an appropriate strategy to handle this particular state.

Two approaches were evaluated and subsequently discarded. These were:

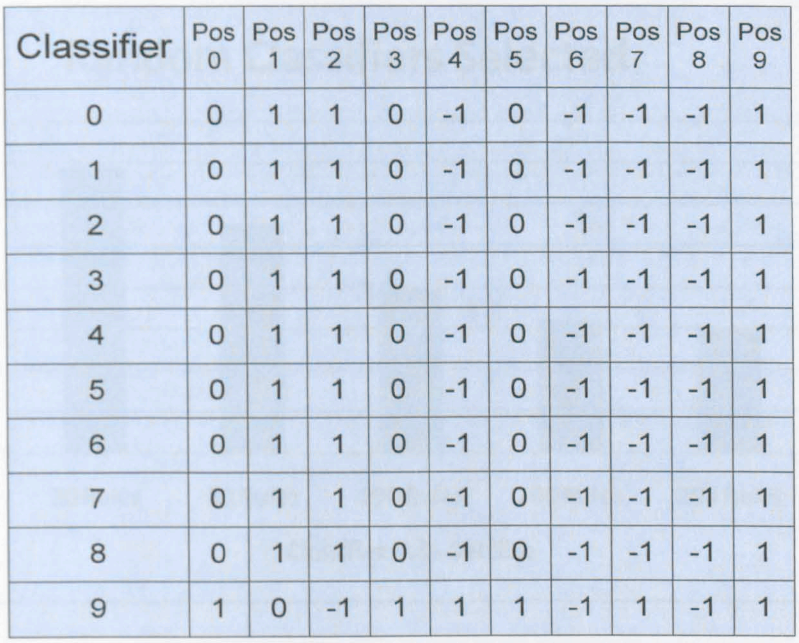
1) The selection of a random classifier
2) The selection of the nearest matching classifier based on the closeness of the condition element (specificity quantifier outline design in Figure 160 and Figure 161).

Experiment 1

This was a relatively simple experiment consisting of a classifier set of 10 rules, 5 action cycles and 39 populations. If the classifier set did not contain a matching classifier to a particular environmental message a random classifier was selected from the rule-set.

A number of issues were apparent from this experiment

i) Out of 195 environmental messages, only 24 matched classifier conditions (171 random classifiers were picked in response to no match). This was an obvious impact of an inadequate rule set size.
ii) The problem and impact of convergence in both the classifier condition and message set became apparent. Figure 67 displays the final classifier set and demonstrates how 9 out of 10 condition classifiers are identical (the crowding factor protects Classifier no. 9). This was replicated in the message set of these classifiers, which were also identical (barring Classifier no. 9)
iii) Mutation was not taking place due to a low mutation rate combined with a small rule-set size.



Classifier	Pos 0	Pos 1	Pos 2	Pos 3	Pos 4	Pos 5	Pos 6	Pos 7	Pos 8	Pos 9
0	0	1	1	0	-1	0	-1	-1	-1	1
1	0	1	1	0	-1	0	-1	-1	-1	1
2	0	1	1	0	-1	0	-1	-1	-1	1
3	0	1	1	0	-1	0	-1	-1	-1	1
4	0	1	1	0	-1	0	-1	-1	-1	1
5	0	1	1	0	-1	0	-1	-1	-1	1
6	0	1	1	0	-1	0	-1	-1	-1	1
7	0	1	1	0	-1	0	-1	-1	-1	1
8	0	1	1	0	-1	0	-1	-1	-1	1
9	1	0	-1	1	1	1	-1	1	-1	1

Figure 67 Experiment 1 Example of Convergence in the Classifier Set

Experiments 2 – 6

Given the high proportion of randomly selected classifiers apparent in experiment 1, a further set of experiments were performed with varying rule-set sizes. The rule-sets sizes included 20, 50, 100 and 200 to observe the reduction in the requirement to select a random classifier from the rule-set.

The results of these experiments can be observed in Figure 68.

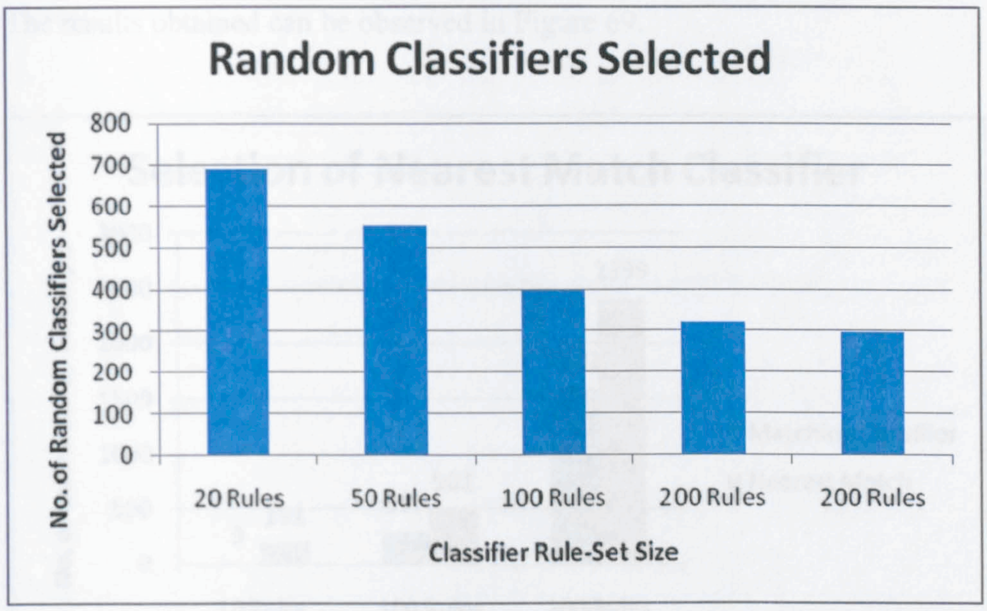


Figure 68 An Analysis of the Random Classifier Selection Experiments 2 - 6

It was apparent from the results outlined in Figure 68 that the number of random classifiers being selected was too high. An approach could have been simply to keep increasing the rule-set size but convergence would mean that even large rule-sets would eventually lose their ability to adequately "cover" the environmental messages being received from the virtual world environment.

The whole approach of the selection of a random rule in the event of no match seemed fundamentally incorrect as an approach. Rules were being randomly selected and then were subsequently punished or rewarded depending on their ability to manage the environment. Therefore a more "refined" approach was taken during the next set of experimentation.

As the selection of a random classifier seemed flawed as a general approach it was necessary to develop an alternative method of classifier selection in the absence of a matching classifier.

The next set of experiments consisted of using a condition quantifier to select a nearest match classifier. This negated the requirement for a random classifier but required a method for quantifying a classifier condition. This was made difficult by the occurrence of wildcard values in the condition element of the classifier.

The results obtained can be observed in Figure 69.

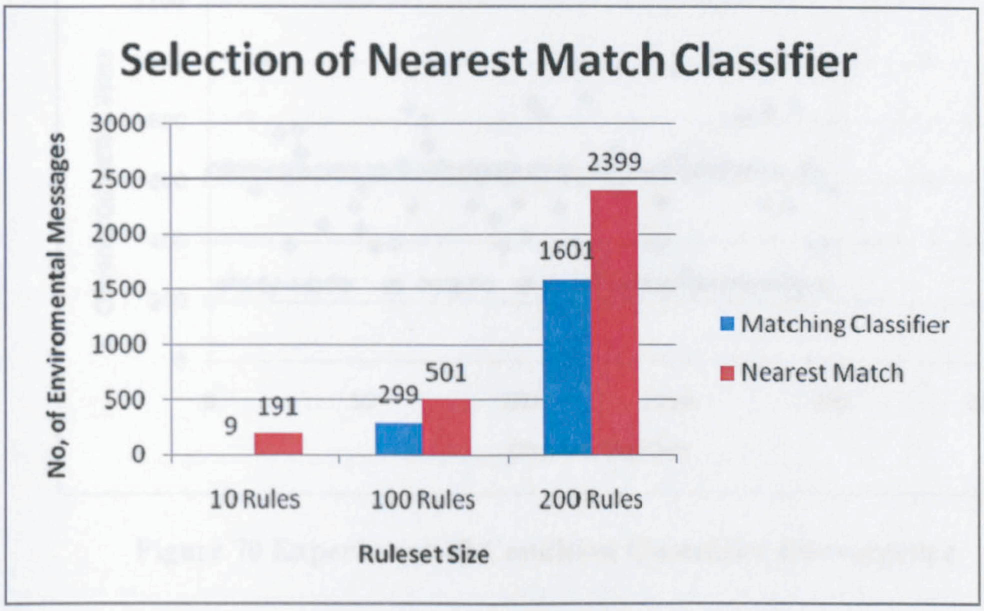


Figure 69 Selection of Nearest Match Classifier Analysis

While it seemed more appropriate to select a close match classifier than a random classifier, the results above demonstrate that the approach used thus far could not be said to be producing an environmental model due to the lack of matching classifiers. The 10 rule-set example only found a matching classifier on 4.5% of total observations whereas the 100 rule-set experiment could only find a matching classifier in 37.4% of environmental messages. The 200 rule-set size could only find a matching classifier on 40.2% which was perhaps lower than expected given the increased coverage provided by a rule-set of this size.

This was, in part, due to the convergence apparent in the rule-set. Figure 70 demonstrates how the convergence occurred in experiment 10. Two main values developed, namely 253 and 627, which can be observed in the graph. This would mean that increasingly nearest match classifier would become less useful as the population developed and converged.

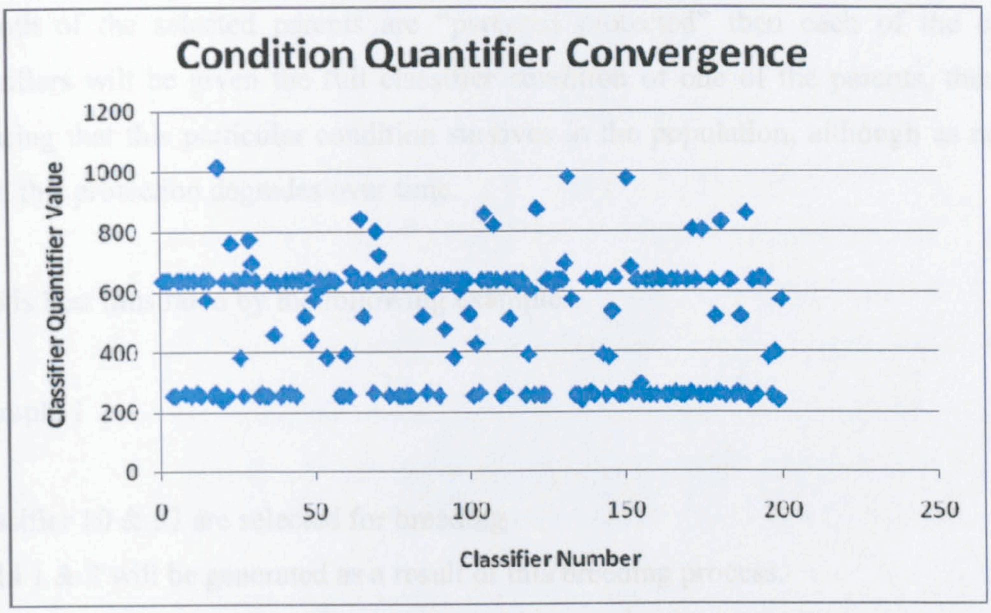


Figure 70 Experiment 10 Condition Quantifier Convergence

It became obvious from these results that an alternative approach would be required to resolve the issue of environmental messages without a matching classifier in the rule-set.

6.7.1 The parent protection approach

An approach to solve the issue was the concept of parent protection, which was used within the prototype.

If it is assumed that in the early stage of the learning cycle an environmental message is received and a matching and subsequently winning classifier exists then a parent protection flag is initialised for that particular classifier.

If classifier breeding is handled within the prototype then the two children produced from any two parent classifiers are a random hybrid of both the condition and messages of both parents.

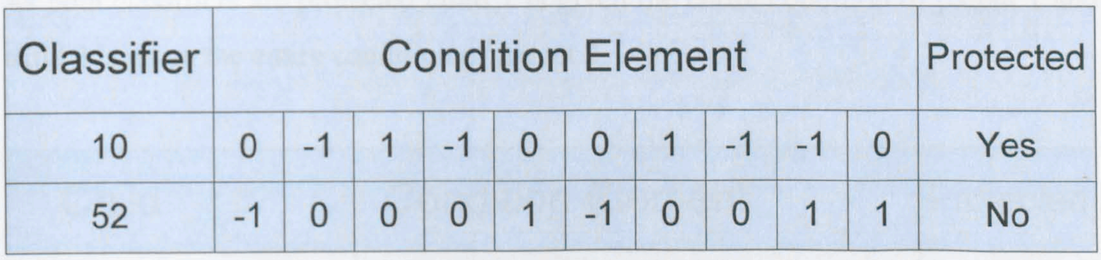
If one of the selected parents is "parent protected" then one of the child classifiers is given the full classifier condition of that particular parent with the second child being generated in the usual manner.

If both of the selected parents are "parented protected" then each of the child classifiers will be given the full classifier condition of one of the parents, thereby ensuring that this particular condition survives in the population, although as noted later, this protection degrades over time.

This is best illustrated by the following examples.

Example 1

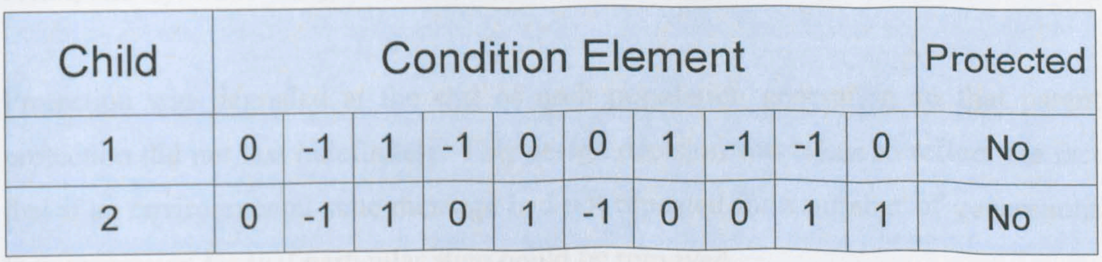
Classifier 10 & 52 are selected for breeding
Child 1 & 2 will be generated as a result of this breeding process.



Classifier	Condition Element										Protected
10	0	-1	1	-1	0	0	1	-1	-1	0	Yes
52	-1	0	0	0	1	-1	0	0	1	1	No

Figure 71 Parent Protection Example

As classifier 10 is protected child 1 is given its entire condition whereas child 2 is given a hybrid condition (at breakpoint 3 in this particular example).



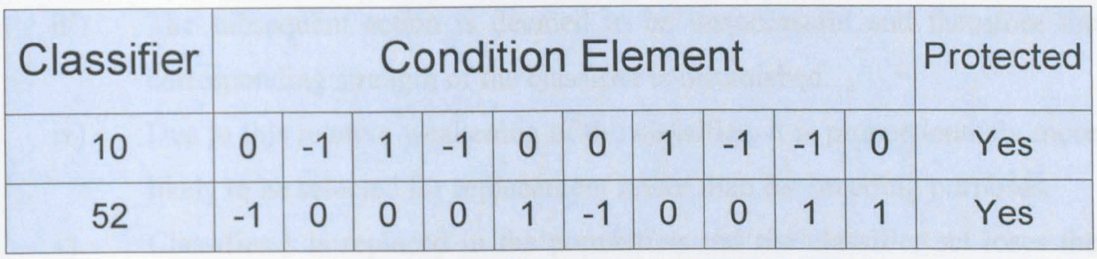
Child	Condition Element										Protected
1	0	-1	1	-1	0	0	1	-1	-1	0	No
2	0	-1	1	0	1	-1	0	0	1	1	No

This was implemented so that classifiers that match environmental conditions previously were given a breeding advantage (in terms of the condition element only). The result of this would be to breed classifiers that were known to match environmental messages being received by the prototype system.

Example 2

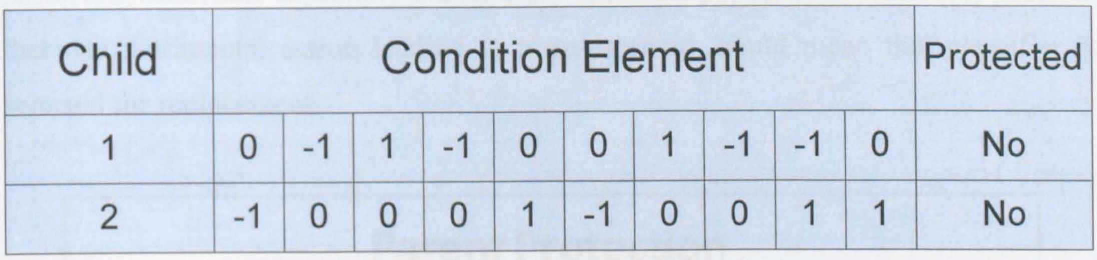
Classifier 10 & 52 are selected for breeding

Child 1 & 2 will be generated as a result of this breeding process.



Classifier	Condition Element										Protected
10	0	-1	1	-1	0	0	1	-1	-1	0	Yes
52	-1	0	0	0	1	-1	0	0	1	1	Yes

As both classifiers are protected child 1 is given the entire condition of parent 1 and child 2 is given the entire condition of parent 2.



Child	Condition Element										Protected
1	0	-1	1	-1	0	0	1	-1	-1	0	No
2	-1	0	0	0	1	-1	0	0	1	1	No

It is important to note that this protection exists only on the condition element of the breeding process and that the message element is generated in the usual random breakpoint hybrid method.

Protection was degraded at the end of each population generation so that parent protection did not last indefinitely. This design decision was taken to reflect the fact that if an environmental state message had not repeated for a number of generations then protection for that particular state could be removed.

One weakness of this approach was that parent protection was only used if the classifier in question was selected for breeding purposes and did not protect the classifier from replacement.

This weakness is illustrated by

i) Classifier 1 is the only classifier that matches the environment message and therefore is allowed to post its message to the effectors.

ii) Parent protection is then flagged for this particular classifier.

iii) The subsequent action is deemed to be unsuccessful and therefore the corresponding strength of the classifier is diminished.

iv) Due to this relative weakening of the classifier, it is proportionately more likely to be selected for replacement rather than for breeding purposes.

v) Classifier 1 is replaced in the population and the classifier set loses the ability to match that particular environmental message in subsequent generations.

Experiments showed that often parent protection was an improvement in terms of protecting classifiers especially during early classifier populations when it is possible that one detrimental action leading to a punishment, could mean that classifier is selected for replacement.

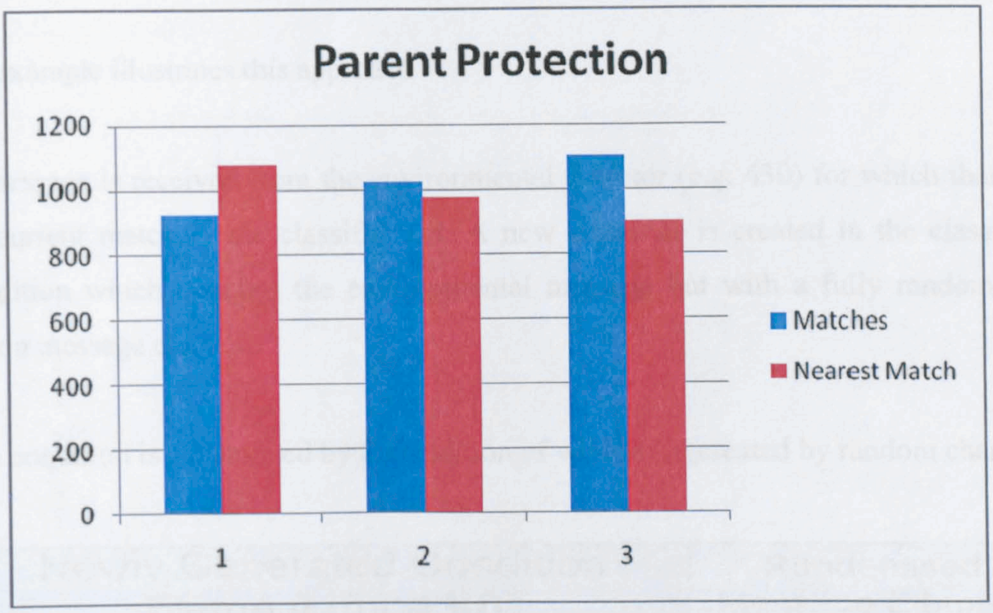


Figure 72 Experiment 15 Parent Protection (100 Rule-Set)

It became apparent from these experiments that the performance of parent protection (implemented to increase the number of exact matching classifiers) was relatively consistent at achieving approximately 50% matches in a series of 100 rule-set experiments. Previously approaches with this size rule-set had achieved a 33% match success rate.

Parent protection was a useful addition to the prototype system and strengthened the ability to obtain an environmental match but further enhancement was required in order to enhance the prototype.

6.7.2 The covering approach

Wilson's XCS classifier system used the concept of covering classifiers to generate a new classifier matching the environmental condition in the event of no match. The basic approach of covering is that when a message is received from the environment and no match exists, then a new classifier is generated and inserted it into the population at that point.

Bull suggests that it is useful to generalise the condition by introducing a random number of wildcards into it.

An example illustrates this approach

A message is received from the environmental detector (e.g. 430) for which there is no current match in the classifier set. A new classifier is created in the classifier condition which matches the environmental message but with a fully randomised action message element.

The condition is generalised by the addition of wildcards (created by random chance)

Newly Generated Condition Element (for 430)										Randomised Message Element
0	1	1	0	1	0	1	1	1	0	0111000011110011
The condition is then generalised by the addition of wildcards using random function										
0	-1	1	0	-1	-1	1	1	1	0	0111000011110011

The newly created classifier replaces one of the lower performing classifiers in the rule set and is given an initial strength of the average strength of the full rule-set. This average strength is necessary to give the classifier a chance of survival in a highly fit rule-set.

This created an issue when a new classifier is created to match the environmental message but it subsequently turns out to be a poor rule. This is perfectly likely as the message is created randomly and the rule is likely to be a candidate for replacement, which may lead to detrimental performance.

The decision taken was to introduce three matching classifiers into the set. By randomly creating three classifiers the chance of a "good" rule being created is increased, which can subsequently flourish in future generations and therefore reduce the longer term requirement for the generation of further covering classifiers.

The prototype used a combination of both "parent protection" and covering to enhance its ability to match environmental messages.
This meant that the whole concept of nearest match and classifier "quantification" could be removed from the prototype system. The results obtained were as follows:

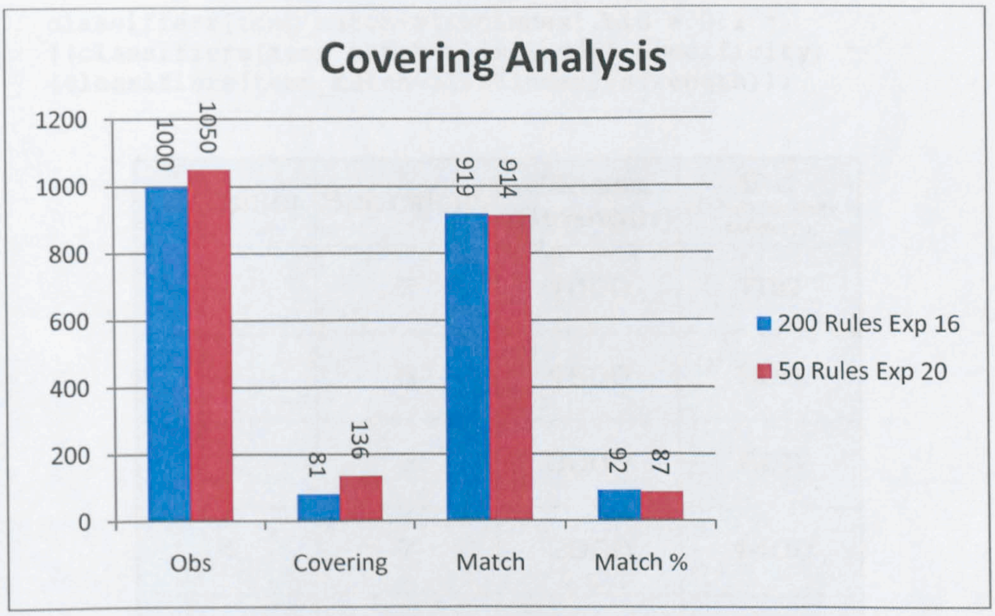


Figure 73 Analysis of Covering

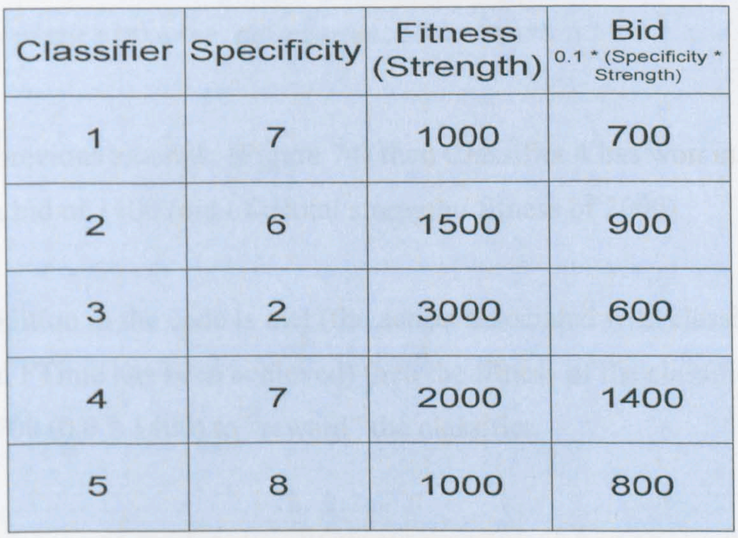
The results obtained demonstrated the success of the combination of parent protection and covering in terms of producing classifier rule-sets that could match environmental messages. The prototype, during the later experiments, produced three covering classifiers. The rationale behind this approach was that it increased the probability of a "good" action set being generated. This, in turn, would increase the probability of at least one of the covering classifiers surviving and flourishing in the population rule-set.

6.8 The Genetic Algorithm Fitness Function

The fitness (or strength) of a learning classifier provides it with the attribute required to both survive and increases the likelihood of it being selected as a parent by the genetic algorithm. The fitness of a learning classifier was determined by its ability to move the current FTime closer to the goal required by the prototype.

The value of increased fitness is best explained by illustrating how a classifier is selected during the bidding process. If it is assumed that five classifiers match the environmental message and therefore "bid" to become the classifier whose action is subsequently taken by the prototype system. The bid is calculated by the following code

```
classifiers[temp_match->listindex].bid = 0.1 *
((classifiers[temp_match->listindex].specificity) *
(classifiers[temp_match->listindex].strength));
```



Classifier	Specificity	Fitness (Strength)	Bid 0.1 * (Specificity * Strength)
1	7	1000	700
2	6	1500	900
3	2	3000	600
4	7	2000	1400
5	8	1000	800

Figure 74 Bid Calculation

Figure 74 emphasises the importance of fitness (strength) on the likelihood of a classifier being selected to have its actions enacted by the prototype system.

6.8.1 How does the fitness (strength) of a classifier change?

There are three ways in which the fitness of a classifier is changed by the prototype systems. When a classifier "wins" in the bidding process its associated action is taken by the prototype systems and there are four potential outcomes illustrated in the code below.

```
if (post_action_ftime == environmental_goal)
            {
            printf("\nGreat, it hit the goal!...") ;
            bid_payment += 0.5 * classifier_array[bid_winner].bid ;
            classifier_array[bid_winner].strength += bid_payment ;
            environment_account -= bid_payment ;
            bid_payment = 0.0 ;
            }

      if(comparison_1 < comparison_2)
            {
            printf("\nIt made it worse!\n") ;
            bid_payment += 0.5 * classifier_array[bid_winner].bid ;
            classifier_array[bid_winner].strength -= bid_payment ;
            environment_account += bid_payment ;
            }
      else if(comparison_1 > comparison_2)
            {
            printf("\nIt made it better!...") ;
            bid_payment += 0.5 * classifier_array[bid_winner].bid ;
            classifier_array[bid_winner].strength += bid_payment ;
            environment_account -= bid_payment ;
            }
      else
            {
            printf("\n*** No change... ***\n") ;
            }
```

If we use the previous example (Figure 74) then Classifier 4 has won in the bidding process with a bid of 1400 (out of a total strength / fitness of 2000).

If the first condition in the code is met (the action associated with classifier 4 has meant the goal FTime has been achieved) then the fitness of the classifier is increased by 700 (0.5 * 1400) to "reward" the classifier.

If the second condition in the code is met (the action associated with classifier 4 has moved the FTime further away from the goal) then the fitness of the classifier is decreased by 700 (0.5 * 1400) to "punish" the classifier.

If the third condition in the code is met (the action associated with classifier 4 has moved the FTime towards the goal) then the fitness of the classifier is increased by 700 (0.5 * 1400).

If the fourth condition in the code is met (the action associated with classifier 4 has not made a difference – no movement towards or away from the goal) then no change is made to the fitness of the classifier.

If should be noted that it is possible that an action which hits the goal FTime and moves the FTime closer to the goal will be rewarded twice (conditions 1 & 3) and therefore receive, in this example, an increase in fitness of 1400 (700 + 700).

The Life tax can alter the fitness of a classifier as follows:

After each action cycle a life tax is deducted from the strength of all classifiers. The impact of life tax is to degrade the fitness of classifiers that are not bidding and therefore not matching messages received from the environment.

```
void  impose_life_tax(CLASSIFIER classifiers[MAXCLASS],
float *env_account)
{
int index ;

for(index = 0; index < MAXCLASS; index++)
   {
    classifiers[index].strength -= life_tax ;
   *env_account += life_tax ;
   }
}
```

6.9 The Bidding Process and Effecting Change in the Environment

A variety of bidding strategies were tested when deciding which of the matching classifiers would be allowed to post their message to the virtual world simulator effectors.

The standard approach for bid calculation was as follows

Bid = 0.1* (Classifier_Strength * Classifier_Specificity)

This would have the effect of favouring strong classifiers with high specificity and therefore making them more likely to prevail in the bidding process. Although a number of alternative bidding strategies were evaluated during the development process.

In the event of duplicate winners (two or more matching classifiers making the same bid) a random classifier from the duplicate winners was chosen as the winning classifier.

6.9.1 Measuring classifier success and subsequent payoff strategies

A variety of success criteria and subsequent payoff strategies were used during the testing and evaluation of the prototype and an interesting element was how the learning classifier system would exploit any small advantage in the payoff strategy.

One example of this was when bid tax was removed and therefore bidding classifiers, which seldom won the subsequent auction were not taxed beyond the life tax that all classifier rules are subject to. The effect of this taxation is to discourage over generalised classifiers. Once this tax was removed the classifier rule set conditions became less specific as the advantage was swayed in favour of bidding as often as possible in the hope of winning an auction every so often.

A second example was when the specificity quantifier experiment was being tested as an approach it became problematic to "specify" the quantifier due to the wildcard element.

An approach was to reward specificity in the payoff strategy. An example of this can be seen below:

```
 if (classifier_array[bid_winner].specificity ==7)
     {
     classifier_array[bid_winner].strength += 20;
     }
if (classifier_array[bid_winner].specificity ==8)
     {
     classifier_array[bid_winner].strength += 30;
     }
if (classifier_array[bid_winner].specificity ==9)
     {
     classifier_array[bid_winner].strength += 40;
     }
if (classifier_array[bid_winner].specificity ==10)
     {
     classifier_array[bid_winner].strength += 50;
     }
```

It was hoped that by adding a specificity bonus payment to the payoff strategy more specific rules would be encouraged to develop. However, it quickly became apparent that the net result of this approach was that the more specific classifiers were being bred but at the expense of strong rules of the message element. Therefore, poor performing rules were promoted which adversely affected the prototypes ability to manage the simulator environment goal.

It emphasised the sensitivity of the payoff strategy and how any competitive advantage is exploited to full effect by a learning classifier system / genetic algorithm approach.

A number of different strategies were developed, an example of one such strategy can be seen below

```
if (post_action_ftime - environmental_goal > 0.0)
```

```
     {
     bid_payment -= (post_action_ftime -
     environmental_goal) * 10 ;
     }
else
     {
     bid_payment -= (environmental_goal -
     post_action_ftime) * 10 ;
     }
```

This approach sought to compare where the FTime was with respect to the goal as compared to the previous approach and pay a reward to the classifier strength on that basis.

After much experimentation, it was decided that perhaps the complexity of the payoff strategy was becoming a hindrance rather than beneficial and that a simpler approach was required.

The final system used a payoff / punishment tariff system, which although less sophisticated, lowered the chance that a slight competitive advantage present in the logic would skew the classifiers rule set created over subsequent populations.

6.9.2 Effecting change within the Second Life Simulator

Once a classifier has been selected it is allowed to post its message to the effectors in the virtual world simulator. This is achieved using a combination of PHP scripts, an XML-RPC broadcaster situated in the virtual world and the listening functionality of the scripted objects in the region simulator.

Each classifier message consists of sixteen states (either on or off) and is broadcast into the region via an XML-RPC broadcaster on a specified channel to the entire region. The scripted objects are constantly scanning the specified channel for any messages relating to that particular object. They can interpret the commands to start or stop and will enact the required command.

The winning classifiers full action set was written to a text file by the prototype system and a single PHP call then read the text file and sent the full 16 character length string (of 0's and 1's) into the virtual world region using the XML-RPC broadcaster. A scripted object was developed on the simulator that listened for the 16 character length string and subsequently read each character and broadcast them individually into the region.

6.10 The Genetic Algorithm

The genetic algorithm element of the prototype system provides the fundamental strength behind the use of the Learning Classifier System to produce a model of the environment and allowing exploration of the search space to take place.
The genetic algorithm function is called at the end of the action cycle (or learning element) and is primarily responsible for the generation of new and hopefully fitter classifiers and selecting the classifiers to be removed from the existing set.

6.10.1 The selection and mating of classifier condition and action sets

At the end of each action cycle there is a requirement to generate a proportion (as defined by PROPORTIONSELECT rate) of new classifiers that will replace the same proportion of classifiers in the existing classifier set. Effectively, from a biological view, this seeks to assure the survival of the fittest through reproduction and as Zielger suggests *"the choice of an appropriate mating partner is the most important prerequisite to assure the fitness of the off-spring"* (Ziegler, Santos, Kellermann, & Uchanska-Ziegler, 2009).

The underlying logic behind this process is to select two parent classifiers (from which two child classifiers will be generated) from the current classifier population. This is performed using a roulette wheel approach where the chance of being selected is proportional to the strength of the classifier.

Therefore, a classifier with a strength of 1000 is ten times more likely to be selected as a potential parent than a classifier with a strength of 100. The result of this is that over time stronger performing classifiers will be chosen as parents and therefore the overall fitness of the classifier set will improve over time.

The next stage is to randomly generate cross-over points for both the condition (0 to 9) and message element (0 to 15) of the selected classifiers. Once this is decided, the parent classifiers are split at the crossover point and two hybrid children are created.

The theoretical underpinning behind this suggests that the hybrids of two strong parents may actually surpass the parents in terms of performance during the next learning cycle.

The crossover process because of its nature does not provide any new information, as Wu *et al.* states *"once the entire population possesses the same value at a particular bit position, only mutation is able to insert new values of that bit into the population"* (Wu, Lindsay, & Riolo, 1997). Therefore the genetic algorithm process requires both crossover and mutation to produce fit rules with the ability to navigate away from search space blind alleys.

The children inherit an average of the strength of the two parent classifiers to give them a chance to flourish in the subsequent population.

6.10.2 Who lives and who dies?

If new and hopefully fitter classifiers are created then it is necessary to select the same proportion of the current rule-set for replacement and therefore effectively death. This is achieved by taking a random selection of classifiers from the current population and selecting the classifier with the lowest strength as a potential candidate for replacement.

Once a number of potential candidates have been selected, these are compared to the new classifier (on a position by position basis) and the most similar classifier is selected for replacement by the newly created child. The reason for this is to maintain variety in the population by allowing potentially weaker classifiers to survive due to the genetic diversity they bring to the population and has been discussed in more detail in the earlier section on crowding.

6.11 Summary

The development of the EMMA prototype required a number of experiments to be conducted to calibrate the initial conditions and heuristics such as: the rule set size, payoff strategies, taxation, mutation, strategies for when there were no matches.

It emphasised the number of decisions that designers have to consider when developing such systems. There are a number of tuning parameters that can significantly affect performance in particular application domains and it does appear to be an "art" involving significant trial and error.

For this research project, the tuning element was not a major concern as the prototype was seeking to demonstrate the applicability of learning classifier systems and genetic algorithms to provide models of environmental and capability for use by the prototype system. However, perhaps interesting developmental work could be in terms of the adaptive tuning of these parameters to obtain optimum results.

The prototype fulfils the major design considerations of reacting to environmental change by developing a current model view of both the environment and current capability of the system.

A fuller evaluation of the results from a variety of experiments using the prototype system will be considered in the following chapter of the thesis.

"Results! I have gotten a lot of results. I know several thousand things that won't work"

Thomas A. Edison

Chapter 7: Results and Evaluation

This thesis has sought to use cybernetics as a guiding principle for the design of the prototype system. The essence of the prototype is summed up by a desire to "satisfy" classical cybernetic thinking that both

"Every good regulator of a system must be a model of that system"
Ashby's Law of Requisite Variety
(Conant & Ashby, 1970)
&
"In order to adequately compensate perturbations, a control system must "know" which action to select from the variety of available actions"
Aulin's Law of Requisite Knowledge
(Aulin, 1982)

However, the research performed in this thesis suggests that these two principles may be further refined to:

"Every good regulator of a system must be a current and evolving model of that system"

"In order to adequately compensate perturbations, a control system must "know" which action to select from the variety of available actions and be able to evolve new actions in response to as yet unknown environmental conditions"

The fundamental question would be whether the prototype system could be successful in satisfying the requirement for self-adaptive systems to evolve both their external model of the environment and internal model of capability to satisfy the enhanced requirement statements outlined above.

Figure 75 illustrates how a learning classifier system approach can be used to provide a current and evolving representation of the internal model of capability and external model of the environment. The internal model of capability is provided by the message action element whilst the external model of the environment is represented by the condition element of the learning classifier. The nature of genetic algorithms means that these models can evolve in response to changing environmental conditions encountered by the system.

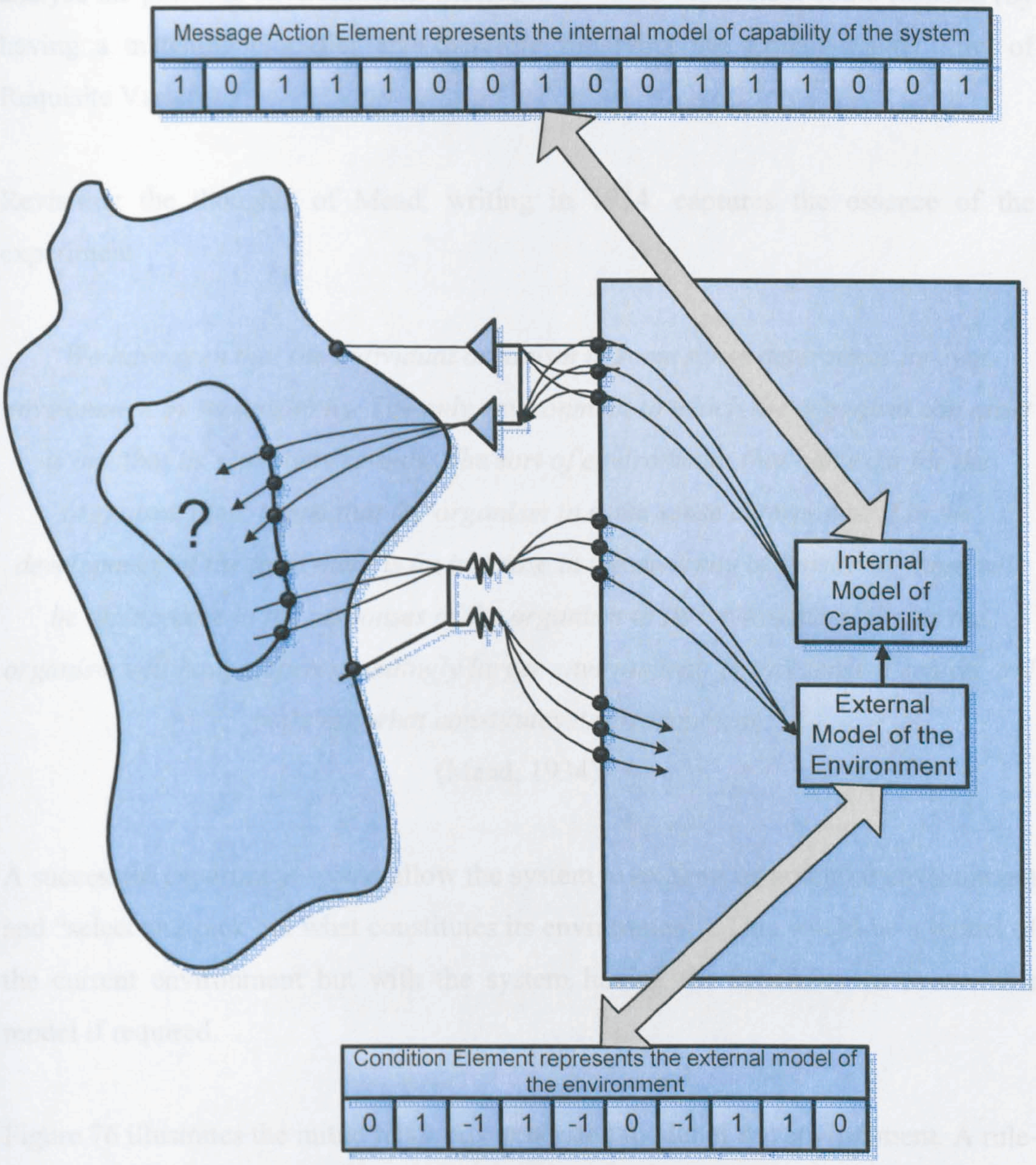

Message Action Element represents the internal model of capability of the system
1 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1
?
Internal Model of Capability
External Model of the Environment
Condition Element represents the external model of the environment
0 1 -1 1 -1 0 -1 1 1 0

Figure 75 VSM System 4 & Learning Classifier System Comparison

7.1 Demonstration of an Evolving External Model of the Environment

A method for an analysis of the environmental model was developed that could analyse the potential environmental messages to which the system could respond (by having a matching classifier and therefore satisfying the Conant-Ashby Law of Requisite Variety).

Revisiting the thoughts of Mead, writing in 1934, captures the essence of the experiment

> *"We have seen that the individual organism in some sense determines its own environment by its sensitivity. The only environment to which the organism can react is one that its sensitivity reveals. The sort of environment that can exist for the organism, then, is one that the organism in some sense determines. If in the development of the form there is an increase in the diversity of sensitivity there will be an increase in the responses of the organism to its environment, that is the organism will have a correspondingly larger environment. In this sense it selects and picks out what constitutes its environment."*
>
> (Mead, 1934).

A successful experiment would allow the system to explore its potential environment and "select and pick out what constitutes its environment". This would be a model of the current environment but with the system having the capability to evolve this model if required.

Figure 76 illustrates the initial randomly generated model of the environment. A rule-set of 100 was used, which as determined in earlier experimentation would give reasonably effective coverage of the search space in this particular application. It can be seen that the prototype system initially has no "informed" view of its environment but rather a view of the potential environment of the system. To satisfy the requirements outlined above, the prototype must be able to model the environment but also evolve this model as required in response to changing environmental conditions.

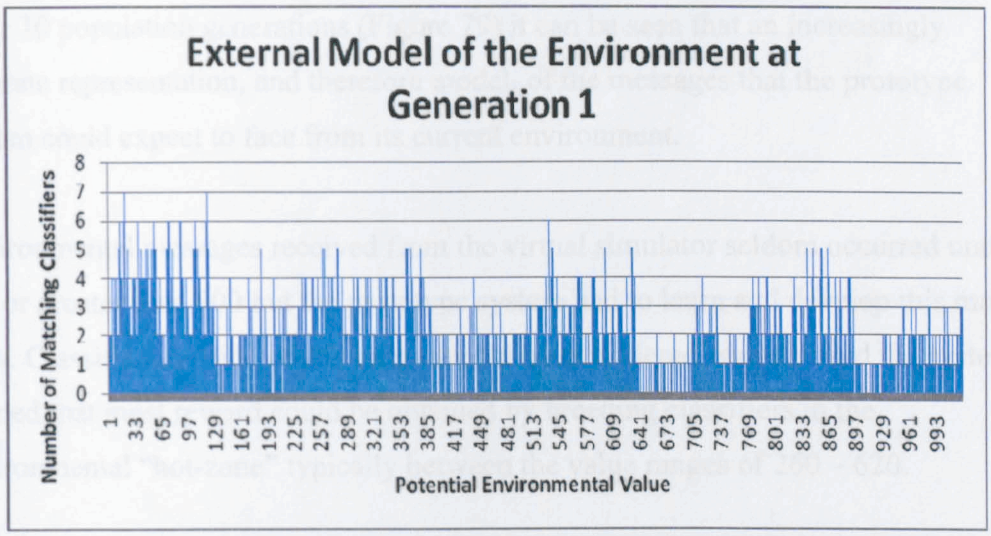


Figure 76 External Model – Generation 1

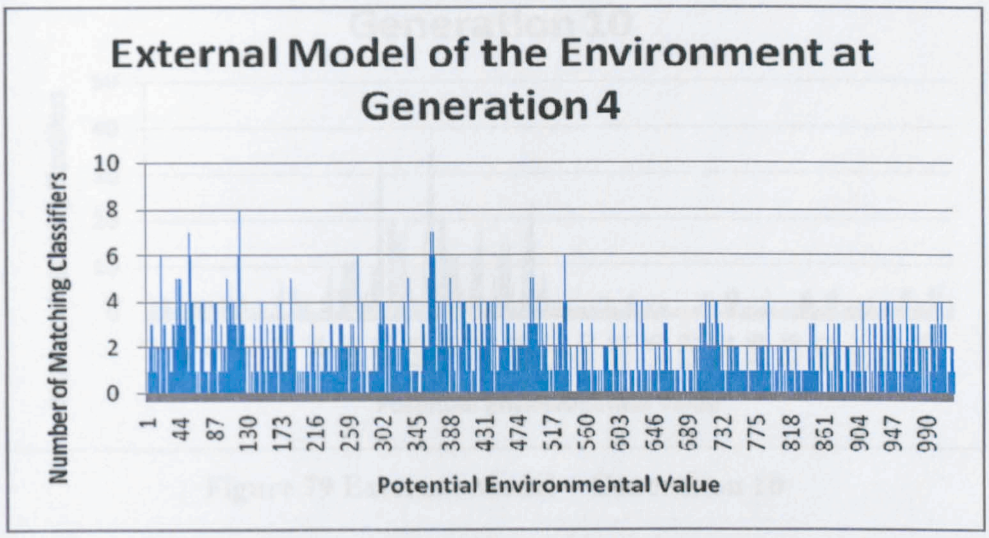


Figure 77 External Model – Generation 4

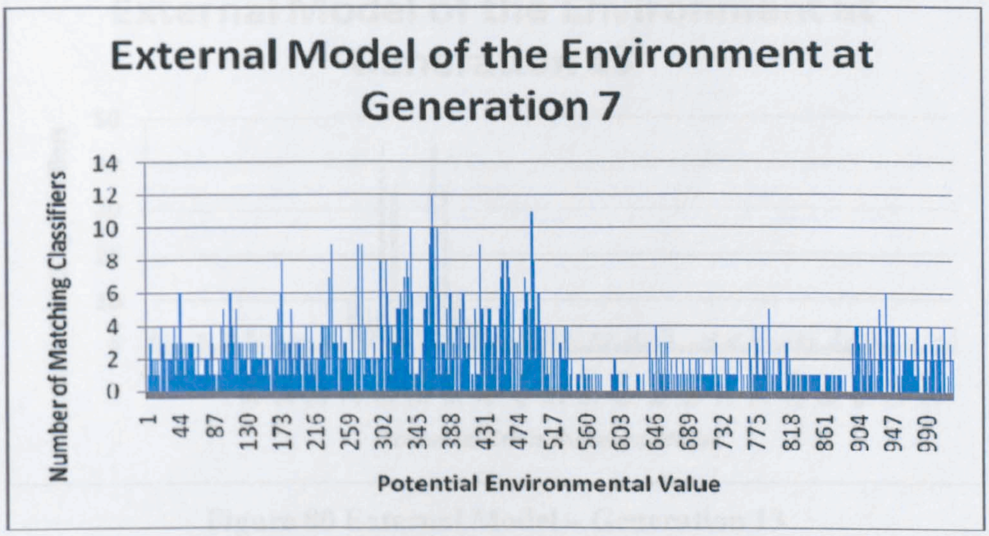


Figure 78 External Model – Generation 7

After 10 population generations (Figure 79) it can be seen that an increasingly accurate representation, and therefore model, of the messages that the prototype system could expect to face from its current environment.

Environmental messages received from the virtual simulator seldom occurred under 200 or greater than 800 but the prototype system had to learn and develop this model view. Classifiers outside of this range would be considered wasteful and the system learned that most reward could be obtained by breeding classifiers in the environmental "hot-zone" typically between the value ranges of 260 – 620.

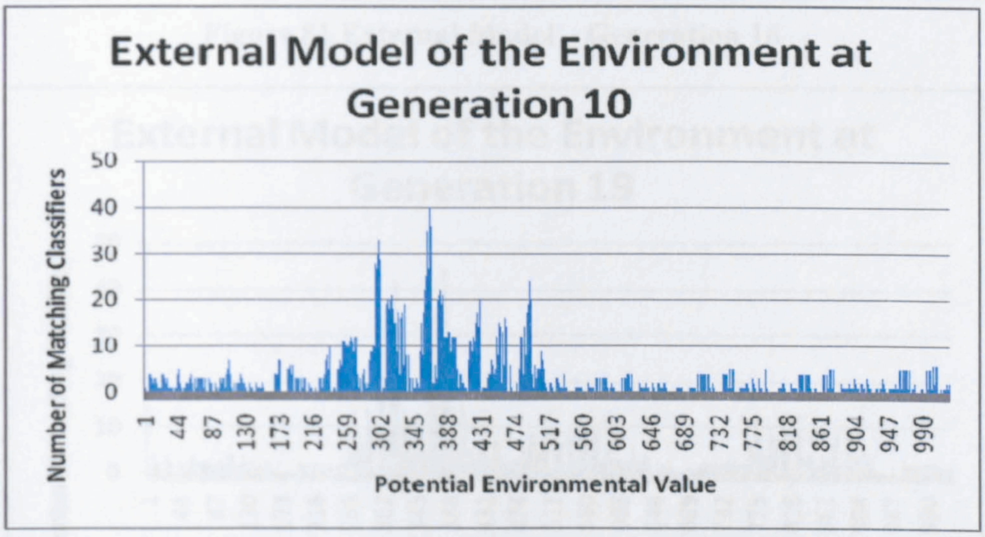


Figure 79 External Model – Generation 10

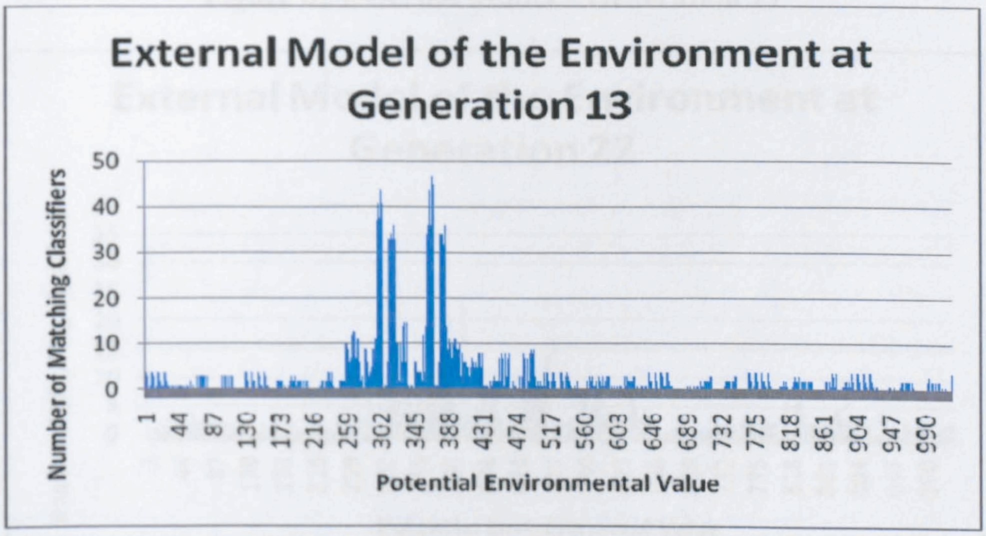


Figure 80 External Model – Generation 13

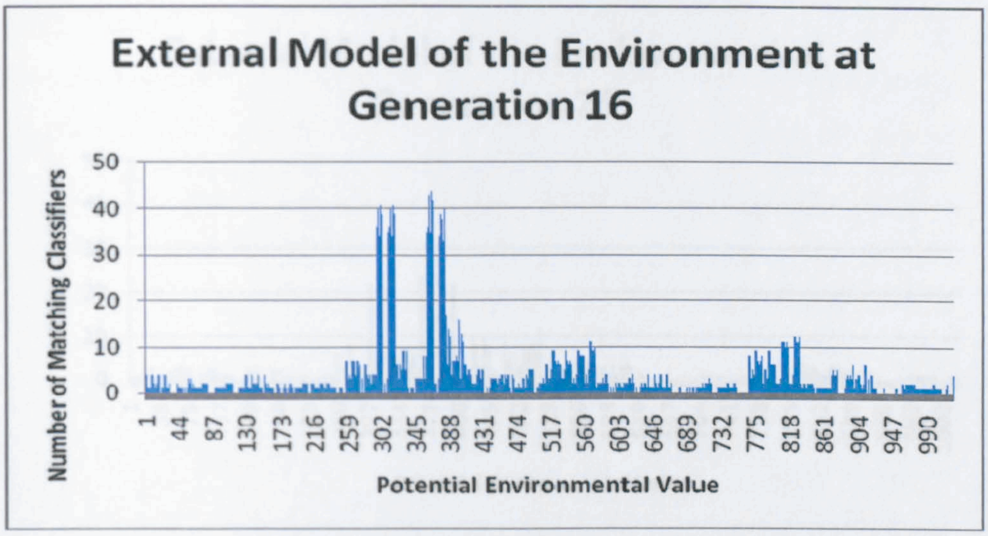


Figure 81 External Model – Generation 16

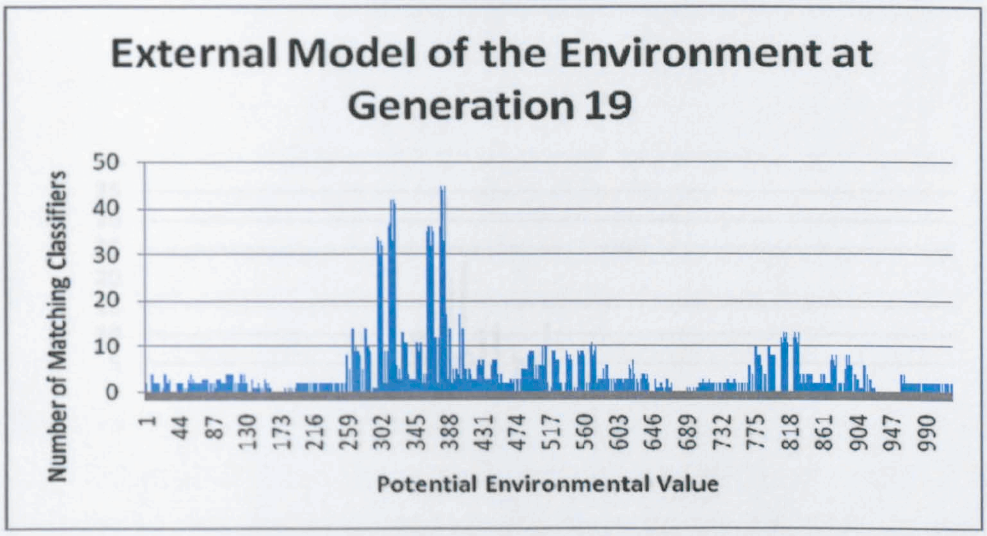


Figure 82 External Model – Generation 19

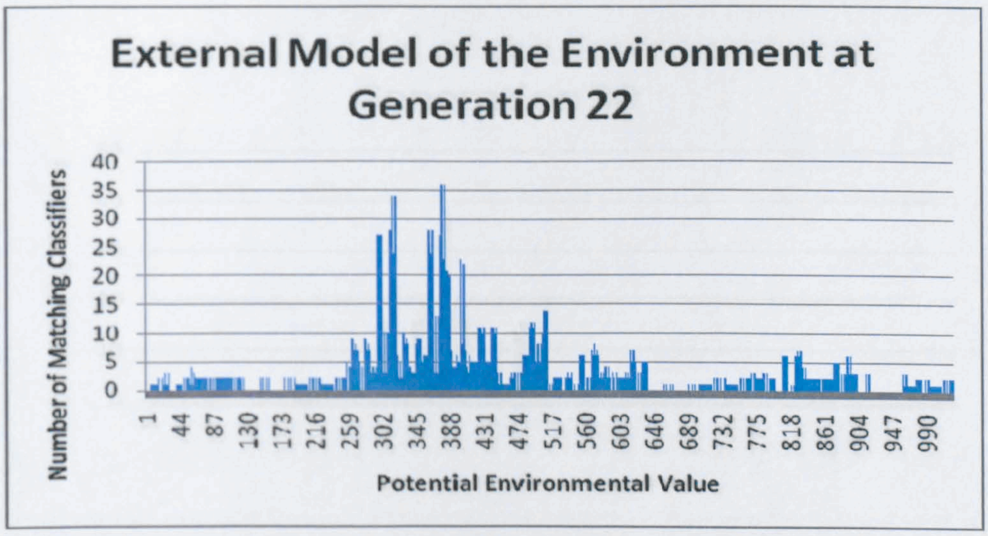


Figure 83 External Model – Generation 22

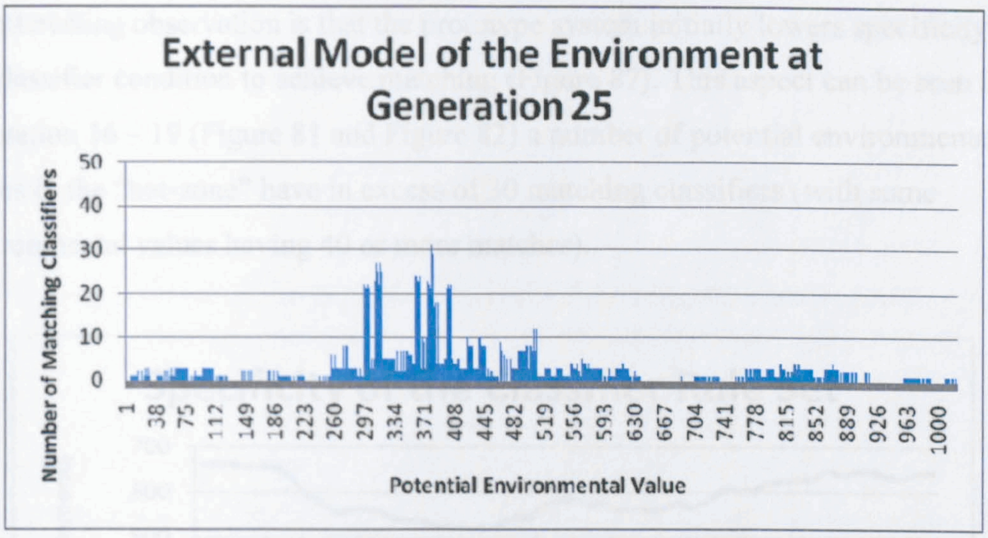


Figure 84 External Model – Generation 25

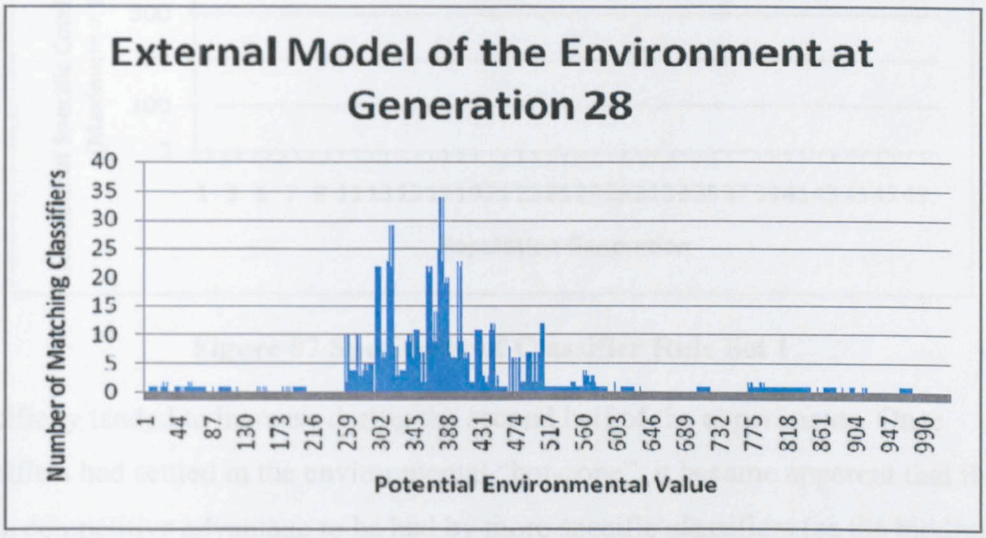


Figure 85 External Model – Generation 28

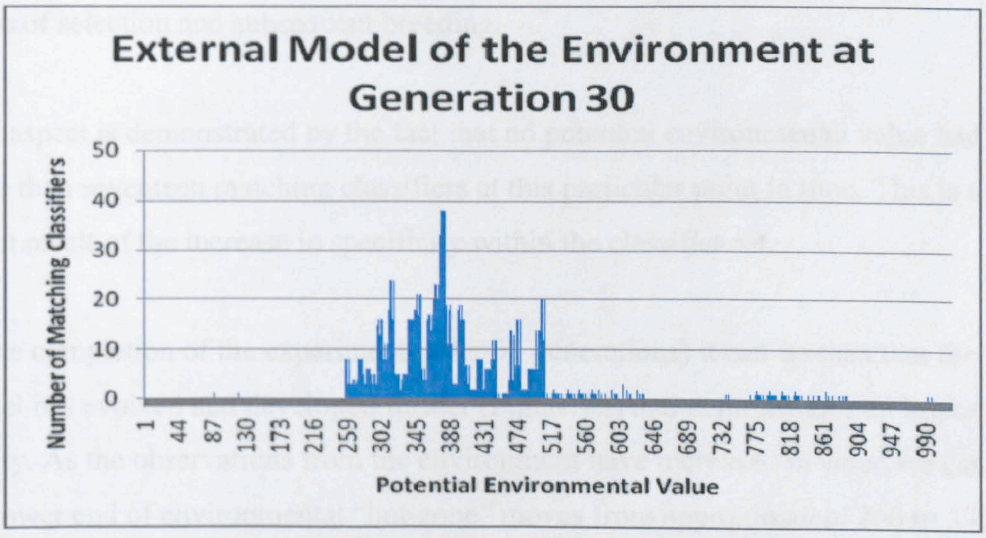


Figure 86 External Model – Generation 30

An interesting observation is that the prototype system initially lowers specificity in the classifier condition to achieve matching (Figure 87). This aspect can be seen in generation 16 – 19 (Figure 81 and Figure 82) a number of potential environmental values in the "hot-zone" have in excess of 30 matching classifiers (with some environmental values having 40 or more matches).

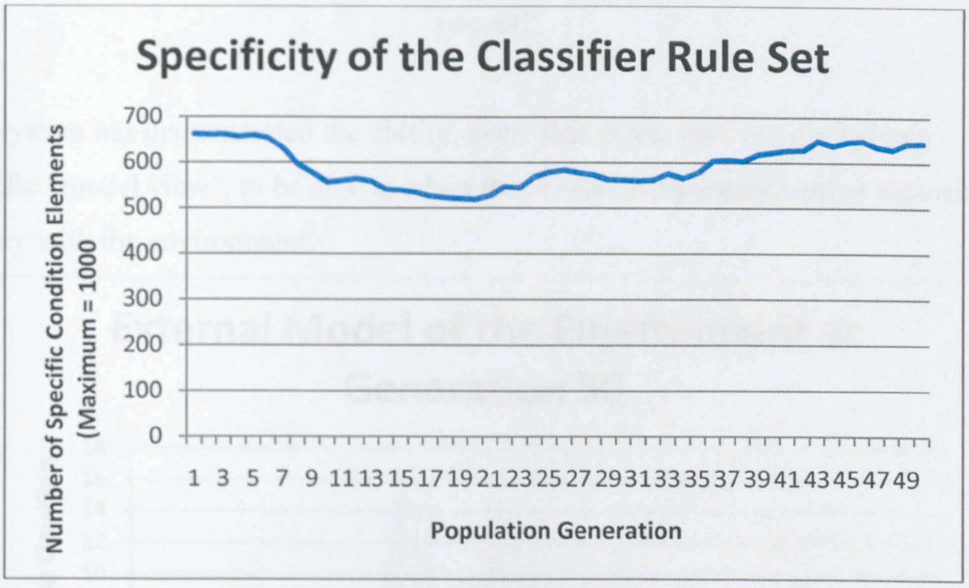


Figure 87 Specificity of Classifier Rule Set 1

Specificity tended to increase during the second half of the experiments. Once classifiers had settled in the environmental "hot-zone", it became apparent that there was a competitive advantage to be had by more specific classifiers (as the bidding process favoured specificity). Therefore, more specific classifiers were favoured in terms of selection and subsequent breeding.

This aspect is demonstrated by the fact that no potential environmental value had more than seventeen matching classifiers at this particular point in time. This is a direct result of the increase in specificity within the classifier set.

At the completion of the experiment (after 50 generations) it can be seen that the model has evolved and developed further (Figure 88) and demonstrates an important ability. As the observations from the environment have increased in value we can see the lower end of environmental "hot-zone" moves from approximately 260 to 330.

The model has adjusted to replicate this move in environmental trend and therefore maintain model currency with respect to current observations.

This demonstrates our extended Law of Requisite Variety that

> *"Every good regulator of a system must be a current and evolving model of that system"*

The system has demonstrated the ability, even after it has evolved a relatively specific "model view", to be able to adapt this model to maintain current requisite variety with the environment.

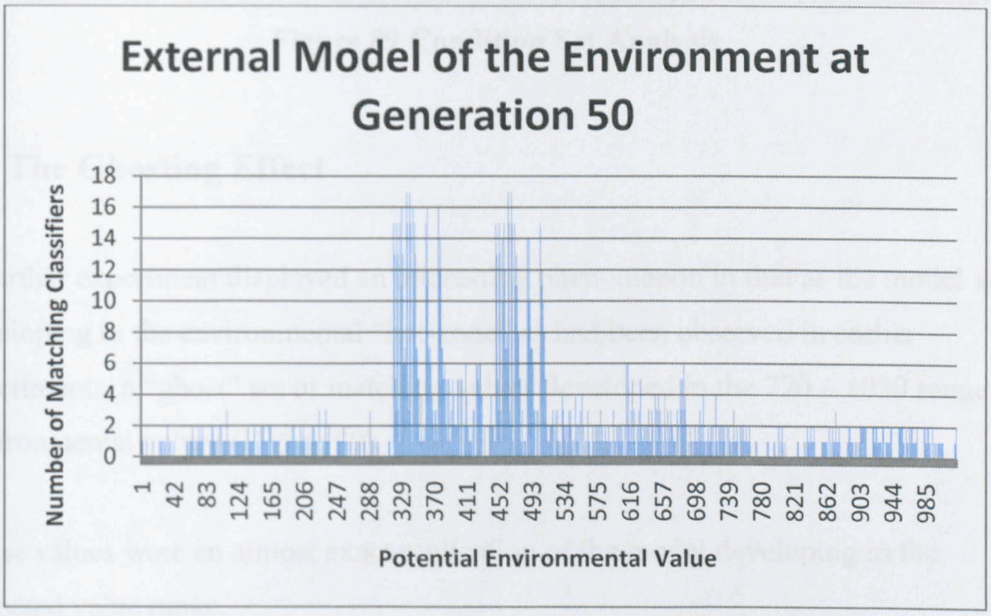


Figure 88 Environmental Model – Generation 50

An analysis of the condition set (Figure 89) shows that the prototype is breeding out 1's in position 0 and 9. Position 0 equates to 512 in the binary representation of the environmental value and is a reflection of the scarcity of environmental values over 520, which would necessitate the use of a "1" in this position. Position 9 equates to 1 in the binary representation of the environmental value and therefore would only be required if the environmental value was an odd number value. However, the design of this particular experiment (using only even numbers) means that the environment detectors do not observe odd numbered environmental values and the classifier set quickly evolves, learning that there is no value in a "1" in position 9 of the condition element.

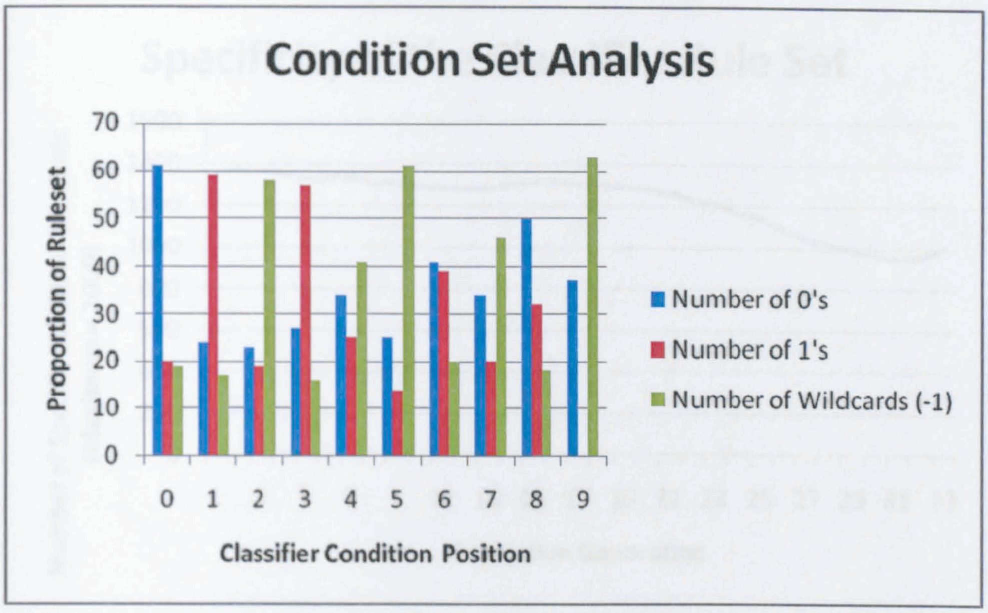


Figure 89 Condition Set Analysis

7.2 The Ghosting Effect

A further experiment displayed an interesting phenomenon in that as the model was developing in the environmental "hot-zone" as had been observed in earlier experiments, a "ghost" set of matching values developed in the 770 – 1050 range of environmental values (Figure 90).

These values were an almost exact replication of the model developing in the expected value range.

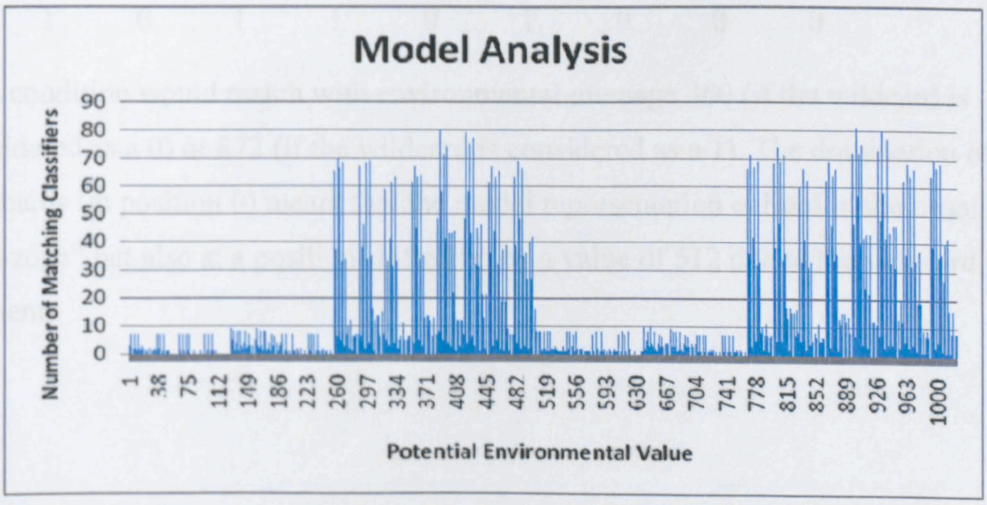


Figure 90 The Ghost Environmental Model

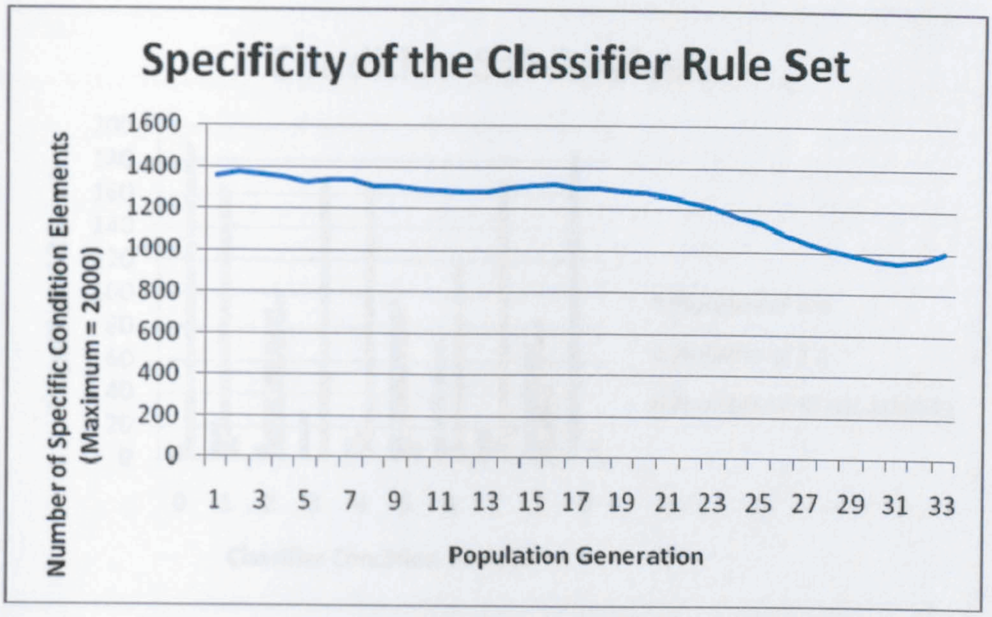


Figure 91 Specificity of the "Ghost Model"

The reason for the "ghost" model can be explained in Figure 91 and Figure 92. As the overall specificity of the classifier rule-set lowers, position 0 (representing 512 in the binary representation of the condition element) had become dominated by the wildcard or "don't care" value (-1). This would lead to a "secondary" shifted ghost representation of the environment.

An example of this would be if we have a highly specific classifier with the condition element consisting of

512	256	128	64	32	16	8	4	2	1
-1	1	0	1	1	0	1	0	0	0

This condition would match with environmental message 360 (if the wildcard is considered as a 0) or 872 (if the wildcard is considered as a 1). The domination of wildcards (in position 0) meant that the model representation existed in the usual "hot-zone" but also at a position shifted up by a value of 512 due to the wildcard element.

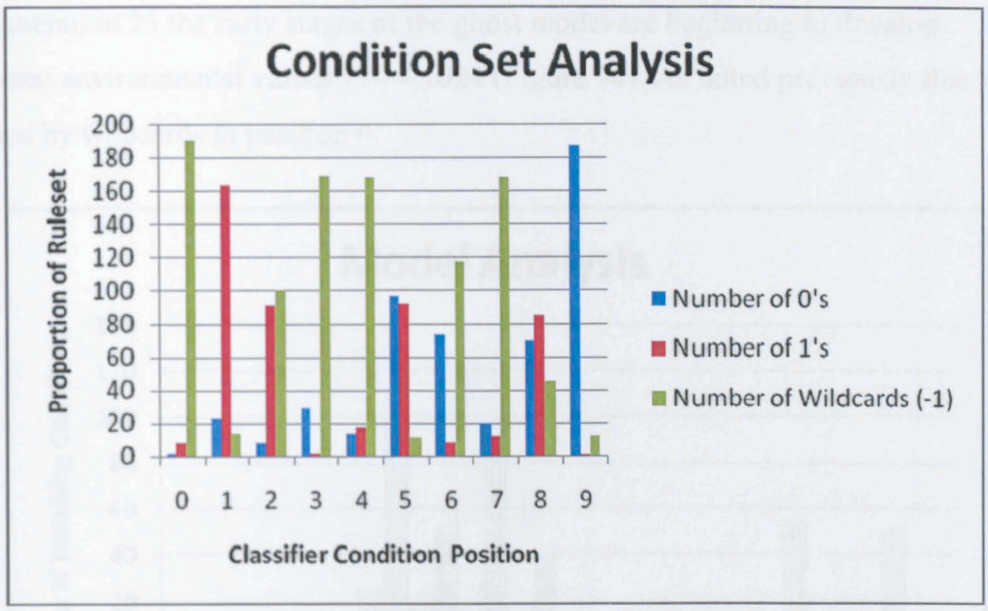


Figure 92 Ghost Model Analysis of Final Condition Set

The experiment demonstrating the ghosting phenomenon had only run for a relatively small number of generations (32) Earlier experiments had observed an increase in specificity during later generations and therefore the experiment was repeated over 50 generations using the same parameters as used in the previous "ghost" experiment.

After 12 generations, the model is developing a reasonable environmental model with the majority of the matching classifiers in the "hot-zone" (Figure 93).

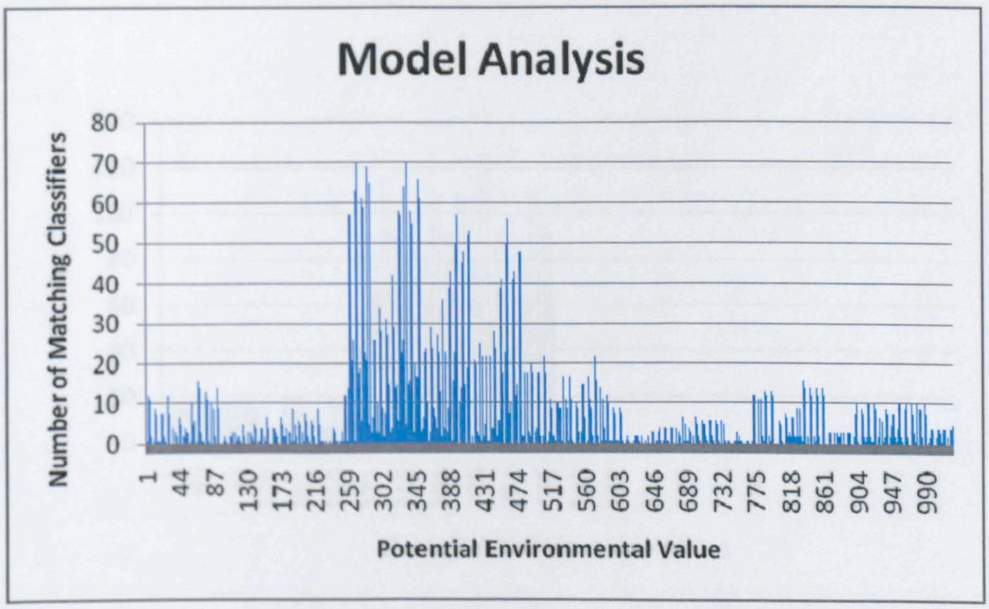


Figure 93 Environmental Model 2 - 12 Generations

At generation 25 the early stages of the ghost model are beginning to develop between environmental values 770 – 1024 (Figure 94). As noted previously this was caused by wildcards in position 0.

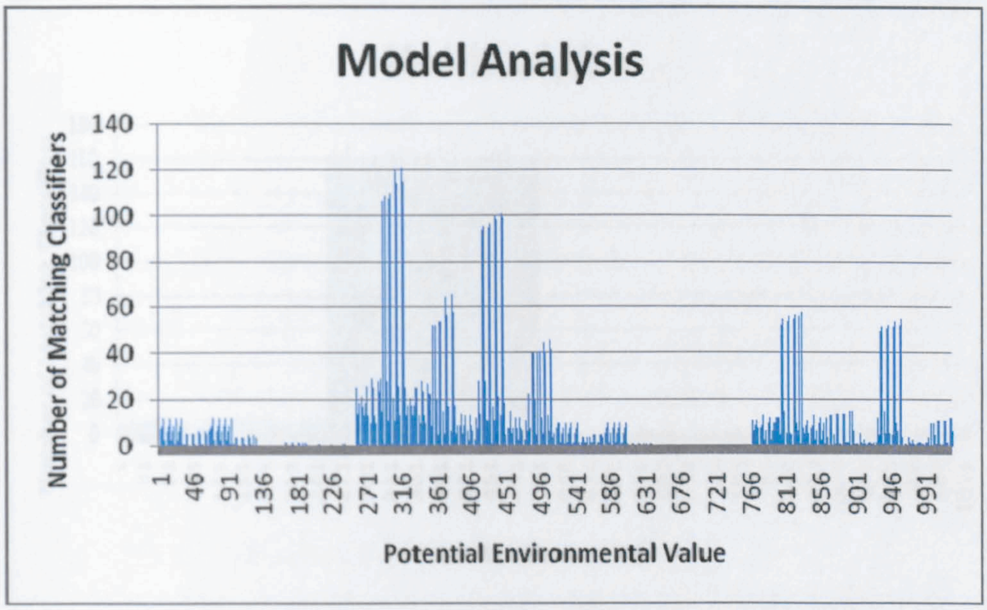


Figure 94 Environmental Model 2 – 25 Generations

At generation 38, wildcards (at condition position 0) have been increasingly bred out of the population and therefore the ghost element has reduced significantly. This is a result of high numbers of classifiers "competing" for selection within the "hot-zone" and higher specificity as a result.

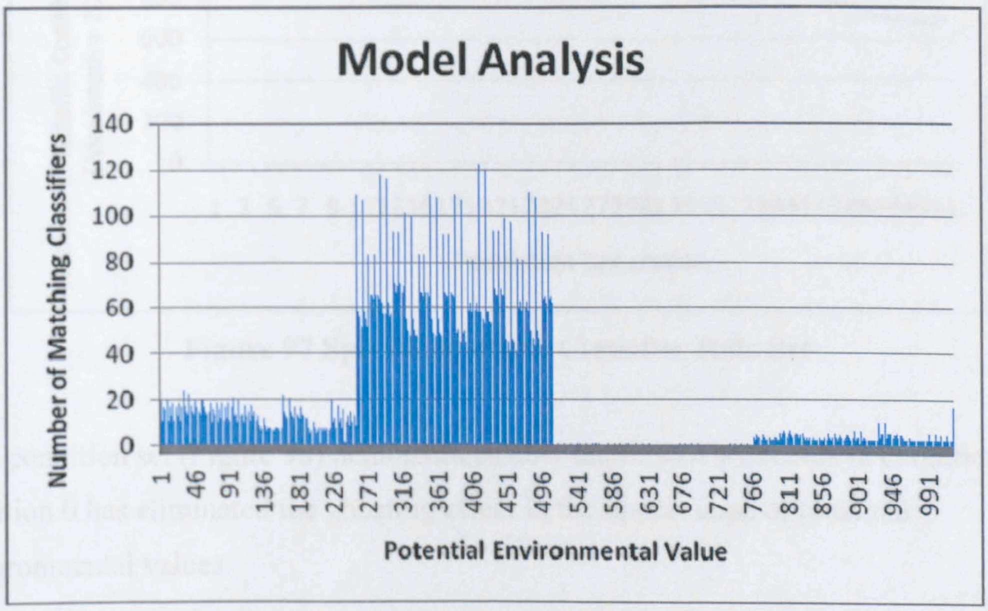


Figure 95 Environment Model 2 - 38 Generations

After 50 generations (Figure 96), there is a tight clustering within the "hot-zone". There is such a significant number of matching classifiers because of wildcards in other condition positions. The lowering of specificity can be observed in Figure 97.

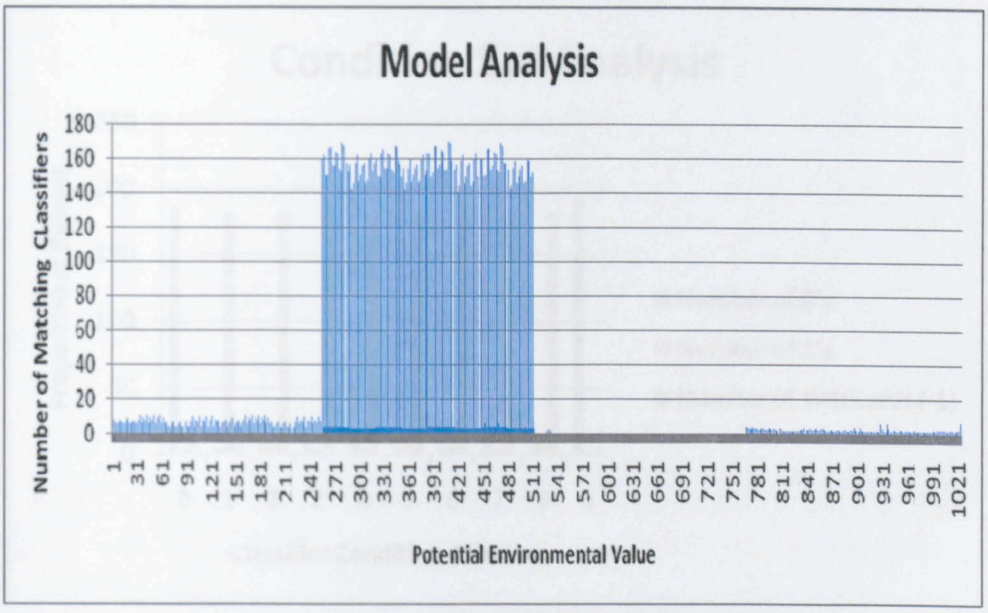


Figure 96 Environmental Model 2 - 50 Generations

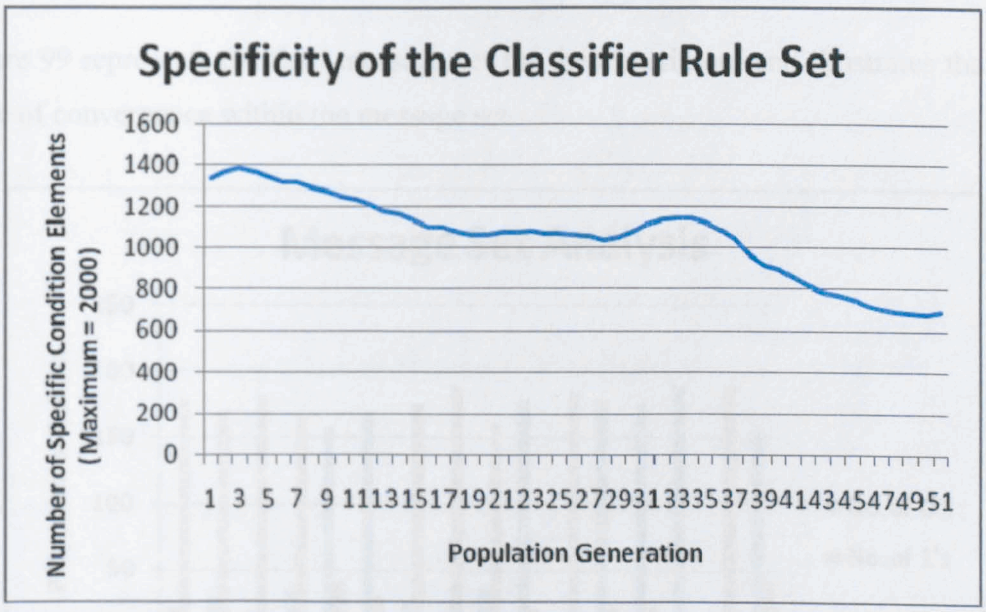


Figure 97 Specificity of the Classifier Rule Set

The condition set (Figure 98) demonstrates how the lack of wildcards in condition position 0 has eliminated the ghosting effect in the upper range of potential environmental values.

Increasingly, there are significant numbers of wildcards in positions 2 – 8 (representing 128 down to 2 in terms of binary value), which accounts for the high proportion of matching classifiers in the "hot-zone".

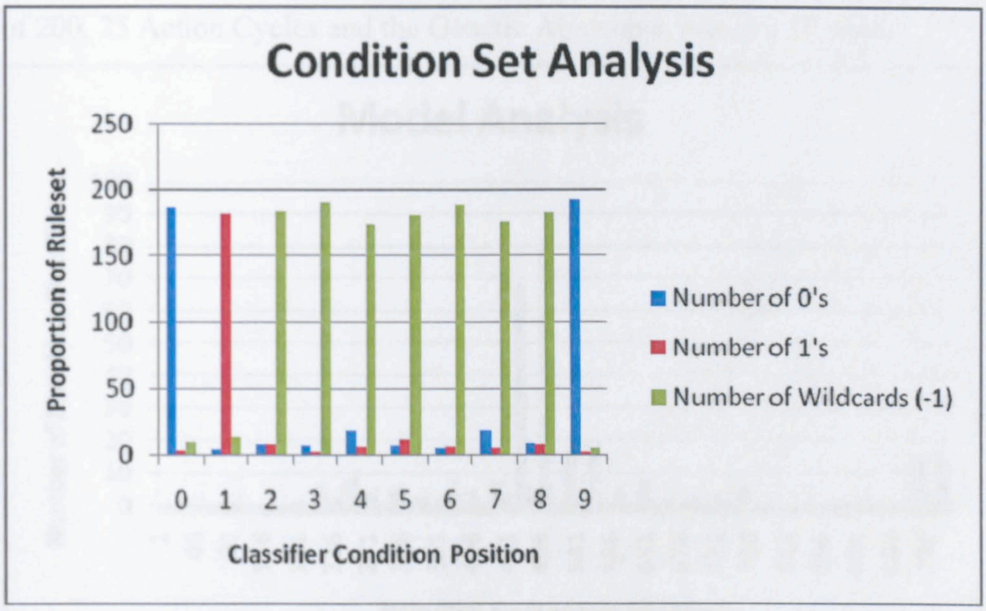


Figure 98 Analysis 2 of Final Condition Set

Figure 99 represents the final message set for the experiment and illustrates that the issue of convergence within the message set.

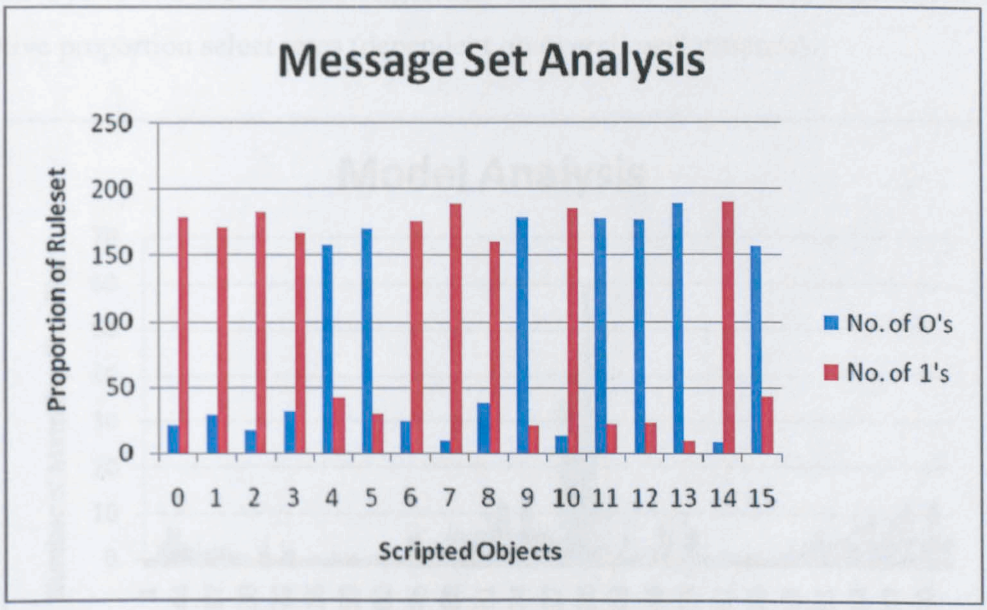


Figure 99 Analysis 2 of Final Message Set

7.3 Other Experimental Models of the Environment

A selection of further experimental results produced by the prototype system are presented below. Environmental Model 3 (Figure 100) was produced using a rule-set size of 200, 25 Action Cycles and the Genetic Algorithm was run 50 times.

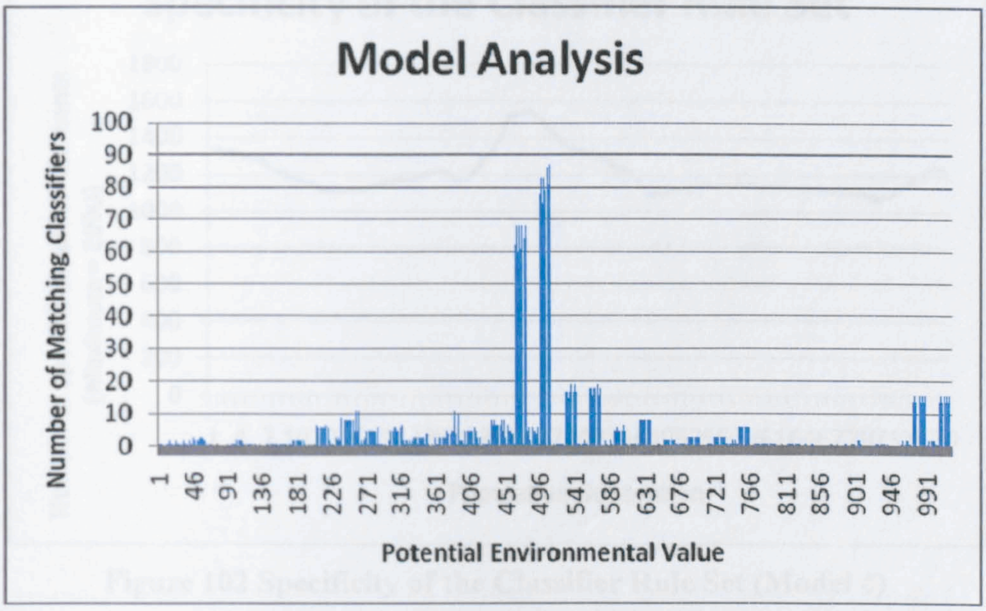


Figure 100 Environmental Model 3

Environmental Model 4 (Figure 101) was produced using a rule-set size of 200, 50 Action Cycles and the Genetic Algorithm was run 80 times. This experiment used adaptive proportion select rates (dependent on overall performance).

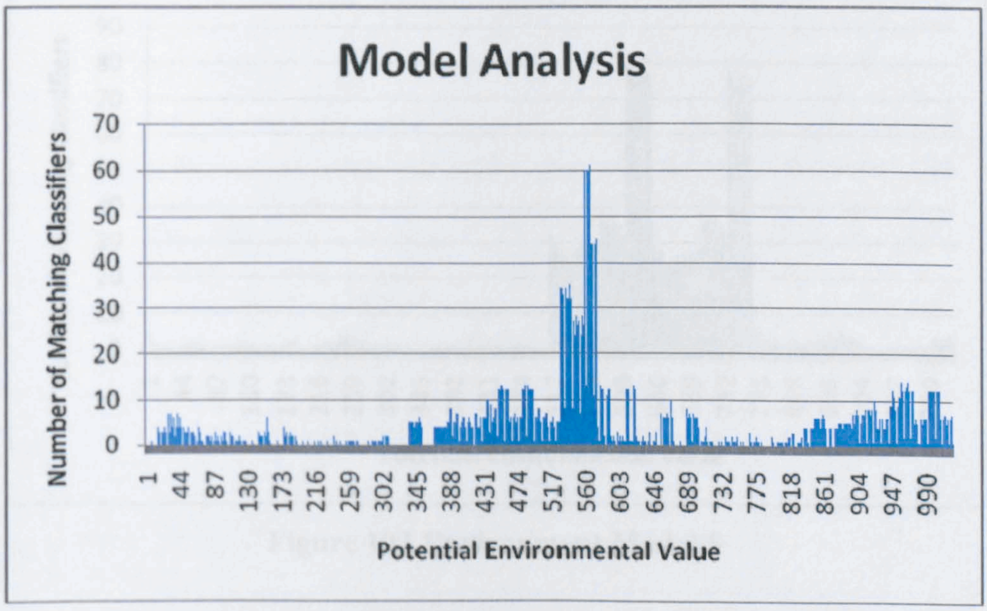


Figure 101 Environment Model 4

The specificity of the classifier rule set of model 4 can be observed in Figure 102 and displays the tendency for specificity to decrease / increase during the experiment depending on the competitive advantage to be gained.

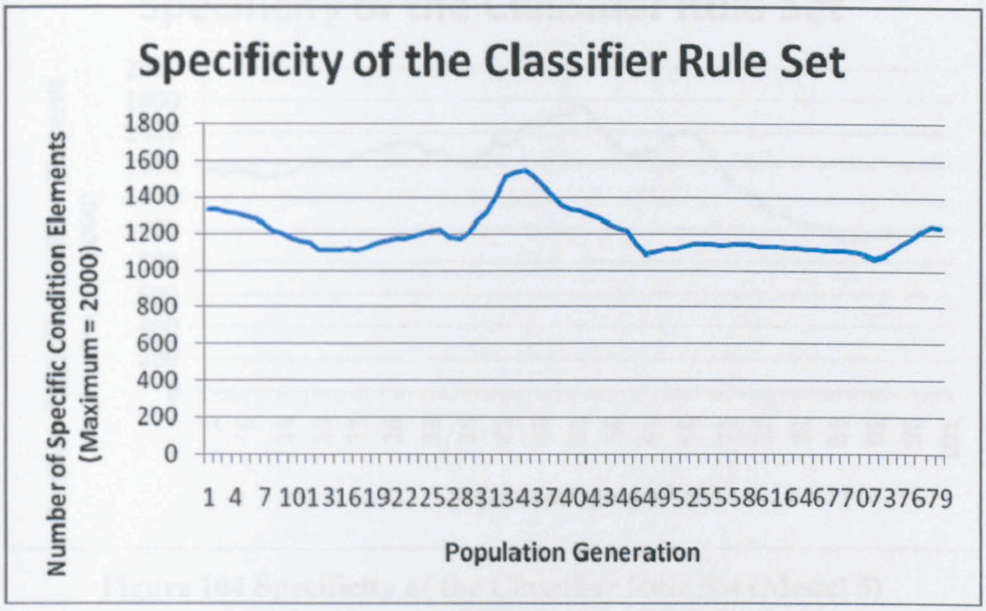


Figure 102 Specificity of the Classifier Rule Set (Model 4)

Environmental Model 5 (Figure 103) was produced using a rule-set size of 200, 10 Action Cycles and the Genetic Algorithm was run 100 times.

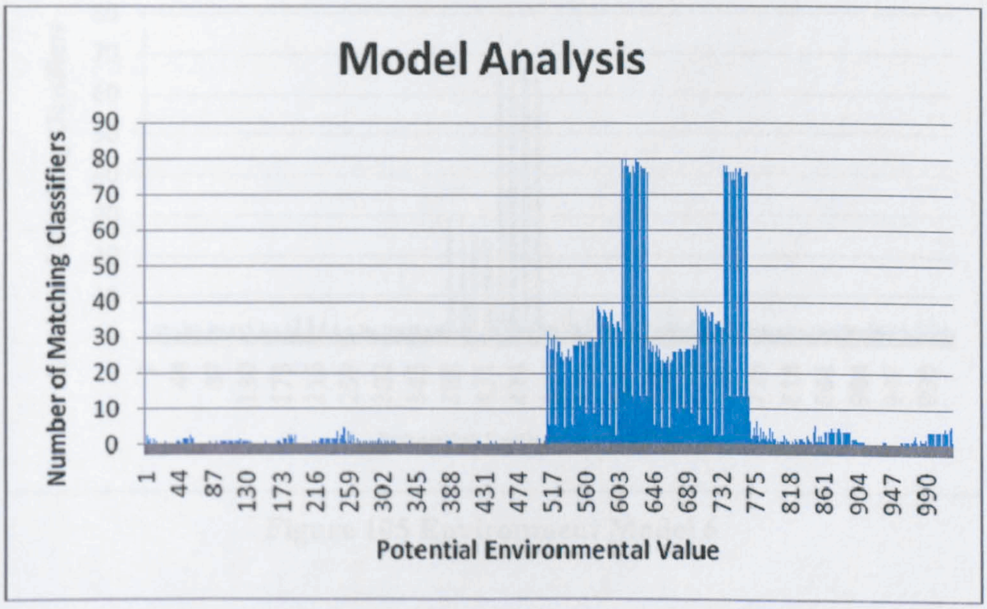


Figure 103 Environment Model 5

The specificity of the classifier rule set of Experiment 5 can be observed in Figure 104.

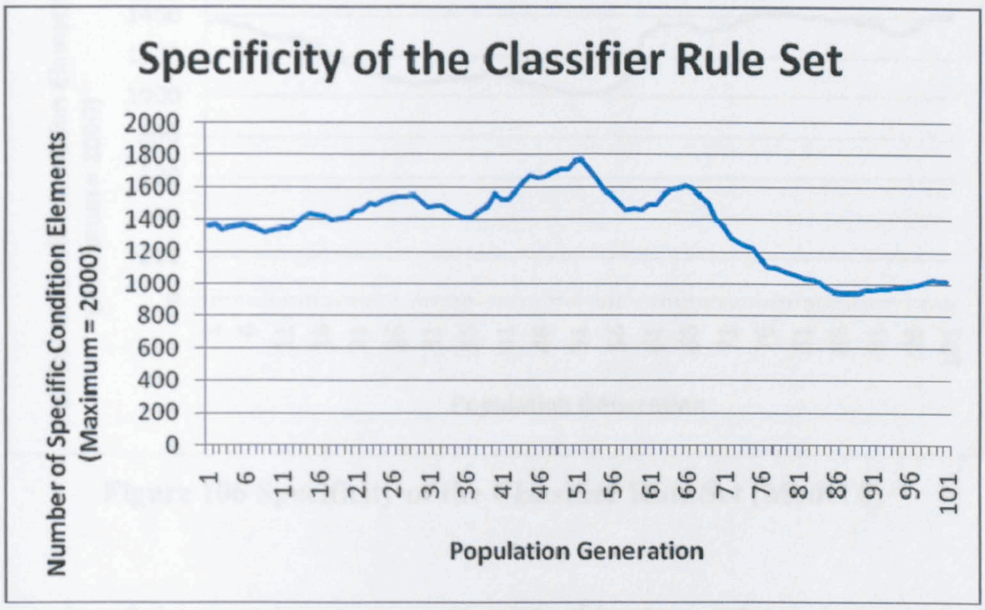


Figure 104 Specificity of the Classifier Rule Set (Model 5)

Environmental Model 6 (Figure 105) was produced using a rule-set size of 200, 25 Action Cycles and the Genetic Algorithm was run 100 times.

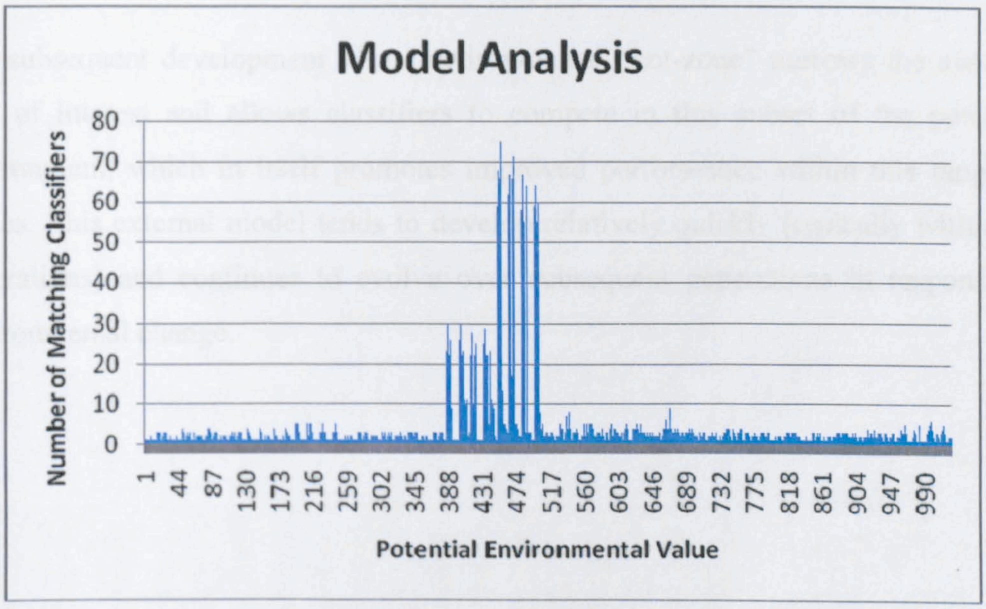


Figure 105 Environment Model 6

The specificity of the classifier rule set of Experiment 6 can be observed in Figure 106.

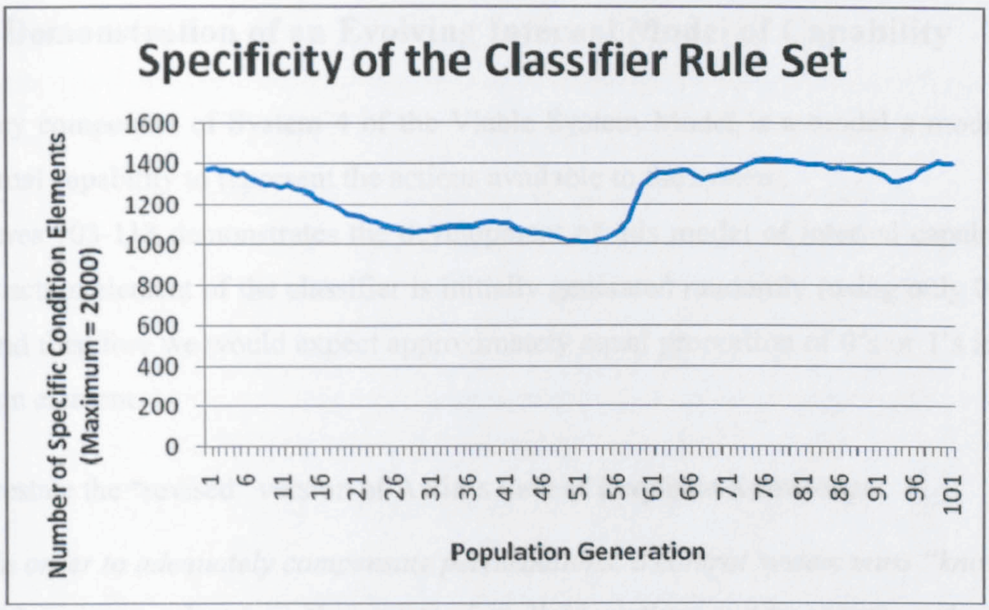


Figure 106 Specificity of the Classifier Rule Set (Model 6)

The results of the experiments as outlined in this chapter have demonstrated the potential of the prototype system in exploring the visible potential environment and using the strength of GAs and LCS to evolve a model of the current external environment.

The subsequent development of an environmental "hot-zone" narrows the systems area of interest and allows classifiers to compete in this subset of the potential environment, which in itself promotes improved performance within this range of values. This external model tends to develop relatively quickly (typically within 10 generations) and continues to evolve over subsequent generations in response to environmental change.

7.4 Demonstration of an Evolving Internal Model of Capability

A key component of System 4 of the Viable System Model is a model a model of internal capability to represent the actions available to the system.
Figures 103-113 demonstrates the development of this model of internal capability. The action element of the classifier is initially generated randomly (using only 0 and 1) and therefore we would expect approximately equal proportion of 0's or 1's in the action element.

To restate the "revised" version of Aulin's Law of Requisite Knowledge:

"In order to adequately compensate perturbations, a control system must "know" which action to select from the variety of available actions and be able to evolve new actions in response to as yet unknown environmental conditions"

Then the prototype system is required to hold a model of its current capability, in terms of effecting change, but also "know" which the appropriate action to select and to evolve this model in response to environmental changes.

The initial randomly generated model of capability (relating to the LCS message action set) is shown in Figure 107 below. As can be seen there is approximately an equal number of 0s and 1s in each position.

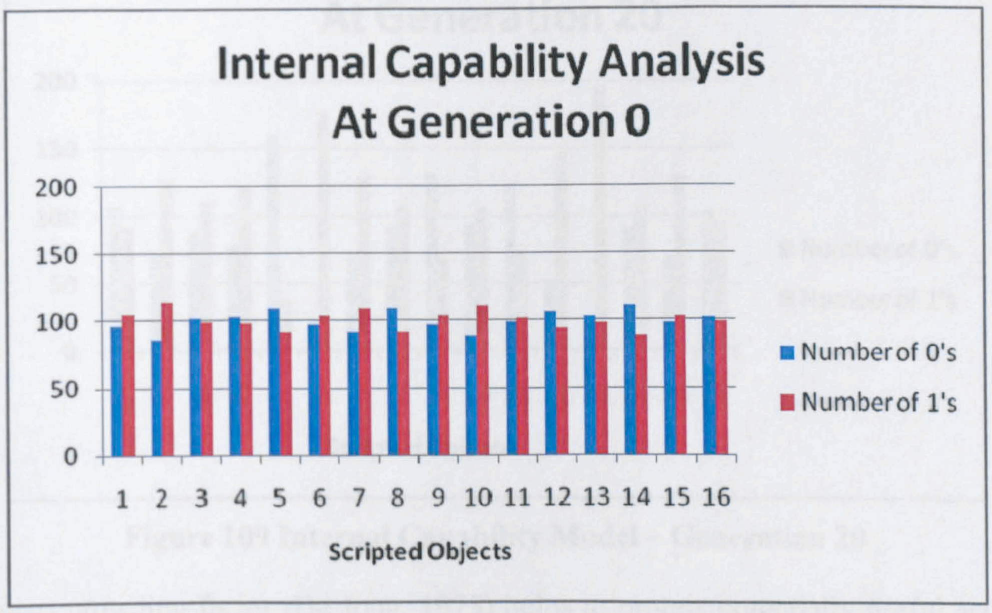


Figure 107 Internal Capability Model – Generation 0

As can be seen in Figure 108 and Figure 109, the prototype system is evolving actions in response to incoming environmental messages. The strength (or fitness) of the classifiers are rewarded for "good" action sets and therefore these classifiers are favoured using the genetic algorithm process at the end of the action cycle.

The computational load on the environment of the scripted objects ranges from Scripted Object 1 (highest) to Scripted Object 16 (lowest).

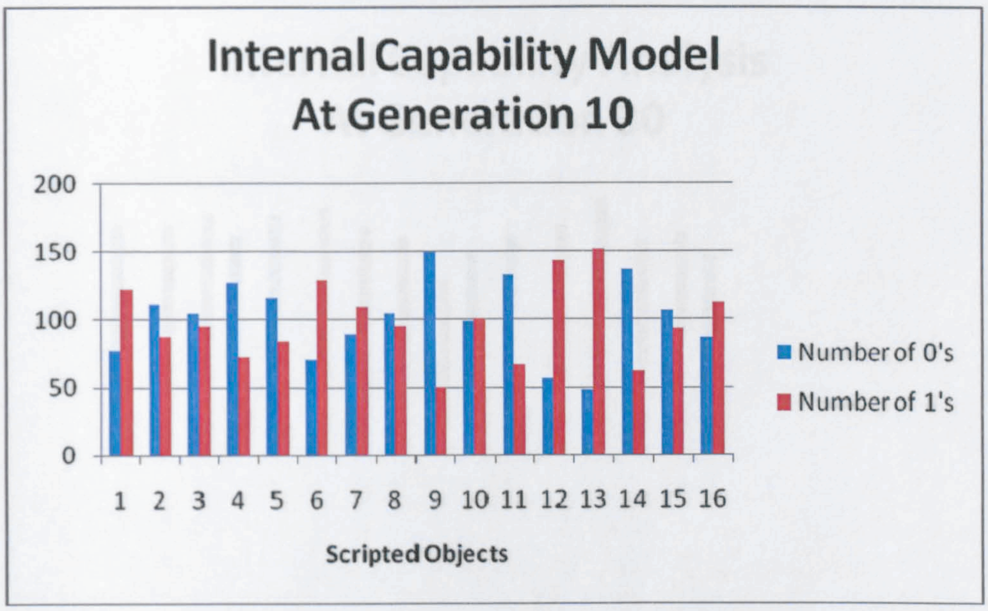


Figure 108 Internal Capability Model – Generation 10

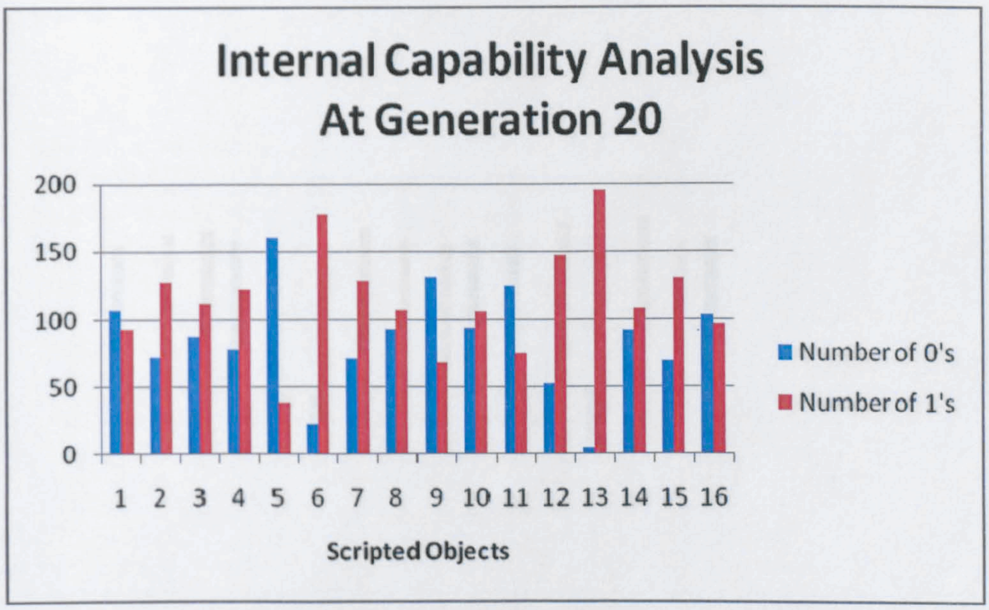


Figure 109 Internal Capability Model – Generation 20

DeJongs crowding factor (De Jong, 1975) helps to protect potentially useful actions being bred out of the LCS population. We can observe that in generation 30 (Figure

110) a number of actions at particular positions would potentially be susceptible to being lost to the prototype. The most "at-risk" at this particular point in the experiment is the ability to switch off scripted object 13. The benefit of maintaining this "ability" can be observed in Generation 70 (Figure 114) where the switching off of scripted object 13 is seen as beneficial to a number of the classifier rule-sets. If this ability had been lost at this point in the model development mutation would be relied upon to re-introduce this element to the action set.

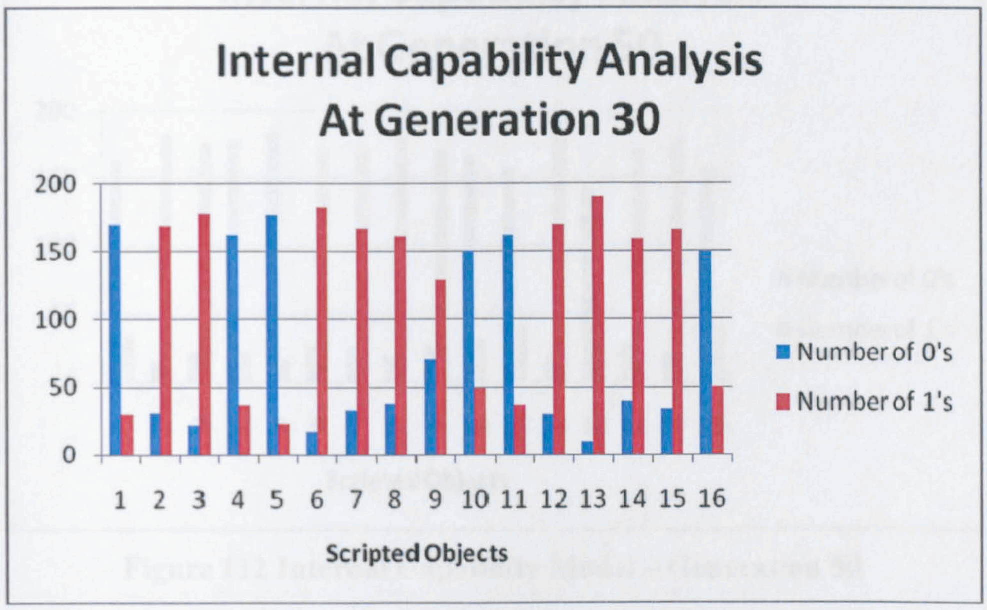


Figure 110 Internal Capability Model – Generation 30

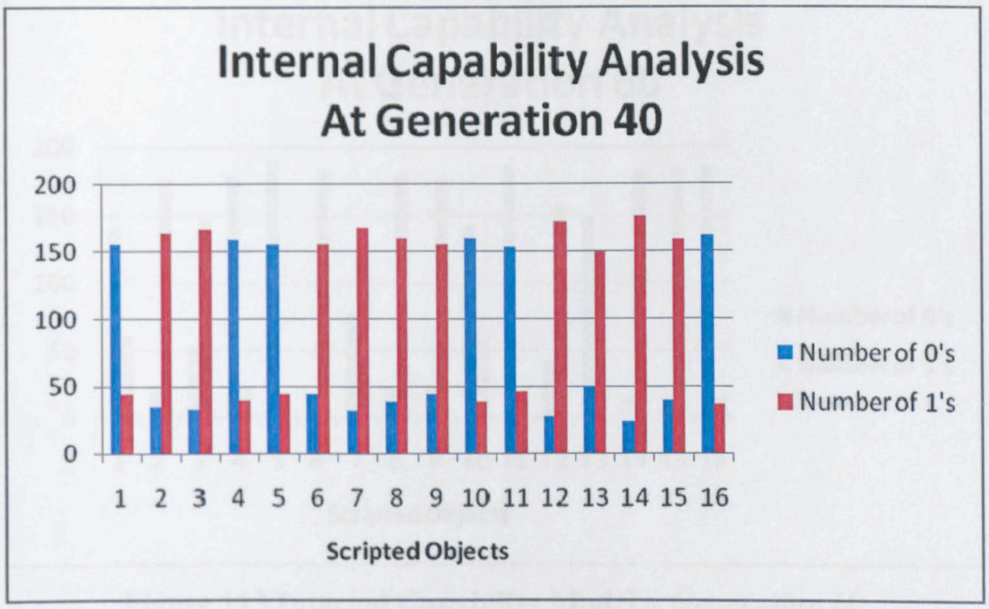


Figure 111 Internal Capability Model – Generation 40

Generations 30 – 60 (Figure 110 - Figure 113) have sought to allow the system the ability to maintain its performance levels developing requisite knowledge to switch off relevant scripted objects (particularly in the most significant positions in terms of computation load). This would indicate that performance is being affected by other factors outside the visible environment of the prototype system and the system is evolving its rule-set and consequently capability to adequately compensate for these perturbations to the system.

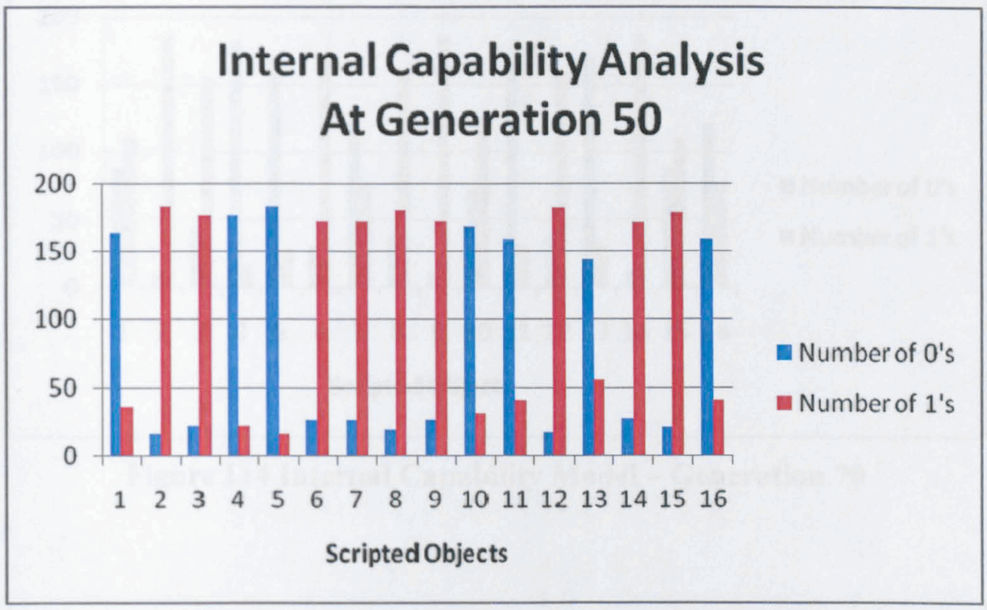


Figure 112 Internal Capability Model – Generation 50

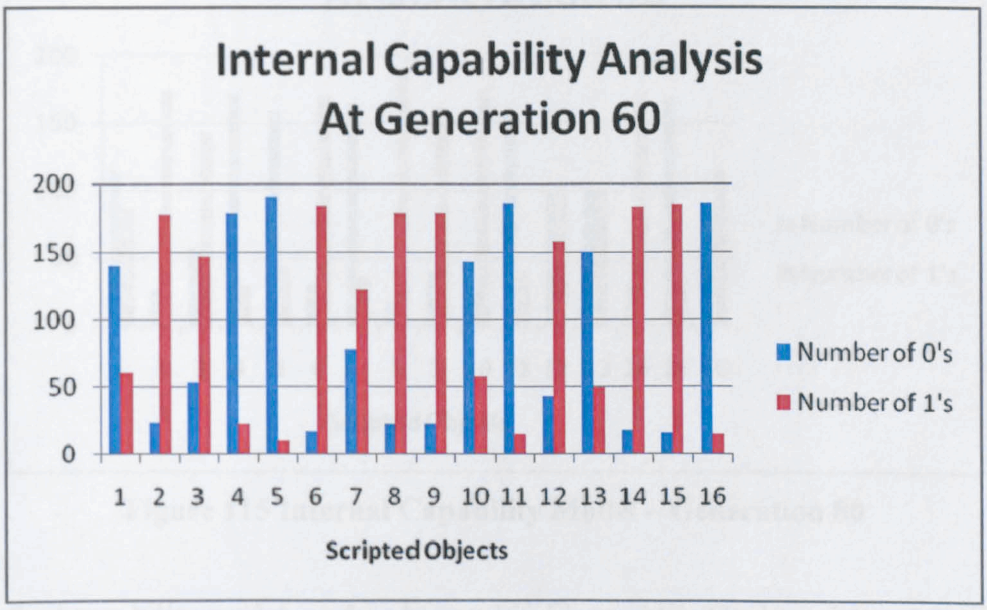


Figure 113 Internal Capability Model – Generation 60

As the perturbations to the environment have reduced we can see that the system in generations 70 – 80 (Figure 114 and Figure 115) begins to evolve its capability model to allow the system to respond with appropriate actions, with respect to the current environment.

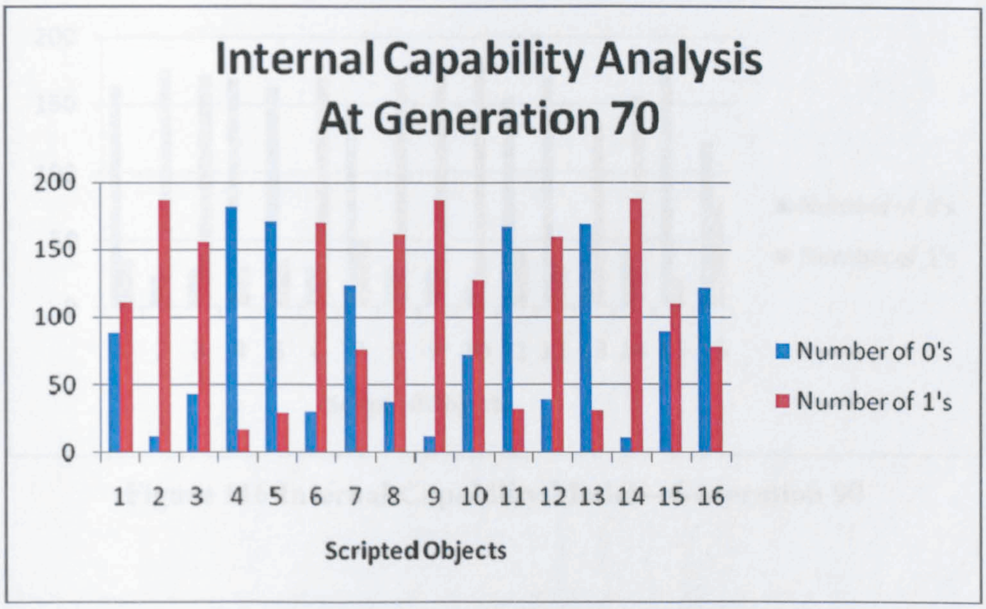


Figure 114 Internal Capability Model – Generation 70

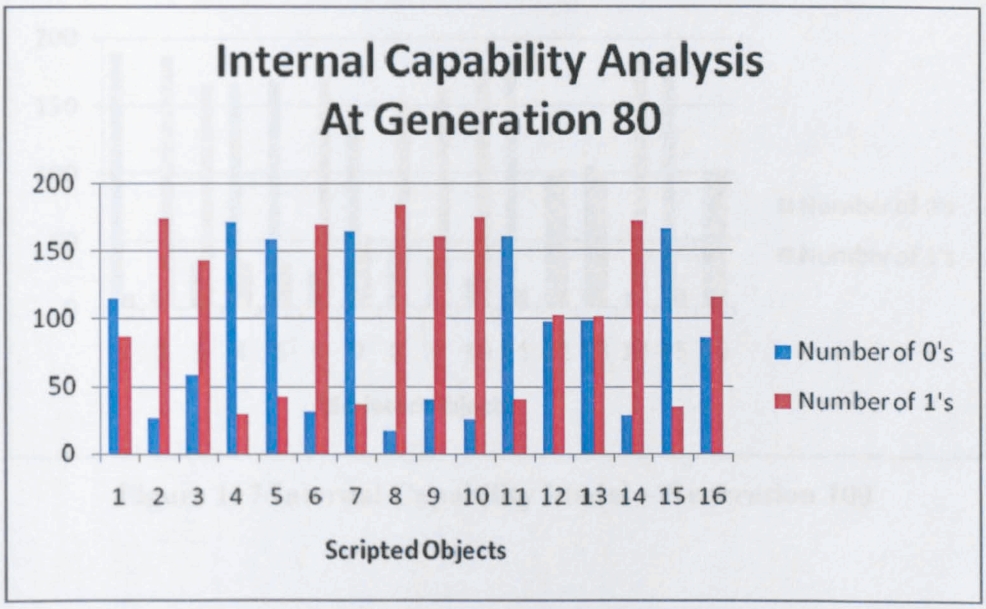


Figure 115 Internal Capability Model – Generation 80

The final capability model can be observed in Figure 117. The key element from this experiment is to observe the evolving nature of the actions available to the system and the ability of the system to "backtrack" in order to respond to changing environmental condition, which necessitates such action to be undertaken.

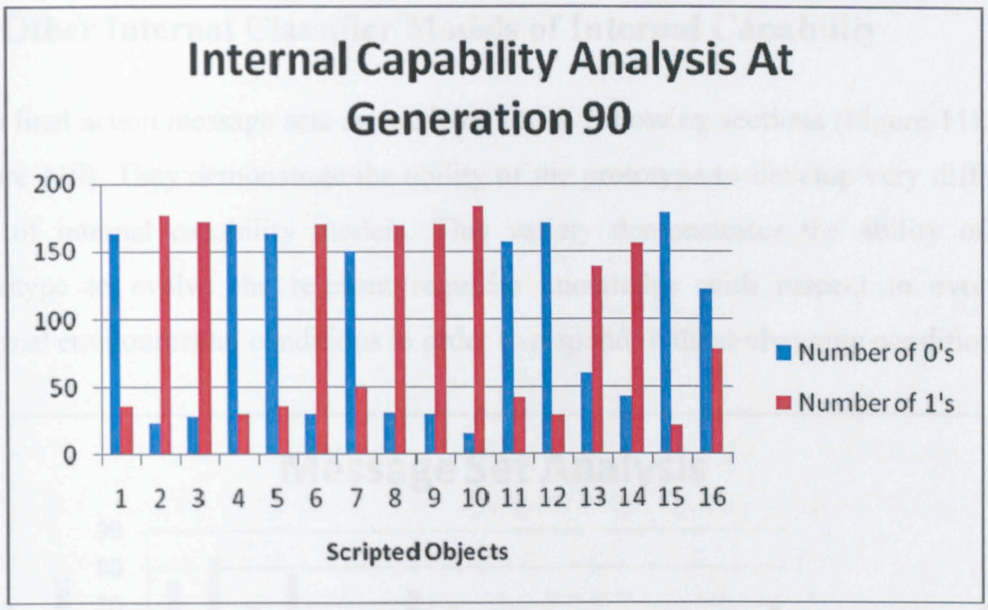


Figure 116 Internal Capability Model – Generation 90

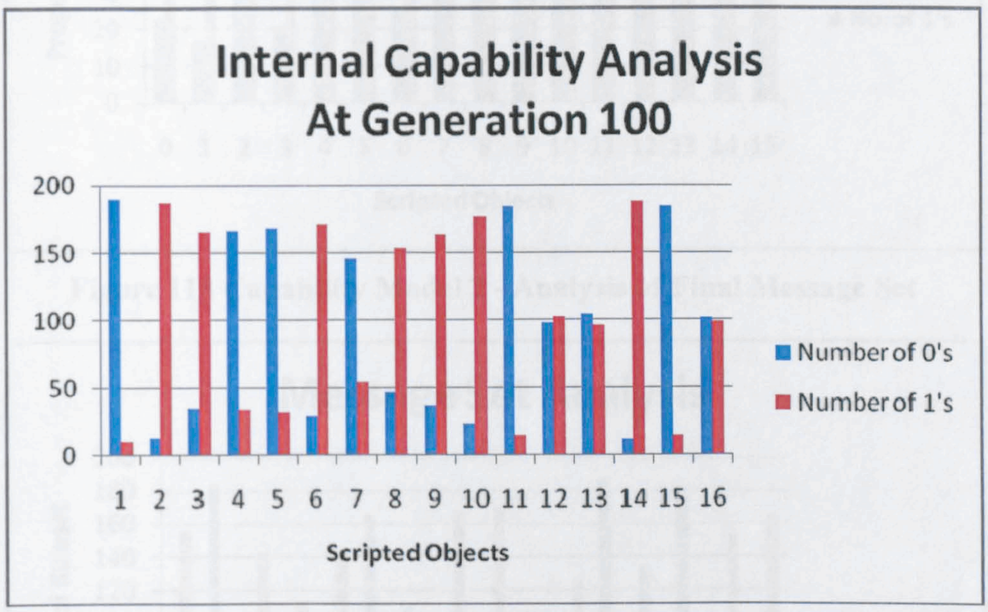


Figure 117 Internal Capability Model – Generation 100

7.5 Other Internal Classifier Models of Internal Capability

Two final action message sets are outlined in the following sections (Figure 118 and Figure 119). They demonstrate the ability of the prototype to develop very different sets of internal capability models. This variety demonstrates the ability of the prototype to evolve the relevant requisite knowledge with respect to evolving external environmental conditions in order to respond to these changing conditions.

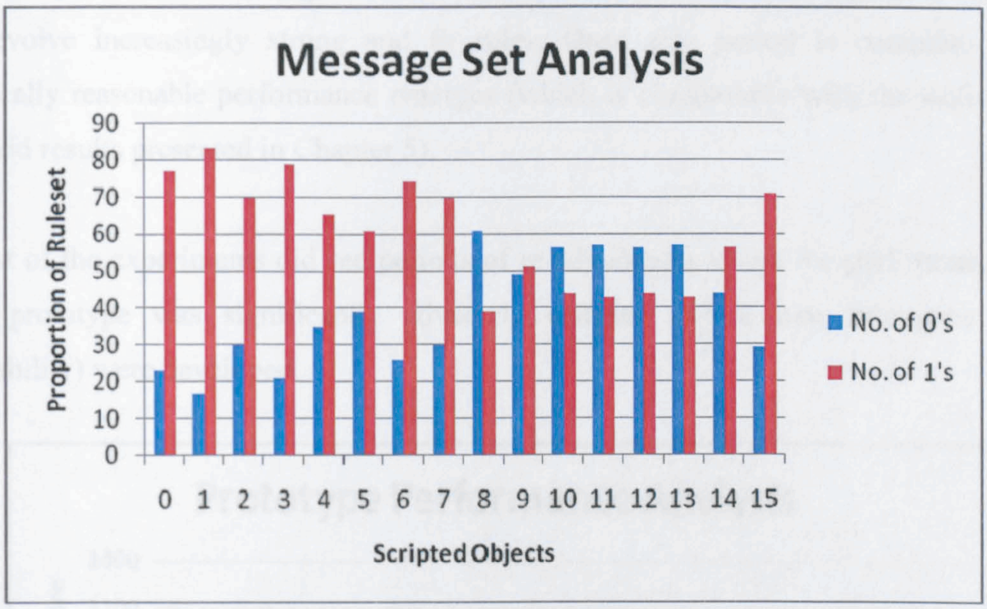


Figure 118 Capability Model 2 - Analysis of Final Message Set

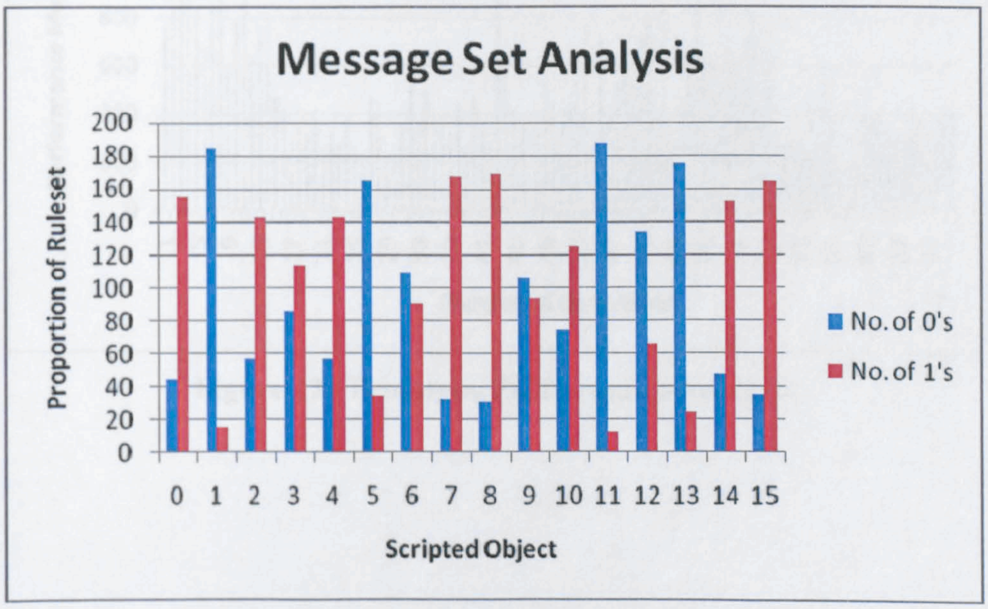


Figure 119 Capability Model 3 - Analysis of Final Message Set

7.6 Analysis of the Performance of the Prototype

The results of experiments conducted to determine the ability of the prototype to control performance on the simulator region are presented below.
Significantly more experiments were conducted but they displayed similar characteristics to those presented in the thesis.

Typically 1 – 10 "learning generations" execute before the genetic algorithm begins to evolve increasingly strong and fit rules. Once this period is complete then typically reasonable performance emerges (which is comparable with the static and hybrid results presented in Chapter 5).

Most of the experiments did see periods of re-adjustment where the performance of the prototype was significantly adversely affected before new strategies (i.e. capability) were developed.

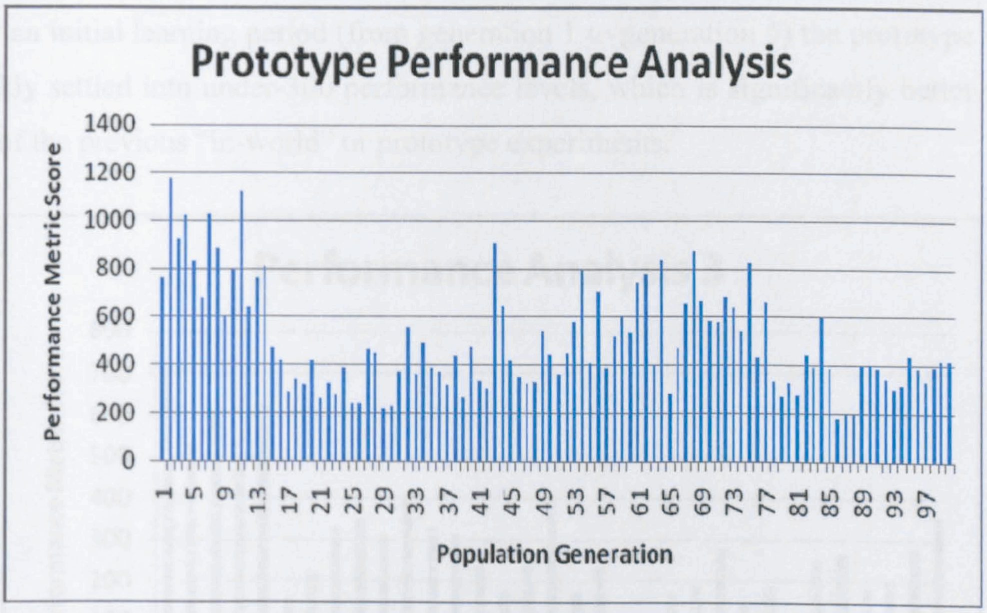


Figure 120 Prototype Performance Analysis

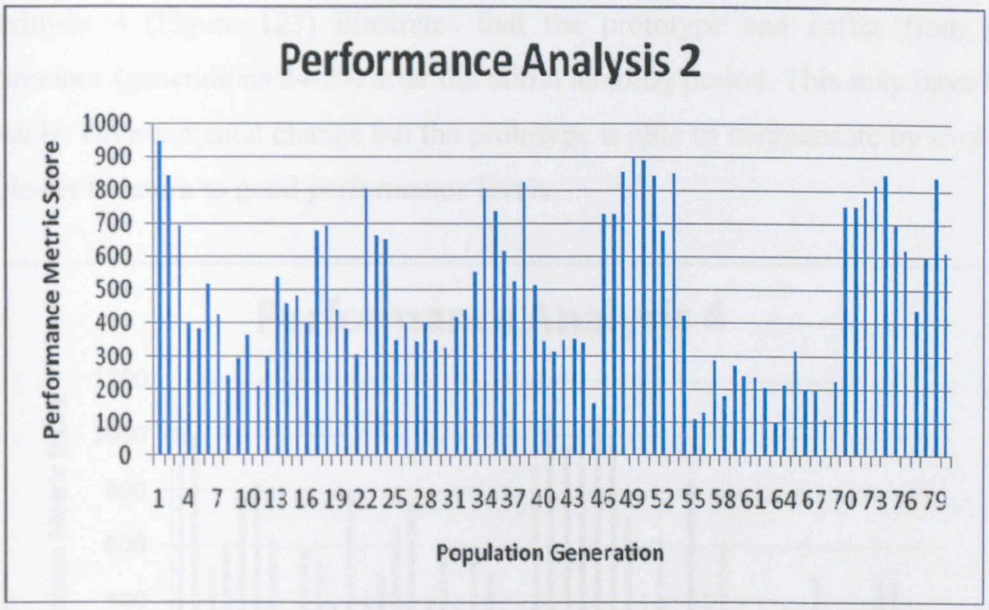


Figure 121 Prototype Performance Analysis 2

As illustrated in Figure 122, the performance analysis of this experiment (i.e. hitting the goal metric) was significantly better than those observed previously. Notably, after an initial learning period (from generation 1 to generation 5) the prototype very quickly settled into under-300 performance levels, which is significantly better than any of the previous "in-world" or prototype experiments.

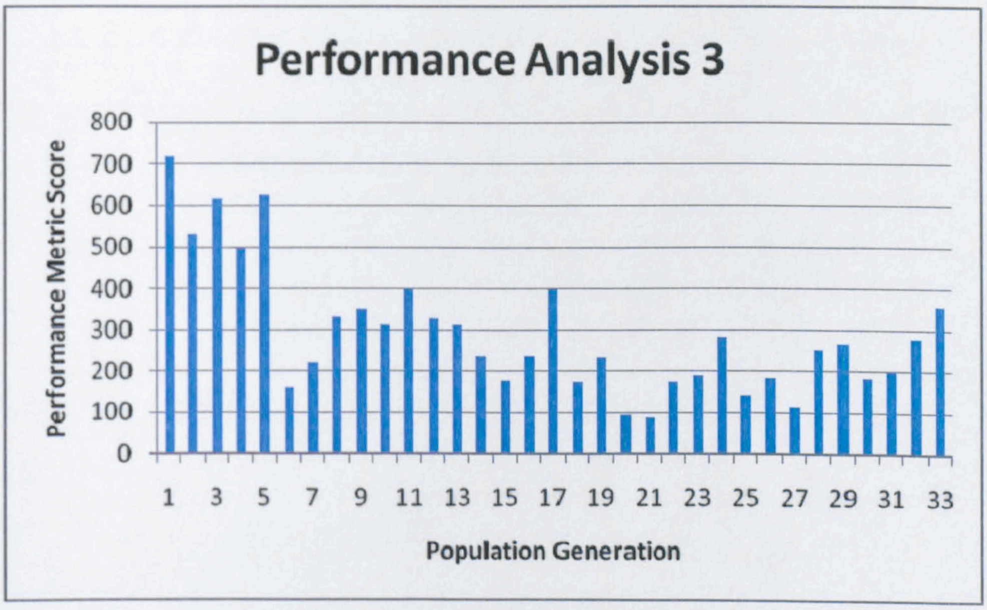


Figure 122 Prototype Performance Analysis 3

Experiment 4 (Figure 123) illustrates that the prototype can suffer from poor performance (generations 24-29) after the initial learning period. This may have been caused by environmental change but the prototype is able to compensate by evolving its rule-set to return to good performance levels.

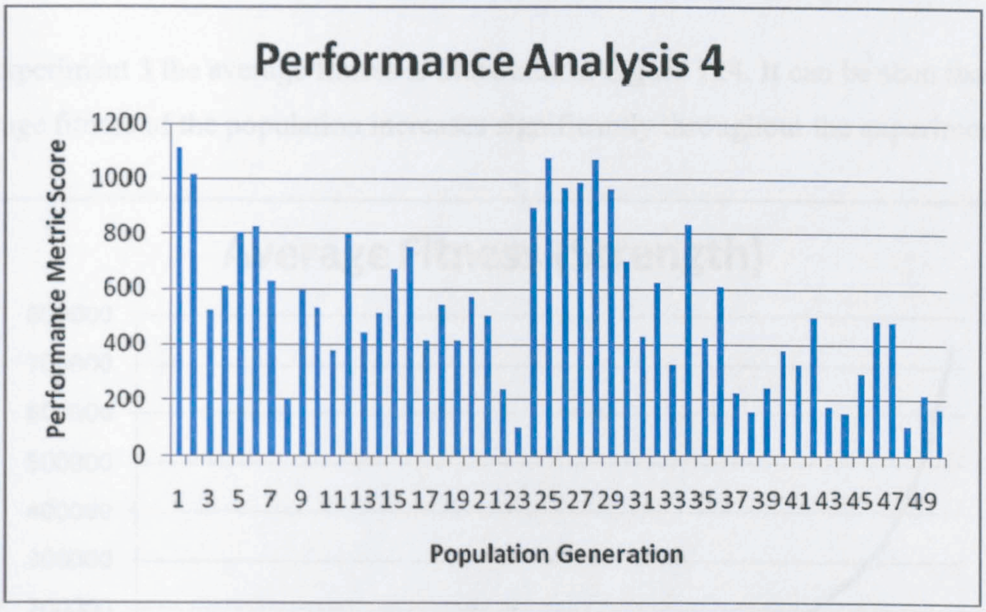


Figure 123 Prototype Performance Analysis 4

7.7Analysis of the Performance of the Genetic Algorithm Performance and Convergence

Given the mixed performance levels of the prototype system, it may be useful to analyse the average population fitness over a specified number of generations.

In Experiment 3 the average fitness is illustrated in Figure 124. It can be seen that the average fitness of the population increases significantly throughout the experiment.

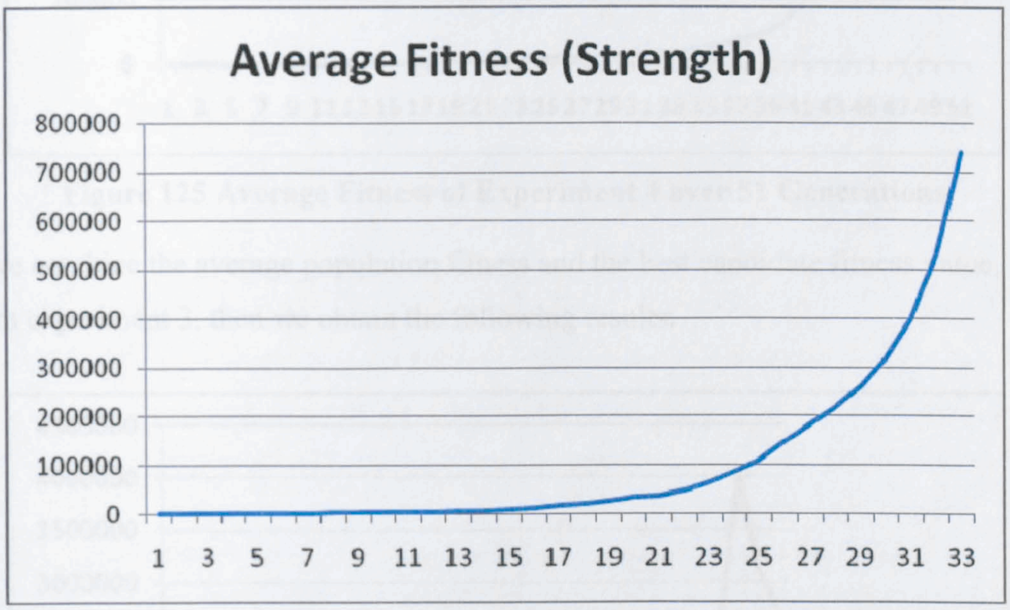


Figure 124 Average Fitness of Experiment 3 Over 33 Generations

In Experiment 4 the average fitness is illustrated in Figure 125. We again observe a similar growth in population fitness as in the previous example.

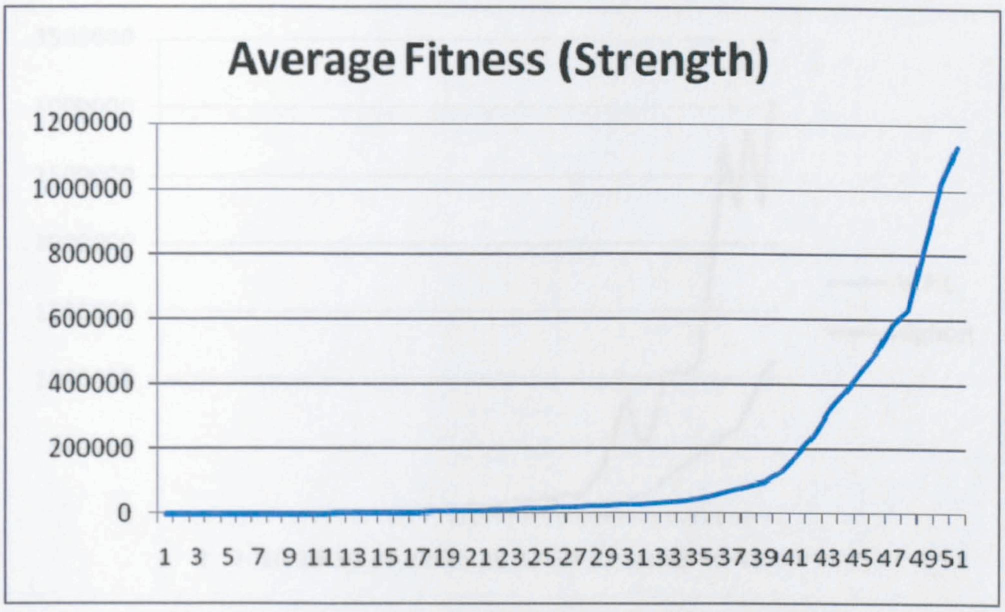


Figure 125 Average Fitness of Experiment 4 over 51 Generations

If we combine the average population fitness and the best candidate fitness value, from experiment 3, then we obtain the following results.

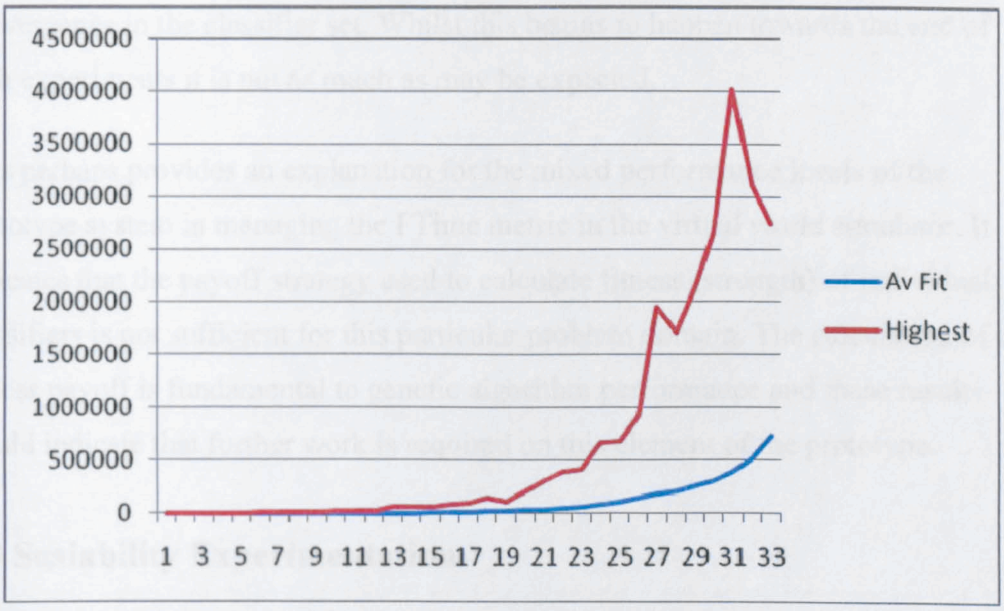


Figure 126 Average Fitness/Best Candidate Fitness of Experiment 3

If we combine the average population fitness and the best candidate fitness value, from experiment 4, then we obtain the following results.

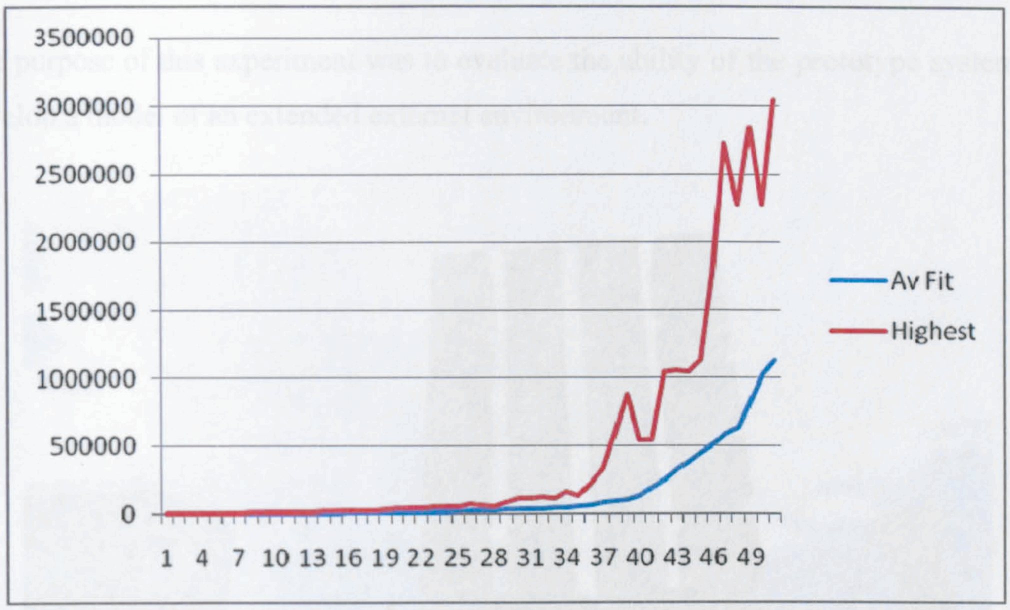


Figure 127 Average Fitness / Best Candidate Fitness of Experiment 4

In terms of the genetic algorithm performance and convergence we would expect the difference between the average fitness and best candidate to converge. This indicates convergence in the classifier set. Whilst this begins to happen towards the end of both experiments it is not as much as may be expected.

This perhaps provides an explanation for the mixed performance levels of the prototype system in managing the FTime metric in the virtual world simulator. It indicates that the payoff strategy used to calculate fitness (strength) of individual classifiers is not sufficient for this particular problem domain. The calculation of the fitness payoff is fundamental to genetic algorithm performance and these results would indicate that further work is required on this element of the prototype.

7.8 Scalability Experimentation

7.8.1 Scalability Experiment 1

To extend the scale of the experiment the parameters were expanded so that

a) The condition element used 12 bits (as opposed to 10 bits). This extended the potential environmental values from 0 – 1024 to 0 – 4096.

b) The number of scripted objects was increased from 16 to 32. This increased the number of combinations of the scripted objects from 65356 to 4,294,967,296.

The purpose of this experiment was to evaluate the ability of the prototype system to develop a model of an extended external environment.

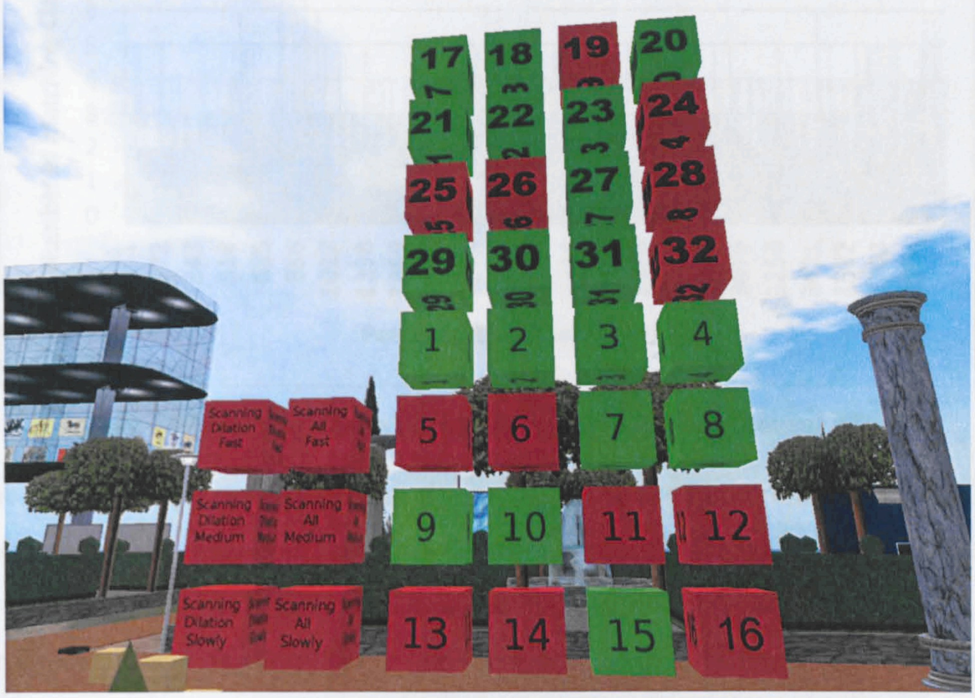


Figure 128 Visualisation of the Scalability Experiment

The initial parameters used were a rule-set of 200, 100 population generations and 25 action cycles.

The initial environmental model can be seen in Figure 129. The randomly generated initial model provides good general coverage of the potential environment of the system.

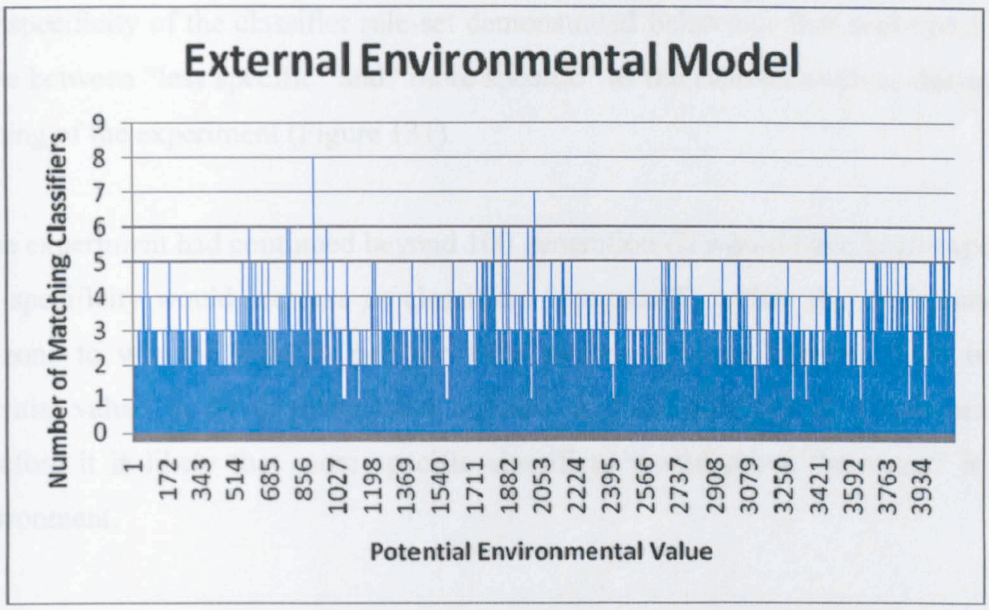


Figure 129 Scalability Experiment Initial Environmental Model

At the completion of the experiment (Figure 130) the final environmental model can be observed. We can see that the environmental "hot-zone" is approximately between 530 – 780 and that the classifier set has evolved to closely model that environment. There are very few classifiers outside of this range and any that do exist are due to the effect of wildcards in the classifier rule-set.

The prototype system has demonstrated its ability in an extended experiment to produce an effective external environmental model of the environment.

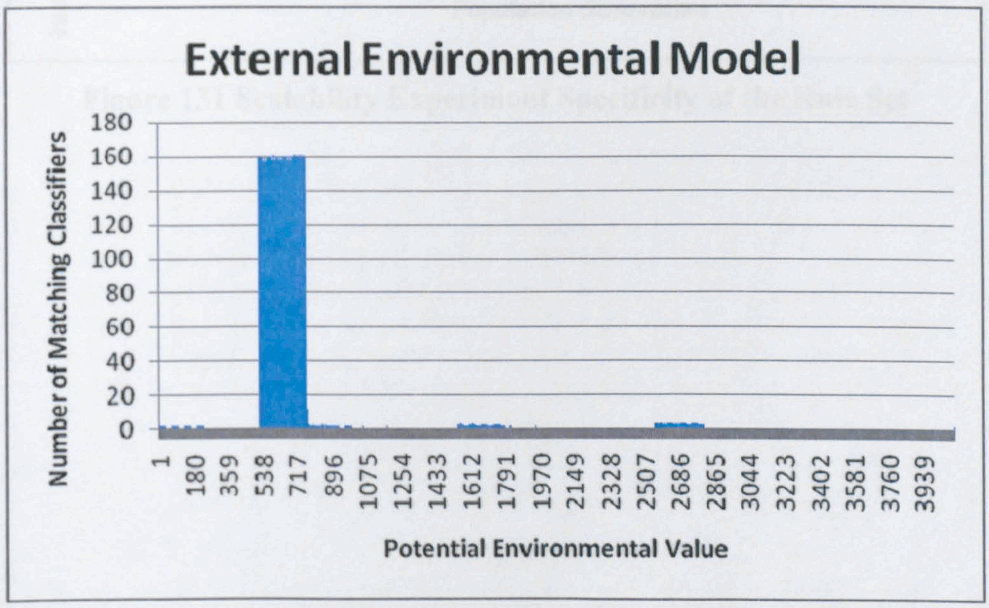


Figure 130 Scalability Experiment – Final Environmental Model

The specificity of the classifier rule-set demonstrated behaviour that sees specificity move between "less specific" and "more specific" as the rule-set evolves during the running of the experiment (Figure 131).

If the experiment had continued beyond 100 generations it would have been expected that specificity would increase as classifiers "competed" within the environmental hot-zone to win the right to perform their actions. This is because most of the potential values in the environmental hot-zone had multiple classifier matches and therefore it is likely that more specific classifiers would prove "stronger" in this environment.

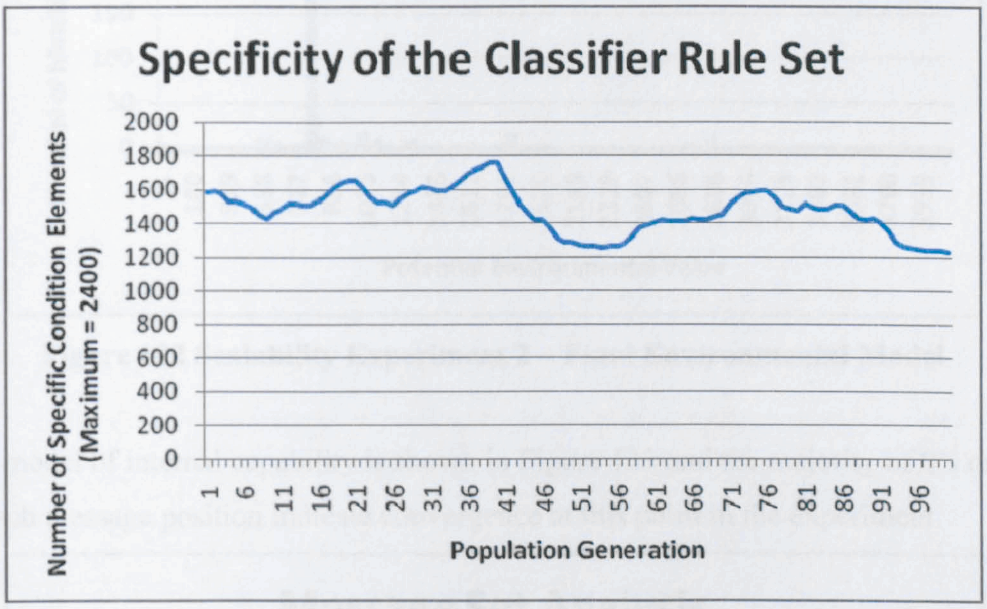


Figure 131 Scalability Experiment Specificity of the Rule Set

7.8.2 Scalability Experiment 2

The experiment was repeated using a rule-set of 500, 80 population generations and 50 action cycles. Again, this demonstrates the ability of the prototype system to evolve a model of the external environment (Figure 132), reflecting the environmental values being observed by the system.

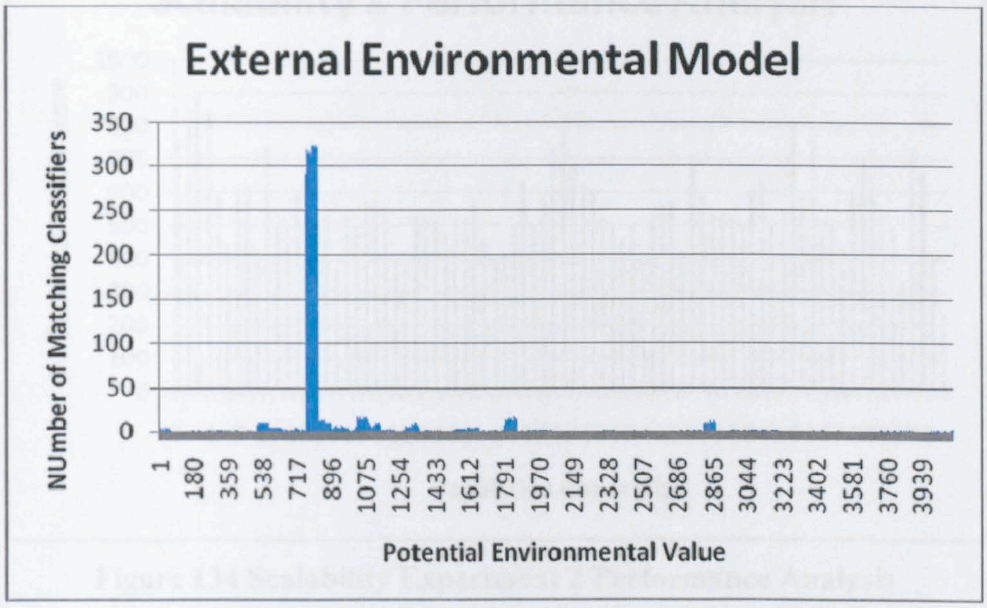


Figure 132 Scalability Experiment 2 – Final Environmental Model

The model of internal capability is shown in Figure 133 and the majority of 0's or 1's in each message position indicate convergence at this point in the experiment.

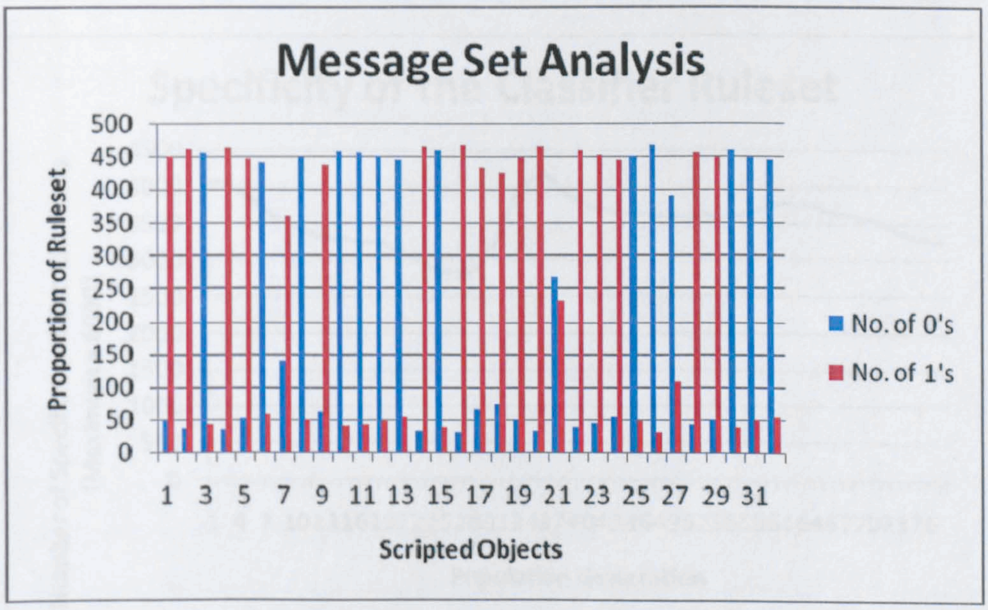


Figure 133 Internal Capability Model – Generation 80

The performance of the Scalability experiment 2 is relatively disappointing (Figure 134). With 50 action cycles, metric scores of between 200-300 were expected whereas the prototype was typically between 400 – 500. Perhaps this is a reflection of the additional complexity of the problem, which would require additional learning to take place before performance would improve.

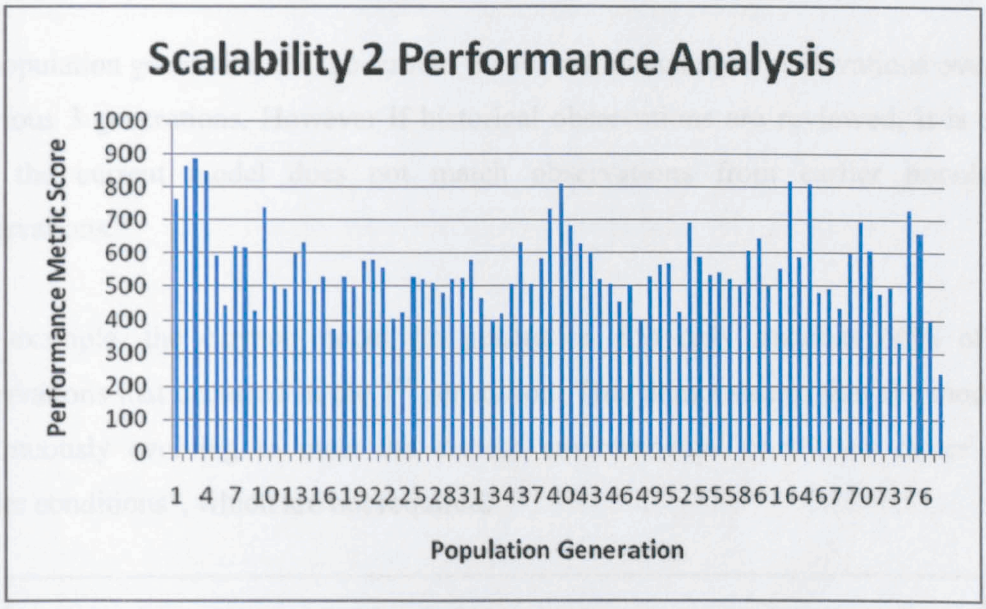


Figure 134 Scalability Experiment 2 Performance Analysis

Again the experiment (Figure 135) demonstrated the same specificity behaviour as observed in the earlier experiment (Figure 131). We observe specificity decreasing and increasing as classifiers as expectations of the benefits of being more specific or general evolve over time.

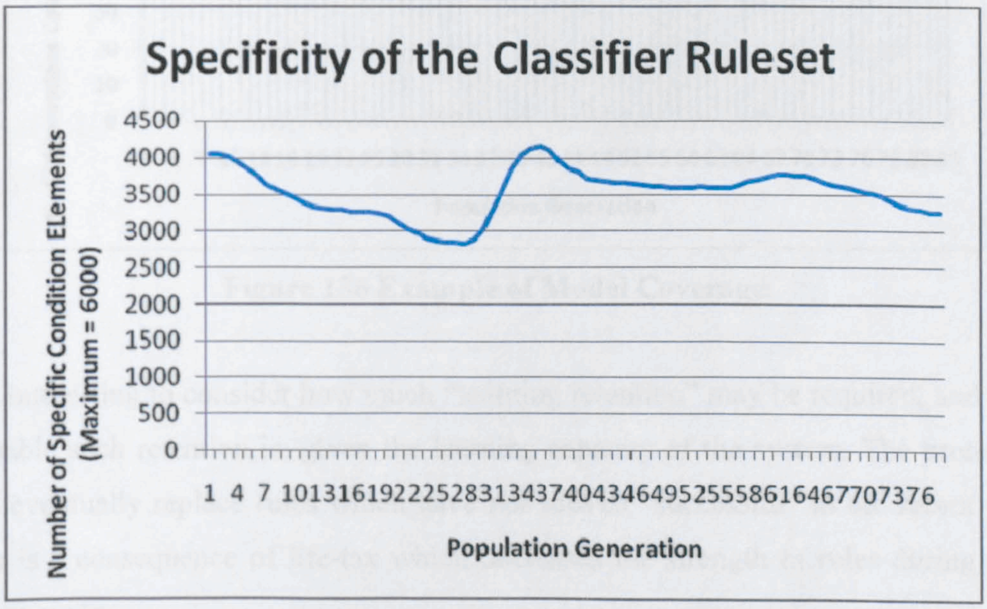


Figure 135 Scalability Experiment 2 Specificity of the Rule Set

7.9 Model Coverage and Currency

An example of the model coverage and currency is provided below (Figure 136). This demonstrates the "model currency" by comparing all previous environmentally observed values.

At population generation 85, the current model still matches all observations over the previous 3 generations. However if historical observations are reviewed, it is clear that the current model does not match observations from earlier population observations.

For example, the current model (in generation 85) only matches 60% of the observations that occurred in the 1st generation. This demonstrates that the model is continuously evolving to meet the current environmental conditions rather than "older conditions", which are not required.

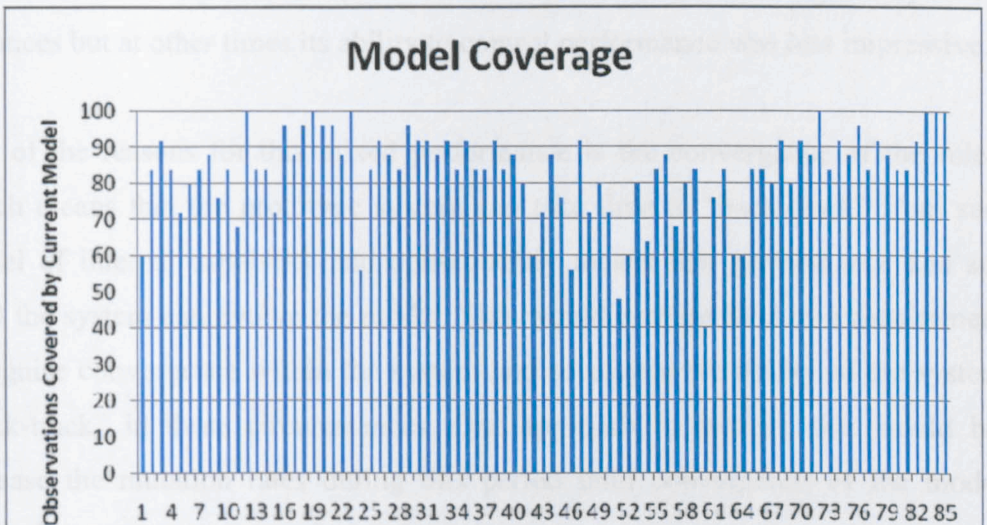


Figure 136 Example of Model Coverage

It is interesting to consider how much "learning retention" may be required, and how valuable such retention is, given the learning capacity of the system. The prototype will eventually replace rules which have not proved "successful" in the recent past. This is a consequence of life-tax which decreases the strength of rules during each action cycle.

7.10 Summary

An overview of key experimental results has been evaluated within this chapter of the thesis. The results have demonstrated the ability of the prototype system to successfully evolve models of the external environment and internal models of the capability of the system in response to environmental conditions.

The prototype has demonstrated the ability to replicate this ability in more complex "scaled up" experiments with relatively detailed environmental models being observed and evolved during the experiment. The model of internal capability has demonstrated the ability to evolve in response to environmental conditions but does suffer from the problem of convergence.

The ability of the prototype system to manage performance within the simulator region was less convincing. It was certainly capable of better performance than the initial baselines experiments (such as static models, hybrid models etc) in some instances but at other times its ability to control performance was less impressive.

One of the reasons for this mixed performance is the convergence of the rule-set, which means that the prototype system can take time to "back-track" from such a model of internal capability and consequently means that performance can suffer until the system can evolve the model. This would perhaps lead to a requirement to recognise convergence within the system and accelerate the ability of the system to "back-track" in these circumstances. One approach to achieve this would be to increase the mutation rates during this period until convergence of the model is reduced.

Chapter 8: Adaptive Environmental Modelling Prototype System Evaluation

8.1 Overall Evaluation of the Project Work

As one of the main aims of the thesis was to study the generic requirements of a machine learning mechanism / service to facilitate runtime monitoring and adaptive modelling of a system environment, the thesis has produced some interesting experimental data and much scope for future work.

The thesis test the cybernetic theory provided by Beer's VSM, which advocated two models for "viability" – a model of the external environment that must be adaptive to cater for environmental change and a similarly adaptive model of internal capability (i.e. the responses the system was capable of). Provision of these models also complied with Ashby's Law of Requisite Variety and Aulin's Law of Requisite Knowledge.

The work has demonstrated that any environmental model used by autonomic self-adaptive systems in the decision making process should also be able to evolve during runtime in order to maintain model currency. The provision of adaptive environmental models has and continues to provide a considerable challenge to the software development community. The thesis has outlined some generic requirements for goals, metrics, detectors, effectors and communication channels etc. necessary both to detect and enact change.

It has also demonstrated the requirement for a process to allow any environmental model to be able to adequately "back-track" in a timely manner. By "back-track" we mean that the system has to be able to respond to environmental change even after the internal and external models used by the system may have "hardened" due to the issue of convergence. This will become increasingly important in time-critical environmental applications where it is not enough to theorise that the system will evolve an appropriate response "eventually".

The project work used Learning Classifier Systems (LCS) as the approach to model a changing environment and in terms of producing appropriate and current models this approach has been successful in this particular application domain. LCS was chosen due to its theoretical "closeness" to cybernetics but there are drawbacks to the particular approach including the difficulty in adequately designing a system solely on binary strings. The problem of convergence, noted earlier in the thesis, is a significant issue within LCS and this "hardening" of the model, especially in the message action set does still remains a significant issue when designing systems operating in changeable environments. The future research section of this thesis does suggest some potential solutions to alleviate the issues caused by convergence.

In terms of the general approach, it would seem to suggest that evolutionary programming techniques, whether "classical LCS" or a different approach such as Cultural Algorithms (Reynolds, 1994), Memetic Algorithms (Moscato, 1989), Simulated Annealing (Fitzpatrick, Gelatt, & Vecchi, 1983), Tabu Search (Glover, 1989) etc., does indeed have a potentially significant role to play in allowing adaptive models for use in self-adaptive software systems to be developed. Genetic Programming as a general approach can and has been used to develop a more sophisticated "action set" in more complex situations (Cramer, 1985),(Koza, 1992),(Genetic Programming Inc, 2007). In terms of a more specific approach it may be that combinations of these approaches will need to be implemented to achieve optimum performance. Genetic Algorithms are effective at finding good global solutions, but perhaps not quite as effective as achieving optimum solutions. Therefore, a hybrid approach may be desirable in this particular instance.

The project work has demonstrated the potential applicability of cybernetic thinking, and in this instance specifically the S4 of the VSM (Viable System Model), to the development of autonomic computing systems. The principles of cybernetics and managerial cybernetics (such as Beer's VSM) would seem to be more generally applicable as a road-map to the development of autonomic self-adaptive software. It has provided a useful guide for the overall project leading to the exploration of techniques such as LCS. Many of the components outlined by IBM and the self-adaptive software community have previously also been considered by the cybernetic community when seeking to identify the general principles of control and communication required for organisms to survive in a changing environment. One element "missing" from the current prototype system is the S4 requirement of planning which would indicate a requirement for forecasting to be present in the system. This is no forecasting in the system as yet, but one possible approach would be the use of "anticipatory LCS" to provide this element.

In terms of the Law of Requisite Variety and the Law of Requisite Knowledge, we have used and also extended the definition of them to capture their essence when considered in a self-adaptive context. This reflects that there is a requirement not only to produce a model of the environment and "know" appropriate actions to respond to that environment but that this model and knowledge must be able to evolve in response to changing external and internal conditions.

Hence,

"Every good regulator of a system must be a current and evolving model of that system"

"In order to adequately compensate perturbations, a control system must "know" which action to select from the variety of available actions and be able to evolve new actions in response to as yet unknown environmental conditions"

This would suggest that to achieve the above requirements, there is a need for a genetically inspired programming approach to allow these models and system knowledge to evolve and adapt to environmental change.

The project work satisfies not only the original "classical" definitions as provided by Ashby (Conant & Ashby, 1970) and Aulin (Aulin, 1982) but also the revised requirement to evolve this concept of requisite variety and knowledge in response to environmental change.

The performance of the EMMA prototype demonstrated the applicability of the approach for holding current and adaptive models of the external environment and internal capability of the system. The performance of the prototype in maintaining "goal" performance levels was mixed, which highlights some of the issues of the approach. These include

- The issue of convergence with LCS
- The sensitivity of LCS to "tuning parameters"
- The importance of payoff strategies

The prototype was at times able to perform better than the static model and hybrid learning model experiments used as a measure of baseline performance. However, if the model had "hardened" then sometimes the prototype would struggle to "back-track" out of a particular set of rules. The results related to average fitness / best candidate (Figure 126 and Figure 127) would indicate that the payoff strategy requires further work be in order to improve the performance levels obtained by the prototype system.

Overall, the Environmental Modelling, Monitoring and Adaptive system (EMMA) demonstrated its ability to manage the major environmental modelling and control functions required by software systems. It emphasised the considerable challenges still facing researchers and software developers to solve large complex problem domains before the full realisation of the vision of truly autonomic environmental aware software systems.

8.2 Overall Evaluation of the Experimental Work

The experimental platform (Second Life) and subsequent problem definition (controlling performance on the virtual simulator region) was selected specifically for its suitability to this particular project work. It was apparent that many of the basic building blocks required by an environmental monitoring system were present, or could be developed in this particular experimental platform.

These building blocks included an ability to generate metrics from the environment using detectors, exporting these metrics to a suitable controlling system and broadcasting instructions back to effectors, which could enact the commands of the controlling system.

Whilst realising that all of these building blocks are not "easily obtained" in all the potential environments such systems could operate in, it has provided an effective platform for these particular experiments to take place.

8.3 Online Training of the Prototype System

The fact that the experiments were being conducted and evaluated on a "live" online system was both advantageous and problematic for a variety of reasons. In terms of the experimental work, one of the constraining factors was the time taken to actually run the experiments. Typically experiments in LCS systems consist of significant "off-line" training but the nature of this work meant that it would require all learning to be performed "on-line".

Typically, the prototype experiments were performed using between 25 – 100 action cycles in a single population generation and a range of 30 – 100 population generations.

A "short experiment" of 25 action cycles and 30 population generations would require 750 environmental readings from the virtual world simulator. A "longer experiment" of 100 action cycles and 100 generations would require 10,000 environmental readings from the virtual world simulator.

We would "typically" expect a single environmental reading to take approximately 1 minute. The reason for this was the requirement to measure time in integer values (due to the limitations of using Unix_Time as a key component of the metric performance measure) and therefore the requirement to measure FTime readings over a longer period to ascertain meaningful metric data.

Therefore a "short experiment" would take approx 750 minutes (12.5 hours) and a "long experiment" would take 10,000 minutes (167 hours which equates to approximately 1 week). This limited the ability to run longer experiments and it would be useful to observe the performance of the prototype system over a significantly longer period of time.

8.4 Comparison of Experimental Data

One difficulty related to the experimental data was the difficulty of comparing experiments. As the initial classifier set was randomly generated it meant that occasionally the prototype was provided with a particularly "good" initial classifier rule-set and conversely could potentially be given a "poor" initial classifier rule-set with which to perform the experiment.

Whilst over time the LCS is expected to explore the environmental space and overcome any initial disadvantage, this may take a number of population breeding and mutation cycles to occur. Giving the experiments the same initial classifier rule-set would not have helped significantly as the randomising nature of LCS means that the same initial classifier rule set would diverge relatively quickly due to the random nature of the process of crossover and mutation.

This meant that when observing experiments in LCS it was necessary to evaluate overall experimental trends rather than trying to compare them on a like-by-like basis.

8.5 Sensitivity of LCS to Parameter Tuning

One significant observation from the prototype experiments has been the sensitivity of LCS systems to initial parameter tuning. Literature currently provides little guidance on LCS parameter setting and it is often considered more of an art than a science with each application requiring experimentation to ascertain the parameters suited to that particular domain.

There were significant numbers of abandoned experiments during the research period due to the prototype being able to exploit the initial parameters in unexpected ways. This manifested itself in numerous experiments with

- a bid tax parameter that was too low, which encouraged classifier conditions to become less specific (increase in the number of wildcards within the condition) with the consequence of reducing the time required to achieve effective learning to occur.

- a life tax parameter being "fixed", which meant that in later populations, life tax had an increasingly insignificant effect due to the overall strength of the classifier set.

To some extent this element was reduced in later versions of the prototype where the taxation parameters could be scaled to the overall average strength of the population rule-set but changes to the pay-off strategy could lead the LCS to very different positions.

8.6 Scalability of the EMMA prototype

The scalability of the prototype system consisted of two main experiments.

The initial experiments consisted of a 10 bit condition element and a 16 bit message set (for each of the scripted objects). This meant that the initial experiments had 1024 potential environmental values (0 – 1024) and 65356 scripted object permutations.

The scalability experiments consisted of a 12 bit condition element and a 32 bit message set. The increased scalability experiments has 4096 potential environmental values (0 – 4096) and 4,294,967,296 scripted object permutations.

The results of these experiments demonstrated the ability of the LCS prototype to evolve models of the external environment and internal capability on larger scale problems but the "limits" of this approach were untested by these experiments. A reason for this was the experiment test-bed. As the experiment fundamentally worked by inducing lag through computational loading on the simulator region, it meant that each scaling experiment severely degraded similar performance.

Also, the actions available to the prototype consisted of the ability to switch a scripted object on or off. There is a need to extend the complexity by using a variety of actions (rather than a binary on / off approach). This would reflect the increasing complexity that systems are likely to face when autonomic self-adaptive software is deployed in real-world environments.

Therefore to test the scalability of the approach it would seem necessary for a different experimental approach to ascertain more effective results in this particular aspect.

8.7 Comparison with other approaches

There are many research approaches currently being proposed and evaluated in order to realise the vision of autonomic computing as envisaged by IBM in 2001.

The major issue remains that a unifying framework with which to develop autonomic systems remains tantalisingly out of reach. The thesis provides another interesting but relatively small scale implementation. However, the true value of the work may well be provided by the use of the VSM as a learning framework as inspiration for developing autonomic computer systems. Large scale system development, remain a possibility, if we can realise the implementation of a full VSM software system.

8.8 Summary

The evaluation of the project work and experimental results has demonstrated the potential of an evolutionary approach to developing models of the external environment and internal capability as demanded by S4 of the Viable System Model (VSM). It has also emphasised the potential value of "cross-disciplinary" approaches, such as managerial cybernetics, in providing both inspiration and "solutions" to the problems facing the autonomic computing community in finally producing

However, the evaluation also equally demonstrates the difficulties that exist and the requirement for further work and experimentation to test scalability and applicability of this approach in a range of application areas.

The experimental work related to the development of the prototype has provided an immensely interesting and, at times, frustrating challenge and has demonstrated the complexity of the challenges still facing researchers in this area.

It is clear that the ability to realise the vision of building a large scale, fully autonomic computing system still has many hurdles to its successful conclusion.

"I think and think for months and years. Ninety nine times, the conclusion is false. The hundredth time I am right"

Albert Einstein

Chapter 9: Conclusions and Future Research

This thesis has applied evolutionary approaches in the form of Holland's Learning Classifier Systems to produce models of both the external environment and an internal model of capability in a software system. The approach has extended the use of cybernetic theory into self-adaptive software development to explore an important component for producing systems that can adapt and evolve in response to environmental change.

This chapter is organised as follows. A summary of the thesis is presented in section 9.1. The main contributions provided by this research are presented in section 9.2. Future research is considered and proposed in section 9.3.

9.1 Thesis Summary

There is a requirement to provide adaptive autonomic software due to the increase in systems that are tethered to the "real-world". A key requirement of successful autonomic computer systems will be their ability, amongst other things, to effect change to respond to changes in performance measures, changes in the environment and recover effectively from failure. It seemed plausible to suggest that by analysing natural systems and how they maintain "robust performance" in the face of environmental turbulence, these naturally developed strategies could be applied when designing computer systems.

The "biological approach" to designing systems was analysed to discover lessons that could be learnt from successful natural systems such as the human autonomic nervous system, ant colonies and the Darwinian approach of natural selection when seeking to producing robust adaptive systems. These approaches were used as a roadmap to producing systems that can survive and indeed flourish in unknown and changing environments, which would previously have "broken" systems designed using traditional software development techniques.

Chapter 1 introduces the key concepts related specifically to the requirement for the development of software systems that have the ability to adapt to environmental change and the significant research challenges that remain unresolved in this area of software development.

The "Grand Challenge" of autonomic computing and its consequential vision, as presented by IBM in 2001, was discussed in further detail in Chapter 2. The challenges as outlined by IBM were analysed, as were the "Blueprint" documents produced by IBM to help in the development of effective solutions to these challenges. This provided an underpinning to subsequent chapters of the thesis regarding the particular requirements that an environmental modelling approach needs to consider. The emergence of the link with other disciplines such as biology and the theory of evolution were also initially presented in this chapter.

Chapter 3 began by providing an overview of the field of cybernetics and particularly managerial cybernetics. This chapter defined a cybernetic approach to system viability and argued that "classical" cybernetics has a valuable role to play in the development of robust system that can successfully "adapt" to unknown and changing environments. The applicability of managerial cybernetics and its relevance to this particular application was presented and discussed. It outlined the main features of the Viable System Model (VSM) and why it has considerable applicability to the development of adaptive software despite the fact that it was originally developed to provide cybernetic models of human organisations rather than in software development. We concentrated on System 4 of the VSM, which is specifically interested with the role of the environment in viable systems. We concluded that the natural world, cybernetics and particularly managerial cybernetics provide a "road-map" for developing systems that are robust and able to survive in changing environmental conditions.

An approach to mapping the original organisational cybernetic application to the VSM to software development was outlined and discussed with reference to the J-Reference Model, an application of the VSM to autonomic software system development. This approach sought to show the applicability of the VSM and cybernetic thinking to the problem of developing robust software that can effectively adapt to its environment. We sought to develop this idea further by looking in greater depth into the particular elements related to the environment and capturing a model of the environment during run-time.

In Chapter 3, we also sought to understand the nature and challenge of producing an environmental model by discussing the fundamental definition and role of the environment in greater detail. We started by providing an initial definition of what is meant by "the environment" and also what may be included in that environment. The difference between open and closed system environments are discussed including why the increasing complexity of software systems are making environments increasingly difficult to specify, thereby lending considerable weight to the necessity for environmentally adaptive software systems. The issue of the growing requirement for systems to hold some "model view" of their environment is considered and the reasons why this could be considered as fundamental for effective adaptive systems are outlined. Approaches to what should be "held" in a model of the environment were discussed, including the likely requirement to classify the environment by breaking it down into elements which currently affect the system, which may affect the system in the future and or that will not affect the system at all.

In Chapter 4, the use of learning classifier systems and genetic algorithms were discussed as an approach to allow the proposed system to model the environment, explore the environment and test optimising plans to learn (and continue to learn) the most appropriate responses to survive when facing environmental turbulence. We initially explained the theoretical underpinning behind the suitability of a genetic algorithm approach to exploring and ultimately modelling the environment. We explained how learning classifier systems use the concept of detectors, effectors and reinforcement learning to allow better rules to obtain a reward from the environment. This learning is supplemented by the use of a genetic algorithm approach to embed learning in the model by breeding "stronger" responses to the environmental over a period of time. The strength behind this approach is that it will allow complex environmental "spaces" to be efficiently and effectively explored by the system to develop appropriate responses to its current environment. We concluded that this lays the foundation of an approach to developing robust systems that are able to adapt to changing environmental conditions, whilst seeking continuous improvements. This is achieved by monitoring the current performance of the prototype in responding to the current environment. The classifier condition rule set at any particular moment in time therefore becomes a model of the visible external environment of the system. This enabled the model to capture the necessary richness, be adaptive and remove any reliance on our ability to "specify" our environment at design time. Rather, the system is allowed to explore its own environment and build its own model of the environment. Therefore the use of these technologies provides the fundamental building blocks of the Environmental Monitoring Modelling Adaptive system (EMMA).

Chapter 5 presented the test-bed for the experimental element of this research project. The design considerations behind the development of the environmental modelling control system and a selection of other experiments used to inform this research were described and analysed in detail. A variety of potential solutions to solving the issues related to monitoring the environment and providing appropriate responses to environmental change were discussed. This section outlined design considerations, challenges and explanations for the particular approaches taken within the thesis. We sought to design and implement static models and hybrid learning models to allow a variety of useful experiments to take place. These experiments enabled us to compare and contrast various approaches to modelling the environment. It sought to bring together the various technologies used in the project and show where and why key decisions were taken. The developed system used a number of technologies to demonstrate the building blocks required for the design of systems of this type and enabled us to generalise and conceptualise a model for the environmental modelling element of self-managing autonomic computer systems.

Chapter 6 presents the Environmental Modelling, Monitoring and Adaptive system (EMMA). The prototype demonstrated that while LCS and GAs are applicable in this particular situation, their use in other real world systems may be limited, although other evolutionary approaches may well be appropriate in those particular domains. The system was able to use high level goals specified by the user or indeed the environment itself and autonomically develop solutions which satisfy these higher level requirements.

The results presented within Chapter 7 demonstrate the ability of the system to monitor and model its environment and use this information to provide enhanced robustness. The system uses the concepts of learning classifiers and genetic algorithms to generate a rule set that is highly appropriate for the environment in which the system currently operates. The condition set element of the learning classifier becomes the current model of the state of the environment. These plans enable the virtual world environment to provide a better level of service to users in terms of the particular performance metrics used by this experiment. If the environment subsequently changes then the classifier set subsequently evolves using a Darwinian approach of "survival of the fittest" to enable the system to continue to survive and operate robustly within its current view of the environment.

Chapter 8 evaluates the environmental control centre prototype system development. The evaluation discusses the performance of the virtual world simulator when controlled by the Environmental Modelling, Monitoring and Adaptive system (EMMA) and compares and contrasts it with the results of the other approaches undertaken in this study. This chapter concludes that the EMMA prototype system provides a framework for providing a current model of both the external environment and internal model of capability. However, this approach could be applied to other software systems and hence this work could have more widespread applicability.

Finally, Chapter 9 presents the conclusions, a summary of contributions and opportunities for further work in this area of research. It is apparent that a cybernetic approach to environmental modelling has real value when combined with an effective learning mechanism. Specifically, we have developed a means of realising the learning provided by humans in the VSM, but have identified an approach to achieve this aspect in software. Whilst a virtual world platform has been used for the purposes of this research project, this should not exclude further development in a variety of software applications. As discussed, the main value of this research is identification of general approaches to environmental modelling that can be enhanced and exploited in the future in areas such as forecasting, which would allow pro-active rather than reactive adaptation to take place.

9.2 Summary of Contributions

This thesis presents a potential approach using a cybernetic underpinning supplemented by a learning approach to provide a method for modelling, not only the visible external environmental, but also a form of internal system capability. Using evolutionary approaches allows the models to adapt to current environmental conditions and therefore maintain their currency in response to changes observed in the environment.

1. The prototype system has provided an outline schema demonstrating that it is possible to provide systems with models of the environmental and internal capability that are learned and evolved online rather than the considerably more difficult task of specifying these models at design time. We proposed that by using computing approaches based on the study of natural systems (Learning Classifier Systems in this particular prototype system) systems can be provided with models that can be considered as current. These models can then be applied to produce systems that are able to make operational decisions based on the current view of the external environment and its own capability. This approach is generalised enough to allow subsequent research and experimentation with more complex environments to ascertain the applicability of learning approaches and evolutionary environmental modelling in larger scale computing challenges.

2. The application of classical cybernetic thinking to developing adaptive environmental models for use in autonomic systems provides a further research contribution. The use of Beers VSM as a roadmap to developing adaptive software systems that can adapt to changing environmental conditions and its own internal capability has demonstrated the relevance of cybernetic thinking to a modern software development challenge. Specifically, the use of System 4 of the VSM as a design approach when applied, in conjunction with LCS, to solving the issue of external and internal models is a novel application of two classic fields of computing to solve a recent research problem.

3. The extension of the Law of Requisite Knowledge and the Law of Requisite Variety to reflect their use in self-adaptive systems is a contribution by providing an updated definition, which emphasises the requirement of a continuous evolution of models and actions, allowing systems to remain robust in the face of changing environmental conditions.

4. Learning Classifier Systems as a discipline, although a long-standing area of computer science research, still requires further case-studies to demonstrate their applicability, or not, in areas as yet unexplored by researchers. Each of these areas can be considered to provide their own unique design challenges and therefore should be considered as contributions to the field in their own right. The use of LCS to control an online, open virtual world simulator is one such area and therefore this thesis should be considered as providing a contribution to the existing body of knowledge. This contribution will allow researchers to develop and advance in this area which would seem to have significant applicability given the likely requirement for software to operate in increasing complex environments in the future.

5. Learning Classifier Design Aspects. The design of the prototype has enabled some innovative LCS approaches to be tested and evaluated. The parent protection function implemented within the prototype system is a novel approach, within LCS development, to maintain the currency of the external environmental model. This approach seeks to protect "useful" condition elements by promoting their ability to breed into subsequent populations of the classifier rule-set and therefore allow increasingly relevant populations to develop. This approach has assisted the prototype system to maintain a more current external model of the environment, whilst evolving more appropriate responses to those environmental values. The prototype system also contains adaptive rates of chromosome replacement which are selected in response to the overall performance of the prototype system. This results in the lowering of the impact of the genetic algorithm element when the system is judged to be performing well and increasing it when the system requires more rapid change. This is achieved to ascertain performance in respect to the goal

required and scaling the proportion of the breeding of population as appropriate.

6. The applicability and usefulness of virtual world simulators as an experimental platform for research purposes in a secondary contribution of this thesis. The development of the prototype system, whilst not dependent on the experimental platform, has demonstrated the usefulness of such platforms for research purposes. The ability to simulate and control open and closed environments, to introduce environmental turbulence to experiments and to monitor and measure a variety of metrics has proved invaluable for this particular research problem. It would seem likely that other computing research problems may well be able to be modelled and therefore benefit from these environments in the future.

9.3 Future Research

For future research, it is planned to extend this work considerably in a number of directions. The thesis while demonstrating the potential of the VSM and LCS as building blocks in helping to produce self adaptive software has also indicated that further work is required to truly realise the vision of autonomic self-adaptive software systems. An outline of the most immediate areas of interest, in terms of future research, are presented below:-

1. **A Forecast of the Predicted Environment.** A full implementation of System 4 of the VSM requires a model of not only the current view of the external environment and internal capability, as provided by the current prototype system, but also a forecast of the predicted future state of the environment. The prototype system currently responds to a system of feed-back control to both develop the models and control performance in this particular application. If these models could be supplemented with an accurate future environmental forecast it would allow the use of feed-forward control, in order to inform, relevant pre-emptive action. This aspect should improve the performance of adaptive systems by using a system of avoidance measures rather than mitigation of potentially adverse environmental conditions.

2. **A System of Historical and Stored Learning.** The prototype currently obtains a target performance level (or goal) from the system environment and seeks to achieve it. If the target subsequently changes, prior learning, with respect to the previous target, is lost as the system evolves its model to meet the new performance level required. It seems wasteful to abandon the valuable learning that has already taken place. A potentially useful approach, when the target changes, would be to "store" the current degree of learning. As the classifier rule set currently provides the models of the external environment and internal capability this would effectively be achieved by archiving the classifier rule set. If the target subsequently matched one of the archived classifier sets, the previous classifier set could be re-introduced and the system allowed to continue its learning from its previous position. This would significantly reduce the learning time required for a new goal and would effectively "seed" the classifier rule set with an appropriate rule-set.

3. **Preventing Hardening in the Model and Timely Backtracking.** One of the major issues in all LCS implementations is convergence causing the model to harden, which can make it difficult for the system to "back-track" from this position. Whilst a LCS may eventually evolve out of this particular state, it relies on a combination of De Jongs crowding factor (De Jong, 1975) and mutation for this purpose. If we want to utilise LCS in online and time sensitive systems we may want to accelerate this process to enable faster "back-tracking". One potential method for this element would be to develop a metric for recognizing model hardness and develop strategies to backtrack more effectively. Potential strategies could be increased short-term high mutation rates or even the injection a number of random generated classifiers into the population to accelerate this process.

4. **The Exploration of Alternative LCS System Approaches.** The prototype system has been developed utilising the Michigan approach of developing LCS (as developed by Holland). The aim of this thesis was to evaluate learning as an environmental modelling approach rather than to compare LCS

approaches. However, there are a variety of different approaches of implementing LCS including Pittsburgh LCS and refinements of these systems such as ZCS which provide alternate aspects and potential benefits. Each of these approaches and refinements are worthy of individual experimentation to ascertain the potential benefits specific to these particular methods of LCS.

5. **Further Experimentation on Scalability and Complexity**. Future work should include experimentation on the scalability of this approach for increasingly complex environments. The strength of learning classifier systems, as a concept, are in its ability to explore complex search spaces effectively but as complexity is increased it is likely to unearth new research challenges and increase the understanding of the applicability of such systems. This experimentation would not just include increasing potential environmental states but also in meeting multiple and perhaps conflicting targets and co-operation between multiple adaptive systems to achieve higher level goals.

6. **Further expansion of the application of the VSM**. This body of work has concentrated in one particular aspects of autonomic computing, the interaction with the environment, and utilised System 4 of the VSM specifically as a roadmap to this particular problem domain. However it would be interesting to develop this research to demonstrate the wider applicability of the VSM in developing autonomic software. A logical next step would be to integrate System 5 (primarily concerning with Policy making) into the prototype system. There are aspects of element in the prototype such as choosing life tax rates, bid tax rates and proportion select rates but it could be developed much further to ascertain the value of the VSM as a road-map for more general advancement in autonomic and self adaptive software systems.

Bibliography

Abido, M. A., & Elazouni, A. M. (2010). Precedence Preserving GAs Operators for Scheduling Problems with Activities' Start Time Encoding. *Journal of Computing in Civil Engineering* , 345-356.

Achterbergh, J., & Vriens, D. (2002). Managing Viable Knowledge. *Systems Reseach and Behavioral Science* , 223-241.

Anthes, G. (2007). Second Life: Is There Any There There ? *December 3* .

Anthony, R. J. (2004). Emergence: A Paradigm for Robust and Scalable Distributed Applications. *Internation Conference on Autonomic Computing (ICAC'04).* IEEE.

Anthony, R., Pelc, M., Ward, P., Hawthorne, J., & Pulnah, K. (2008). A Run-Time Configurable Software Architecture for Self Managing Systems. *International Conference on Autonomic Computing* (pp. 207-208). IEEE.

Asaro, P. M. (2006). *Cybernetics.* Retrieved June 2010, from Peter Asaro Website: http://www.peterasaro.org/writing/cybernetics.html

Ashby, W. (1956). *An Introduction To Cybernetics.* New York: John Wiley.

Ashby, W. (1954). *Design of a Brain.* London: Chapman & Hall.

Aulin, A. (1982). *The Cybernetic Laws of Social Progress: Towards a Critical Social Philosophy and a Criticism of Marxism.* Oxford: Pergamon Press.

Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G. A., Ducatelle, F., Gambardella, L. M., et al. (2006). Design Patterns from Biology for Distributed Computing. *ACM Transactions on Autonomous and Adaptive Systems* , 26-66.

Bacardit, J., Mansilla, E. B., & Butz, M. V. (2008). Learning Classifier Systems: Looking Back and Glimpsing Ahead. *Lecture Notes in Computer Science* , 1-21.

Bahati, R. M., & Bauer, M. A. (2009). An Adaptive Reinforcement Learning Approach to Policy-driven Autonomic Management. *International Conference on Autonomic and Autonomous Systems* (pp. 135-141). IEEE.

Beckmann, B. E., Grabowski, L. M., McKinley, P. K., & Ofria, C. (2008). Autonomic Software Development Methodology Based on Darwinian Evolution. *International Conference on Autonomic Computing* (pp. 203-204). IEEE.

Beer, R. D., Chiel, H. J., & Sterling, L. S. (1990). A Biological Perspective on Autonomous Agent Design. *6*.

Beer, S. (1985). *Diagnosing The System For Organisations.* Chichester: John Wiley & Sons.

Beer, S. (1994). May the Whole Earth Be Happy: Loka Samastat Sukhino Bhavantu. *Interfaces* , 83-93.

Beer, S. (1975). *Platform For Change.* Wiley.

Beer, S. (1991). Reflections of a Cybernetician on the Practice of Planning. *Kybernetes* , *20* (6), 8-13.

Beer, S. (1991). Reflections of a Cybernetician on the Practice of Planning. *Kybernetes* , 8-13.

Beer, S. (1995). *The Heart of Enterprise.* Wiley.

Beer, S. (2004). World in Torment: A Time Whose Idea Must Come. *Kybernetes* , 774-803.

Bell, L., Pope, K., Peters, T., & Galik, B. (2007). In Second Life ? *July / August* .

Bernard, C. (1879). Lectures on the Phenomena of Animals and Plants.

Bigus, J., Schlonnagle, D., Pilgrim, J., Mills III, W., & Diao, Y. (2002). ABLE: A toolkit for building multiagent autonomic systems. *IBM Systems Journal* , *41* (3), 350-371.

Bisadi, M., & Sharifi, M. (2009). Component-Based Self-Healing via Cellular Adaptation. *Internation Conference on Autonomic and Autonomous Systems* (pp. 75-81). IEEE.

Black, S., Boca, P. P., Bowen, J. P., Gorman, J., & Hinchey, M. (2009, September). Formal Versus Agile: Survival of the Fittest. *IEEE Computer*, pp. 37-45.

Booker, L. B., Goldberg, D. E., & Holland, J. H. (1989). Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 235-282.

Bose, P., & Matthews, M. (2000). *An Agent Mediated Approach To Dynamic Change in Co-ordination Policies.* Mitre Technical Paper.

Bourke, P. (2009). Evaluating Second Life For The Collaborative Exploration of 3D Fractals. (33).

Bousquet, O., Boucheron, S., & Lugosi, G. (2004). Introduction to Statistical Learning Theory. *Advanced Lectures on Machine Learning Lecture Notes in Artificial Intelligence*, 169-207.

Bowker, G., & Chou, R.-S. (2009). Ashby's notion of memory and the ontology of technical evolution. *Internation Journal of General Systems*, 129-137.

Bratman, R. A., Israel, D. J., & Pollack, M. E. (1988). Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, 349-355.

Bull, L. (2004). *Applications of Learning Classifier Systems.* Springer.

Bull, L., & Hurst, J. (2002). ZCS Redux. *Evolutionary Computation*, 185-205.

Bull, L., & Lanzi, P. L. (2009). Introduction to the special issue on learning classifier systems. *Natural Computing*, 1-2.

Bull, L., Budd, A., Stone, C., Uroukov, I., Costello, B. d., & Adamatzky, A. (2008). Towards Unconventional Computing through Simulated Evolution: Control of Nonlinear Media by a Learning Classifier System. *Artificial Life*, 203-222.

Burgess, M. (2007). *Biology, Immunology and Information Security.* Elsevier - Information Security Technical Report 12.

Butz, M. V., & Wilson, S. W. (2002). An Algorithmic Description of XCS. *Soft Computing*, 144-153.

Calinescu, R. (2007). Model-Drive Autonomic Architecture. *Fourth International Conference on Autonomic Computing (ICAC'07).* IEEE.

Cannon, W. B. (1932). *The Wisdom of the Body.*

Chang, Y.-H., Ho, T., & Kaelbling, L. P. (2004). Mobilized ad-hoc networks: A reinforcement learning approach. *International Conference on Autonomic Computing (ICAC'04).* IEEE.

Chen, L., & Agrawal, G. (2004). Self-Adaptation in a Middleware for Processing Data Streams. *International Conference on Autonomic Computing.*

Chess, D. M., Segal, A., Whalley, I., & White, S. R. (2004). Unity: Experiences with a Prototype Autonomic Computing System. *International Conference on Autonomic Computing (ICAC'04).* IEEE.

Clark, D. D., Partridge, C., Ramming, J. C., & Wroclawski, J. T. (2003). A Knowledge Plane for the Internet. *Special Interest Group on Data Communication (SIGCOMM)*, (pp. 3-10).

Colombetti, M., & Dorigo, M. (2000). What is a Learning Classifier ? *Lecture Notes in Computer Science*, 3-32.

Conant, R. C., & Ashby, W. R. (1970). Every Good Regulator of a System Must Be a Model Of That System. *International Journal of Systems Science*, 89-97.

Conant, R., & Ashby, W. (1970). Every good regulator of a system must be a model of that system. *International Journal of Systems Science*, 89-97.

Cooter, M. (2010, February). Systematic Well-Being. *Engineering & Technology* , pp. 48-50.

Cramer, N. L. (1985). A Representation for the Adaptive Generation of Simple Sequential Programs. *International Conference on Genetic Algorithms and their Applications* (pp. 183-187). Pittsburgh, USA: ICGA.

Das, R., Kephart, O. J., Lefurgy, C., Tesauro, G., Levine, W. D., & Chan, H. (2008). Autonomic Multi-Agent Management of Power and Performance in Data Centers. *Proc. of 7th Int Conf. on Autonomous Agents and Multiagent Systems (AA-MAS 2008)* (pp. 107-114). Estoril, Portugal: Internation Foundation for Autonomous Agents and Multiagent Systems.

Dayan, P., & Watkins, C. J. (2001). *Reinforcement Learning.* London: Macmillian Press.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems (Doctoral Thesis).* Michigan: University of Michigan.

Dean, T., Allen, J., & Alimonon, Y. (1995). *Artificial Intelligence: Theory and Practice.* Benjamin Cummings.

Dictionary of Cybernetics and Systems. (2010). Retrieved June 4, 2010, from Technology Dictionary: http://miscellaneous.techdictionary.org/Dictionary-of-Cybernetics-and-Systems/regulation

Dixon, K., Pham, T., & Khosla, P. (2001). Port-Based Adaptable Agent Architecture. *Lecture Notes in Computer Science , 1936*, 181.

Dobson, S., Sterritt, R., Nixon, P., & Hinchey, M. (2010, January). Fulfilling the Vision of Autonomic Computing. *IEEE Computer* , pp. 35-41.

Epshteyn, A., Vogel, A., & DeJong, G. (2008). Active Reinforcement Learning. *25th International Conference on Machine Learning.* Helsinki, Finland.

Fitzpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *New Science* , 671-680.

Foerster, H. V. (2003). Cybernetics of Cybernetics. In *Understanding Understanding: Essays on Cybernetics and Cognition* (pp. 283-287). Springer.

Foss, J. (2009). Lessons From Learning In Virtual Environments. *40* (3).

François, C. (1999). Systemics and Cybernetics in a Historical Perspective. *Systems Research and Behavioral Science* , 203-219.

Gallopin, G. (1981). The Abstract Concept of Environment. *International Journal of General Systems* , 139-149.

Gao, Y., Zhexue Huang, J., & Wu, L. (2007). Learning classifier system ensemble and compact rule set. *19* (4).

Gartner. (2007, April 24). *Press Release.* Retrieved April 2009, from Gartner Inc: http://www.gartner.com/it/page.jsp?id=503861

Genetic Programming Inc. (2007). Retrieved April 2010, from Genetic Programming Inc: http://www.genetic-programming.com

Ginis, R., & Strom, R. (2004). An Autonomic Messaging Middleware with Stateful Stream Transformation. *International Conference on Autonomic Computing (ICAC'04).* IEEE.

GIWG. (2001). *A Proposal For DASADA Gauge Infrastructure Working Group Draft 1.0.* ABLE Research Group, Carnegie Mellon University.

Glover, F. W. (1989). Tabu Search - Part I. *ORSA Journal on Computing* , 190-206.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation & Machine Learning.* Addison Wesley.

Goldsby, H. J., Cheng, B. H., McKinley, P. K., Knoester, D. B., & Ofria, C. A. (2008). Digital Evolution of Behavioural Models for Autonomic Systems. *International Conference on Autonomic Computing* (pp. 87-96). IEEE.

Gou, L. (1994). Holonic Planning and Scheduling fo a Robotic Assembly Test Bed. *Proceedings of the 4th International Conference on Computer Integrated Manufacturing.* Troy, New York.

Gregory, A. (2007). Target setting, lean systems and viable systems: a systems perspective on control and performance management. *Journal of the Operational Research Society*, 1503-1517.

Griffith, R., & Kaiser, G. (2006). A Runtime Adaptation Framework for Native C and Bytecode Applications. *Internation Conference on Autonomic Computing* (pp. 93-104). IEEE.

Gronstedt, A. (2007). *Second Life Produces Real Training Results.* American Society for Training & Development.

Hariri, S., Khargharia, B., Chen, H., Yang, J., & Zhang, Y. (2006). The Autonomic Computing Paradigm. *Cluster Computing*, 5-17.

Heylighen, F. (1993, August). *Cybernetics.* Retrieved January 2010, from Principia Cybernetica Web (Principia Cybernetica, Brussels): http://pespmc1.vub.ac.be/cybern.html

Heylighen, F. (2001). Cybernetics and Second-Order Cybernetics. In R. A. Meyers, *Encyclopedia of Phyical Science & Technology (3rd Edition).* New York: Academic Press.

Heylighen, F. (1992). Principles of Systems and Cybernetics: an evolutionary perspective. *Cybernetics and Systems '92*, 3-10.

Heylighen, F. (2001, September). *The Law of Requisite Knowledge.* Retrieved May 2010, from Principia Cybernetica Web (Principia Cybernetica, Brussels): http://pespmc1.vub.ac.be/REQKNOW.html

Heylighen, F. (2000, January). *The Principle of Incomplete Knowledge.* Retrieved June 2010, from Principia Cybernetica Web (Principia Cybernetica, Brussels): http://pespmc1.vub.ac.be./%5EINCOMKNO.html

Holland, J. (2003). A Derived Markov Process for Modeling Reaction Networks. *Evolutionary Computing*, 339-362.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* Cambridge, Massachusetts: MIT Press.

Holland, J. (2000). Building Blocks, Cohort Genetic Algorithms and Hyperplane-Defined Functions. *Evolutionary Computation*, 373-391.

Holland, J. (1986). Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. *Machine Learning: An Artificial Intelligence Approach*, 593-623.

Holland, J. L., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In D. A. Waterman, & F. Hayes-Roth, *Pattern Directed Inference Systems.* New York: Saunders College Publishing.

Holland, J. (2000). What is a Learning Classifier System ? *Lecture Notes in Computer Science*, 3-32.

Horn, P. (2001, October). *Autonomic Computing: IBM's perspective on the State of Information Technology.* Retrieved from http://www.research.ibm.com/autonomic

Huebscher, M. C., & McCann, J. A. (August 2008). A Survey of Autonomic Computing - Degrees, Models and Applications. *ACM Computing Surveys*, 7 - 7:28.

Hurst, J., & Bull, L. (2006). A Neural Learning Classifier System with Self-Adaptive Constructivism for Mobile Robot Control. *Artificial Life*, 353-380.

IBM. (2006). *An Architectural Blueprint For Autonomic Computing 4th Edition.*

IBM. (2008, November). *Autonomic Computing : Overview.* Retrieved from http://www.research.ibm.com/autonomic/overview/elements.html

IBM. (2006). *IBM Autonomic Computing.* Retrieved July 2008, from http://www-01.IBM.Com/Software/tivoli/autonomic/pdfs/ac_blueprint_white_paper_4th.pdf

Jackson, M. (1991). *Systems Methodology for the Management Sciences.* London: Plenum Press.

Jarmon, L., Traphagan, T., Mayrath, M., & Trivendi, A. (2009). Virtual World Teaching, experiential learning, and assessment: An interdisciplinary communication coursein Second Life. *53.*

Jarrett, C. (2009). Get A Second Life. *22* .

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* , 237-285.

Kandasamy, N., Abdelwahed, S., & Hayes, J. P. (2004). Self-Optimization in Computer Systems via Online Control: Application to Power Management. *International Conference on Autonomic Computing.*

Kephart, J. O., & Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer* , *36* (1), 41-50.

Kephart, J. O., Chan, H., Das, R., Levine, D. W., Tesauro, G., Rawson, F., et al. (2007). Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs. *Fourth International Conference on Autonomic Computing (ICAC'07).* IEEE.

Kernighan, B., & Plauger, P. J. (1976). *Software Tools.* Addison-Wesley.

Kesaniemi, J., Katasonov, A., & Terziyan, V. (2009). An Observable Framework for Multi-Agent Systems. *Fifth International Conference on Autonomic and Autonomous Systems* (pp. 336-341). IEEE.

Khalid, A., Haye, M. A., Khan, M. J., & Shamail, S. (2009). Survey of Frameworks, Architectures and Techniques in Autonomic Computing. *5th International Conference on Autonomic and Autonomous Systems* (pp. 220-225). IEEE.

Kharbat, F., Bull, L., & Odeh, M. (2005). Revisiting Genetic Selection in the XCS Learning Classifier System. *Evolutionary Computation* , 2061-2068.

Kiciman, E., & Wang, Y.-M. (2004). Discovering Correctness Constraints for Self-Management of System Configuration. *International Conference on Autonomic Computing.* IEEE.

Klir, G., & Valach, M. (1967). *Cybernetic Modelling.* London: Illife Books.

Kovacs, T., & Kerber, M. (2006). A Study of Structural and Parametric Learning in XCS. *Evolutionary Computation* , 1-19.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection.* Cambridge, MA: MIT Press.

Kumar, V., Cai, Z., Cooper, B. F., Eisenhauer, G., Schwan, K., Mansour, M., et al. (2006). Implementing Diverse Messaging Models with Self-Managing Properties using IFLOW. *Internation Conference on Autonomic Computing* (pp. 243-252). IEEE.

Laddaga, R. (1999). Creating Robust Software through Self-Adaptation. *Intelligent Systems & Their Applications* , *14* (3), 26-29.

Laddaga, R., & Robinson, P. (2000). Model Based Diagnosis in Self Adaptive Software. *Test* , ???

Laddaga, R., Robinson, P., & Shrobe, H. (1999). Results of the First International Workshop on Self Adaptive Software. *1st Internation Workshop on Self Adaptive Software.*

Landau, S., & Sigaud, O. (2008). A comparison between ATNoSFERES and Learning Classifier Systems on non-Markov problems. *Information Sciences* , 4482-4500.

Langlois, M., & Sloan, R. H. (2010). Reinforcement learning via approximation of the Q-function. *Journal of Experimental & Theoretical Artificial Intelligence* , 219-235.

Laws, A. G., Taleb-Bendiab, A., Wade, S. J., & Reilly, D. (2003). From Wetware to Software: A Cybernetic Perspective of Self-Adaptive Software. *Self-Adaptive Software: Applications, Second International Workshop on Self-Adaptive Software.*

Lehman, M. M. (1998). Feedback, Evolution and Software Technology - The Human Dimension. *ICSE 20 Workshop* (pp. 1-9). Kyoto, Japan: ICSE.

Lehman, M. M., & Weir, M. (1980). Progams, Life Cycles, and Laws of Software Evolution. *Proceedings of IEEE* (pp. 1060-1076). IEEE.

Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., & Turski, W. M. (1997). Metrics and Laws of Software Evolution - The Nineties View. *Process Metrics 97.* Alburquerque, New Mexico.

Leonard, A. (2009). The Viable System Models and Its Application to Complex Organisations. *Syst, Pract Action Res.* , 223-233.

Lewis, G. J. (1997). A Cybernetic View of Environmental Management: The Implications For Business Organisations. *6.*

Lin, W.-Y., Lee, W.-Y., & Hong, T.-P. (2003). Adapting Crossover and Mutation Rates in Genetic Algorithms. *Journal of Information Science and Engineering* , 889-903.

Littman, M. L., Ravi, N., Fenson, E., & Howard, R. (2004). Reinforcement Learning for Autonomic Network Repair. *International Conference on Autonomic Computing (ICAC'04).* IEEE.

Lorenz, E. N. (1972). Predictability; Does the Flap of a Butterfly's wings in Brazil Set off a Tornado in Texas? *American Association for the Advancement of Science, 139th Meeting.* AAAS.

Mead, G. (1934). *Mind, Self and Society.* Chicago Univ. Press.

Menasce, D. A., & Kephart, J. O. (2007, February). Autonomic Computing. *IEEE Internet Computing* , pp. 18-21.

Meng, A. (2000). On Evaluating Self-Adaptiv Software. *First International Workshop on Self-Adaptive Softwre.* IWSAS 2000.

Miorandi, D., Yamamoto, L., & De Pellegrini, F. (2010). A Survey of Evolutionary and Embryogenic Approaches to Autonomic Networking. *Computer Networks* , 944-959.

Morgan, G. (1986). *Images of Organisation.* London: Sage.

Moscato, P. (1989). *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms.* Pasadena: Caltech Concurrent Computation Program.

Nagpal, R. (2004). A Catalog of Biologically-inspired Primitives for Engineering Self-Organization. In *Lecture Notes in Computer Science* (pp. 53-62). Springer Berlin / Heidelberg.

Oxford English Reference Dictionary. (2002). Oxford: OUP Oxford.

Parashar, M., & Hariri, S. (2005). Autonomic Computing: An Overview. *Unconventional Programming Paradigms* , *3566*, 257-269.

Patten, B. (1978). Systems Approach To The Concept of Environment. *Journal of Science* , 206-278.

Powers, M. A. (1984). The Rise and Fall of Cybernetics As Seen In The Evolution of the Dewey Decimal System. *20th Annual Meeting of the American Society for Cybernetics.*

Reilly, D., & Taleb-Bendiab, A. (2002). Dynamic Instrumentation for Jini Applications. *Software Engineering and Middleware, 3rd International Workshop.* Orlando, FL.

Reynolds, R. G. (1994). An Introduction to Cultural Algorithms. *Proceedings of the 3rd Annual Conference on Evolutionary Programming* (pp. 131-139). World Scientific Publishing.

Robinson, P. (2000). An Architecture for Self-Adaptation and its Application to Aerial Image Understanding. *First Internation Workshop on Self-Adaptive Software* (pp. 199-223). IWSAS 2000.

Rojanavasu, P., Dam, H. H., Abbass, H., Lokan, C., & Pinngern, O. (2009). A Self-Organised, Distributed, and Adaptive Rule-Based Induction System. *IEEE Transactions on Neural Networks* , 446-459.

Saggu, J. (1996). *Factory Agile.* Retrieved November 2009, from European Agile Enterprise: http://www.sarc.sk/akcie/data/1548.html

Salmon, G. (2009). The Future for (Second) Life and Learning. *40* (3).

Sanchez, J. (2009). A Social History of Virtual Worlds. *February / March* .

Sarkar, B. K., & Sana, S. S. (2009). A hybrid apprach to design efficent learning classifiers. *58.*

Schuhmann, W. (2004). Observing Experiences with the VSM. *Kybernetes* , *33* (3/4), 609-631.

Schulenburg, S., & Ross, P. (2002). A Learning Evolutionary Trading System LETS. *Genetic and Evolutionary Computation Conferences* (pp. 45-53). New York: GECCO 2002.

Schwaninger, M. (2006). Theories of Viability: a Comparison. *Systems Reseach and Behavioral Science* , 337-347.

Schwaninger, M. (2004). What Can Cybernetics Contribute to the Conscious Evolution of Organizations and Society. *Systems Research and Behavioral Science* , 515-527.

Segarra, M. T., & Andre, F. (2009). Building a Context-Aware Ambient Assisted Living Application Using a Self-Adaptive Distributed Model. *Fifth International Conference on Autonomic and Autonomous Systems* (pp. 40-44). IEEE.

Serugendo, G. D., Foukia, N., Hassas, S., Karageorgos, A., Mostefaoui, S. K., Rana, O. F., et al. (2004). Self-Organisation: Paradigms and Applications. In *Lecture Notes in Computer Science* (pp. 1-19). Springer Berlin / Heidelberg.

Seshia, S. A. (2007). Autonomic Reactive Systems via Online Learning. *International Conference on Autonomic Computing (ICAC'07).* IEEE.

Shen, W., & Norrie, D. H. (1999). Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey. *Knowledge and Information Systems* , 129-156.

Smart, J., Cascio, J., & Paffenduff, J. (2007). *Metaverse Roadmap: Pathways to the 3D Web.* Retrieved January 2009, from www.metaverseroadmap.org: http://www.metaverseroadmap.org/overview

Stalph, P. O., Butz, M. V., & Goldberg, D. E. (2009). On the Scalability of XCS(F). *GECCO '09 Proceeding of the 9th Annual Conference on Genetic and Evolutionary Computation* (pp. 1315-1322). Montreal, Quebec: ACM.

Stewart, I. (2002). *Does God Play Dice ? The New Mathematics of Chaos.* Oxford: Blackwell Publishing.

Strassner, J., Samudrala, S., Cox, G., Liu, Y., Jiang, M., Zhang, J., et al. (2008). The Design of a New Context-Aware Policy Model for Autonomic Networking. *Fourth International Conference on Autonomic and Autonomous Systems* (pp. 119-130). IEEE.

Taton, C., De Palma, N., Philippe, J., & Bouchenak, S. (2007). Self-Optimizarion of Clustered Message-Orientated Middleware. *International Conference on Autonomic Computing (ICAC'07).* IEEE.

Tesauro, G., Jong, N. K., Das, R., & Bennani, M. N. (2006). A Hybrid Reinformance Learning Approach to Autonomic Resource Allocation. *International Conference on Autonomic Computing* (pp. 65-73). 2006: IEEE.

Troc, M., & Unold, O. (2010). Sele-Adaptation of Parameters in a Learning Classifier System Ensemble Machine. *Internation Journal of Applied Mathemtics in Computer Science* , 157-174.

Trumler, W., Petzold, J., Bagci, F., & Ungerer, T. (2004). AMUN - Autonomic Middleware for Ubiquitious eNvironments Applied to the Smart Doorplate Project. *International Conference on Autonomic Computing.*

Valetto, G., Kaiser, G., & Phung, D. (2005). A Uniform Programming Abstraction for Effecting Autonomic Adaptations onto Software Systems. *2nd International Conference on Autonomic Computing.* IEEE.

Van Valen, L. (1973). A New Evolutionary Law. *Evolutionary Theory* , 1-30.

Vassev, E., & Hinchey, M. (2009, June). ASSL: A Software Engineering Approach to Autonomic Computing. *IEEE Computer* , pp. 90-93.

Walsh, W. E., Tesauro, G., Kephart, J. O., & Das, R. (2004). Utility Functions in Autonomic Systems. *International Conference on Autonomic Computing (ICAC'04).* IEEE.

Warburton, S. (2009). Second Life in Higher Education : Assessing the potential for and the barriers to depolying virtual worlds in learning and teaching. *40* (3).

White, S. R., Hanson, J. E., Whalley, I., Chess, D. M., & Kephart, J. O. (2004). An Architectual Approach to Autonomic Computing. *International Conference on Autonomic Computing.* IEEE.

Wiener, N. (1954). *The Human Use of Human Beings: Cybernetics and Society.* London: Eyre and Spottiswoode.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation* , 149-176.

Wilson, S. (1994). ZCS: A Zeroth Level Classifier System. *Evolutionary Computation* , 1-18.

Wu, A. S., Lindsay, R. K., & Riolo, R. L. (1997). Empirical Observations on the Roles of Crossover and Mutation. *Proceedings of the 7th International Conference on Genetic Algorithms*, (pp. 362-369). East Lansing, MI, USA.

Xu, J., Zhao, M., Fortes, J., Carpenter, R., & Yousif, M. (2007). On the Use of Fuzzy Modelling in Virtualized Data Center Management. *Fourth International Conference on Autonomic Computing.* IEEE.

Xue, Y., Wang, Q., Ng, B. L., Swerdlow, H., Burton, J., Skuce, C., et al. (2009). Human Y Chromosome Base-Substitution Mutation Rate Measure By Direct Sequencing In A Deep Routing Pedigree. *Current Biology* , 1453-1457.

Yamada, T., & Yamaguchi, S. (2010). Reinforcement Learning Using A Stochastic Gradient Method with Memory-Based Learning. *Electrical Engineering in Japan* , 32-40.

Yeung, K. F., Yang, Y., & Ndzi, D. (2009). A Context-aware System for Mobile Data Sharing in hybrid P2P environment. *Fifth International Conference on Autonomic and Autonomous Systems* (pp. 63-68). IEEE.

Zhang, Z., Lin, S., Lian, Q., & Jin, C. (2004). RepStore: A Self-Managing and Self-Tuning Storage Backend with Smart Bricks. *International Conference on Autonomic Computing.* IEEE.

Zhu, Q., & Agrawal, G. (2008). An Adaptive Middleware for Supporting Time-Critical Event Response. *Internation Conference on Autonomic Computing* (pp. 99-108). IEEE.

Ziegler, A., Santos, P., Kellermann, T., & Uchanska-Ziegler, B. (2009). Assuring Survival of the fittest through reproduction. *Journal of Reproductive Immunology* , 113-175.

Zimmerman, M. (2009). Why Evolution is the Organising Principle for Biology. *Phi Kappa Phi Forum* , 5-7.

Appendix A Program Design Flowcharts

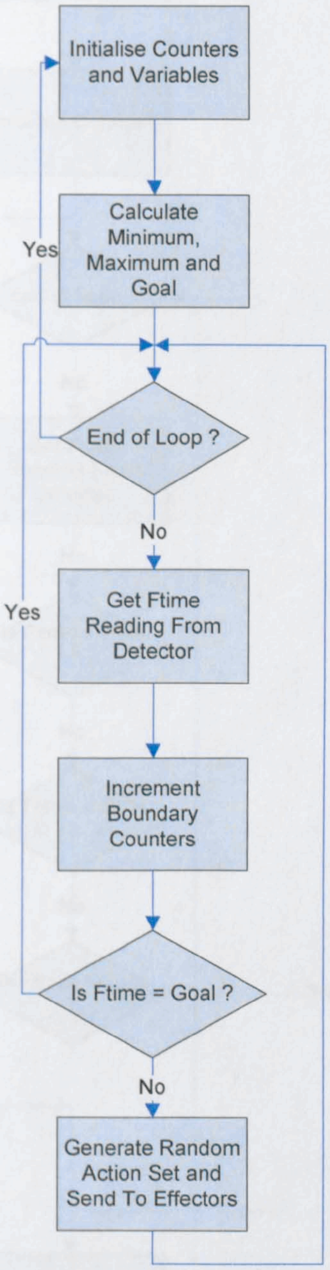

Initialise Counters and Variables
Calculate Minimum, Maximum and Goal
Yes
End of Loop ?
No
Get Ftime Reading From Detector
Yes
Increment Boundary Counters
Is Ftime = Goal ?
No
Generate Random Action Set and Send To Effectors

Figure 137 Outline design of random experiment

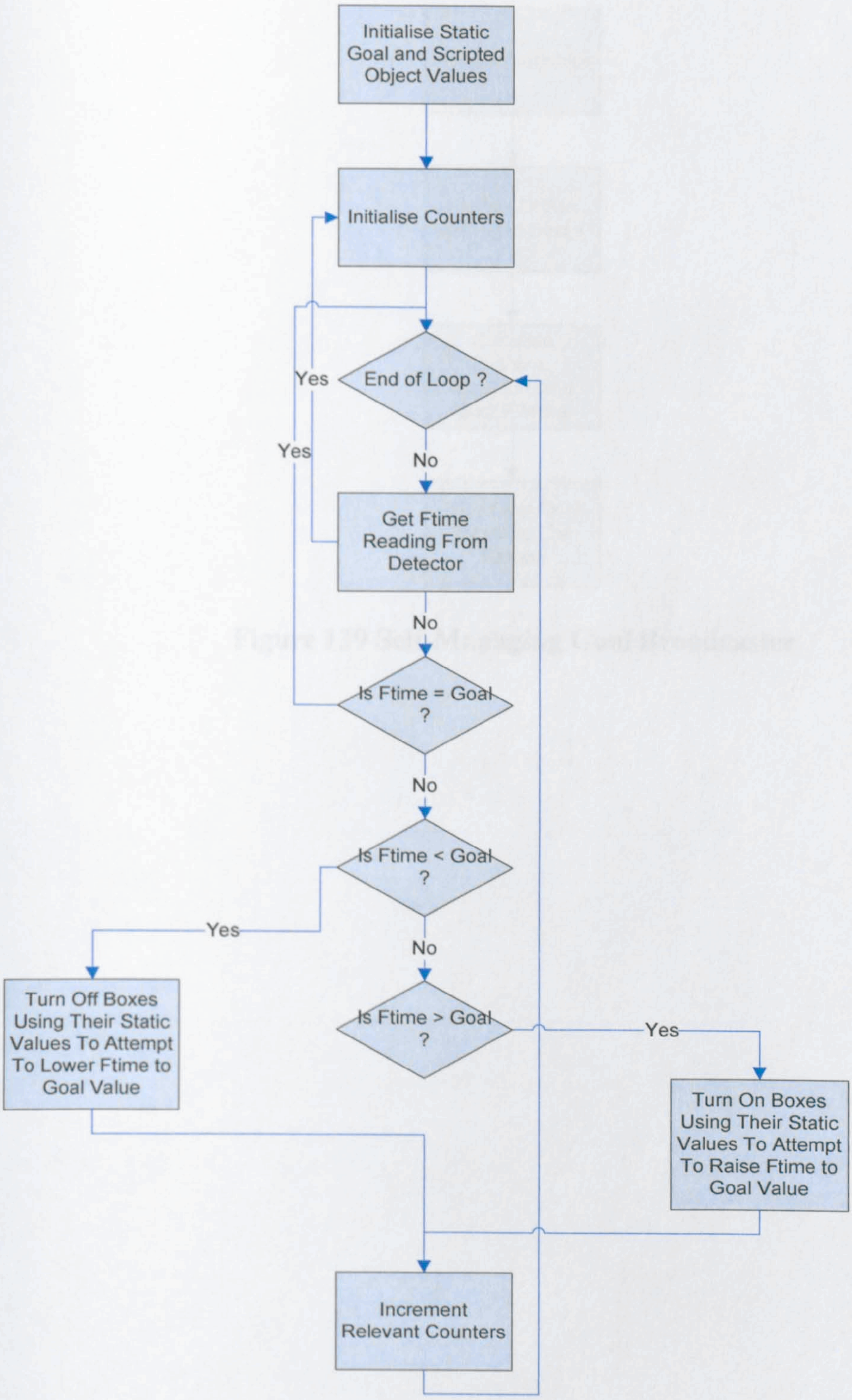

Initialise Static Goal and Scripted Object Values
Initialise Counters
Yes
End of Loop ?
Yes
No
Get Ftime Reading From Detector
No
Is Ftime = Goal ?
No
Is Ftime < Goal ?
Yes
No
Turn Off Boxes Using Their Static Values To Attempt To Lower Ftime to Goal Value
Is Ftime > Goal ?
Yes
Turn On Boxes Using Their Static Values To Attempt To Raise Ftime to Goal Value
Increment Relevant Counters

Figure 138 Outline design of Static Model Experiment

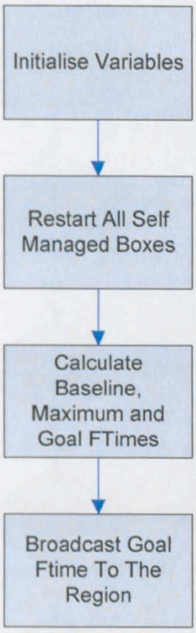

Initialise Variables
Restart All Self Managed Boxes
Calculate Baseline, Maximum and Goal FTimes
Broadcast Goal Ftime To The Region

Figure 139 Self-Managing Goal Broadcaster

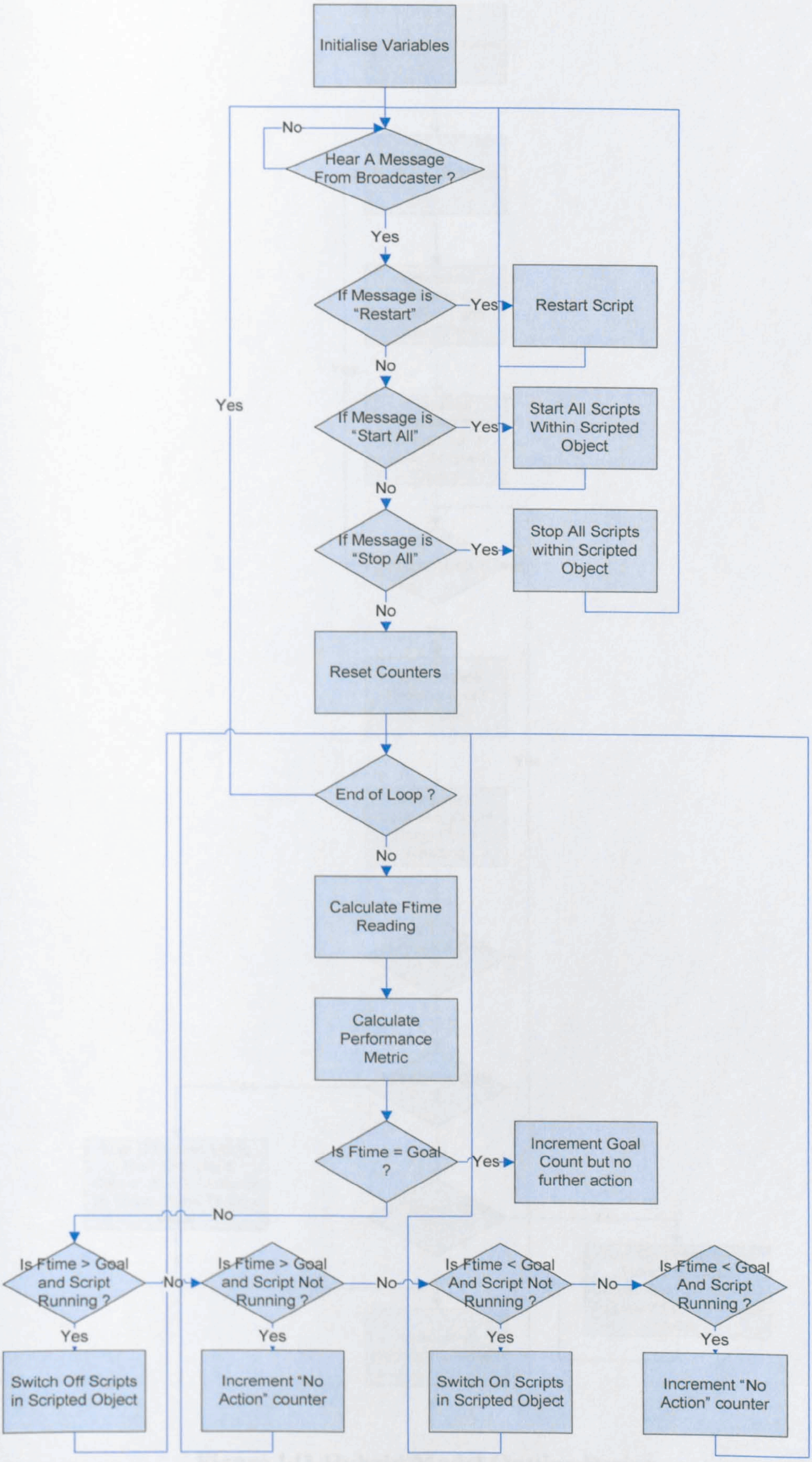

Initialise Variables
No
Hear A Message From Broadcaster ?
Yes
If Message is "Restart"
Yes
Restart Script
No
Yes
If Message is "Start All"
Yes
Start All Scripts Within Scripted Object
No
If Message is "Stop All"
Yes
Stop All Scripts within Scripted Object
No
Reset Counters
End of Loop ?
No
Calculate Ftime Reading
Calculate Performance Metric
Is Ftime = Goal ?
Yes
Increment Goal Count but no further action
No
Is Ftime > Goal and Script Running ?
No
Is Ftime > Goal and Script Not Running ?
No
Is Ftime < Goal And Script Not Running ?
No
Is Ftime < Goal And Script Running ?
Yes
Switch Off Scripts in Scripted Object
Yes
Increment "No Action" counter
Yes
Switch On Scripts in Scripted Object
Yes
Increment "No Action" counter

Figure 140 Self Managing Box Outline Design

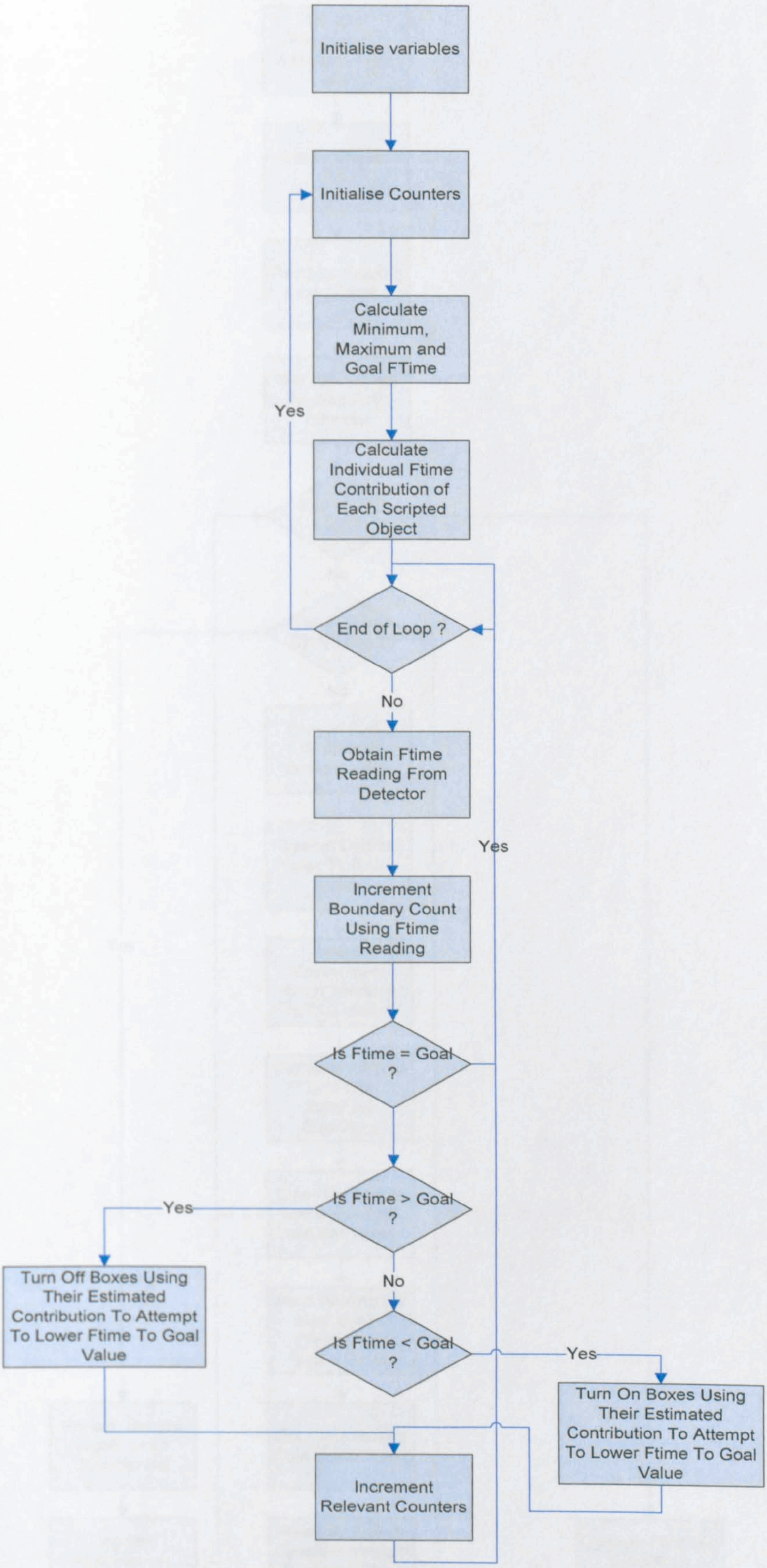

Initialise variables
Initialise Counters
Calculate Minimum, Maximum and Goal FTime
Yes
Calculate Individual Ftime Contribution of Each Scripted Object
End of Loop ?
No
Obtain Ftime Reading From Detector
Yes
Increment Boundary Count Using Ftime Reading
Is Ftime = Goal ?
Yes
Is Ftime > Goal ?
Turn Off Boxes Using Their Estimated Contribution To Attempt To Lower Ftime To Goal Value
No
Is Ftime < Goal ?
Yes
Turn On Boxes Using Their Estimated Contribution To Attempt To Lower Ftime To Goal Value
Increment Relevant Counters

Figure 141 Hybrid Model Outline Design

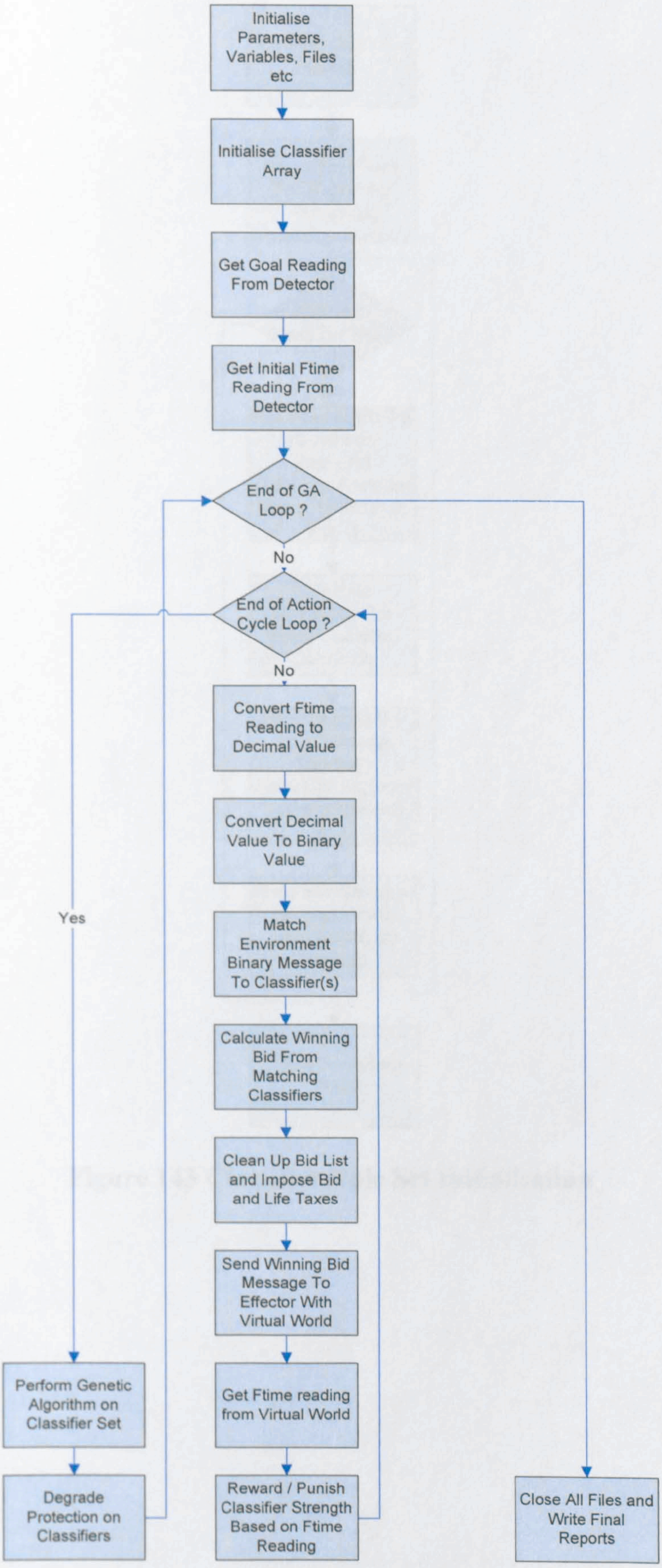

Initialise Parameters, Variables, Files etc
Initialise Classifier Array
Get Goal Reading From Detector
Get Initial Ftime Reading From Detector
End of GA Loop ?
No
End of Action Cycle Loop ?
No
Yes
Convert Ftime Reading to Decimal Value
Convert Decimal Value To Binary Value
Match Environment Binary Message To Classifier(s)
Calculate Winning Bid From Matching Classifiers
Clean Up Bid List and Impose Bid and Life Taxes
Send Winning Bid Message To Effector With Virtual World
Get Ftime reading from Virtual World
Reward / Punish Classifier Strength Based on Ftime Reading
Perform Genetic Algorithm on Classifier Set
Degrade Protection on Classifiers
Close All Files and Write Final Reports

Figure 142 The outline design of the EMMA Prototype System

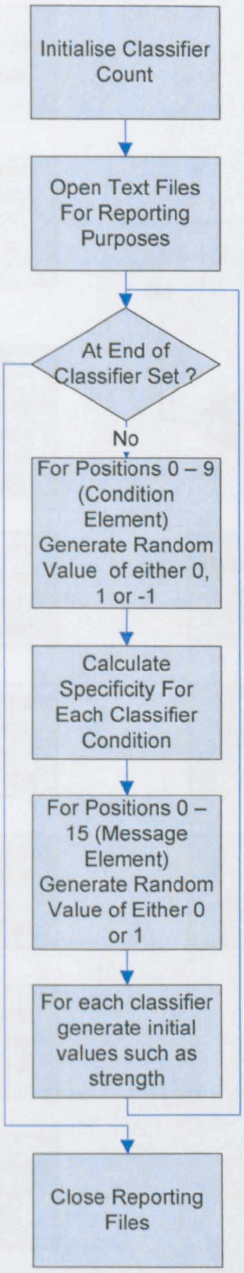

Initialise Classifier Count
Open Text Files For Reporting Purposes
At End of Classifier Set ?
No
For Positions 0 – 9 (Condition Element) Generate Random Value of either 0, 1 or -1
Calculate Specificity For Each Classifier Condition
For Positions 0 – 15 (Message Element) Generate Random Value of Either 0 or 1
For each classifier generate initial values such as strength
Close Reporting Files

Figure 143 Classifier Rule Set Initialisation

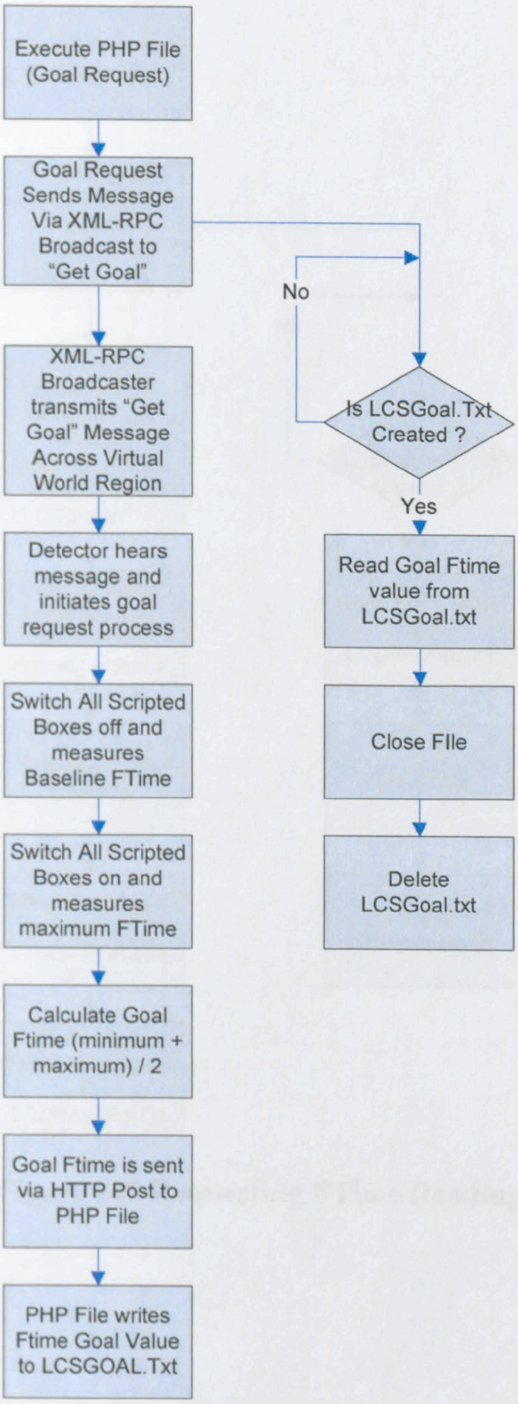

Execute PHP File (Goal Request)
Goal Request Sends Message Via XML-RPC Broadcast to "Get Goal"
XML-RPC Broadcaster transmits "Get Goal" Message Across Virtual World Region
Detector hears message and initiates goal request process
Switch All Scripted Boxes off and measures Baseline FTime
Switch All Scripted Boxes on and measures maximum FTime
Calculate Goal Ftime (minimum + maximum) / 2
Goal Ftime is sent via HTTP Post to PHP File
PHP File writes Ftime Goal Value to LCSGOAL.Txt
No
Is LCSGoal.Txt Created ?
Yes
Read Goal Ftime value from LCSGoal.txt
Close FIle
Delete LCSGoal.txt

Figure 144 Requesting Goal FTime Value

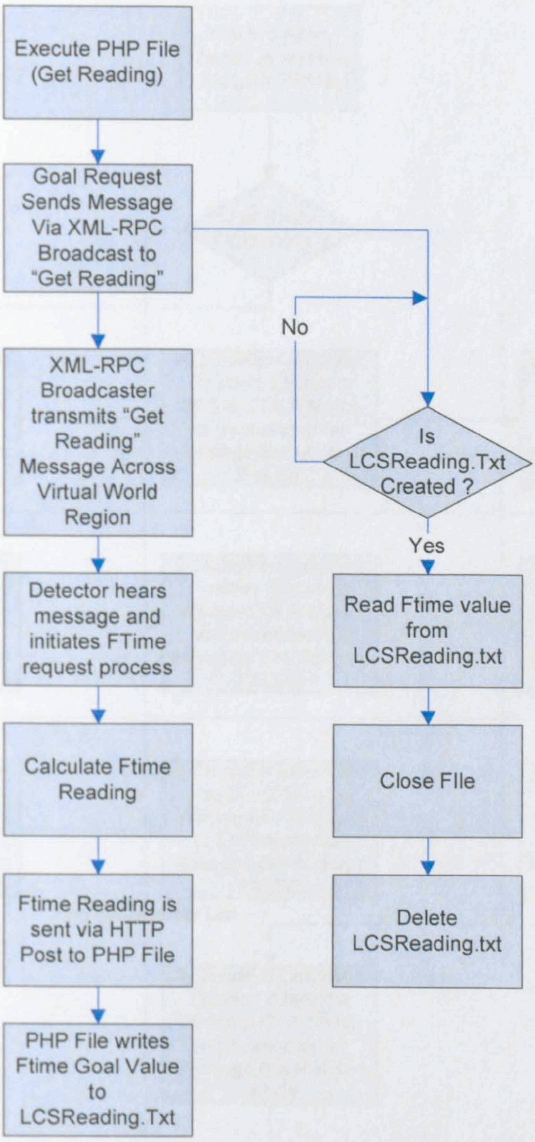
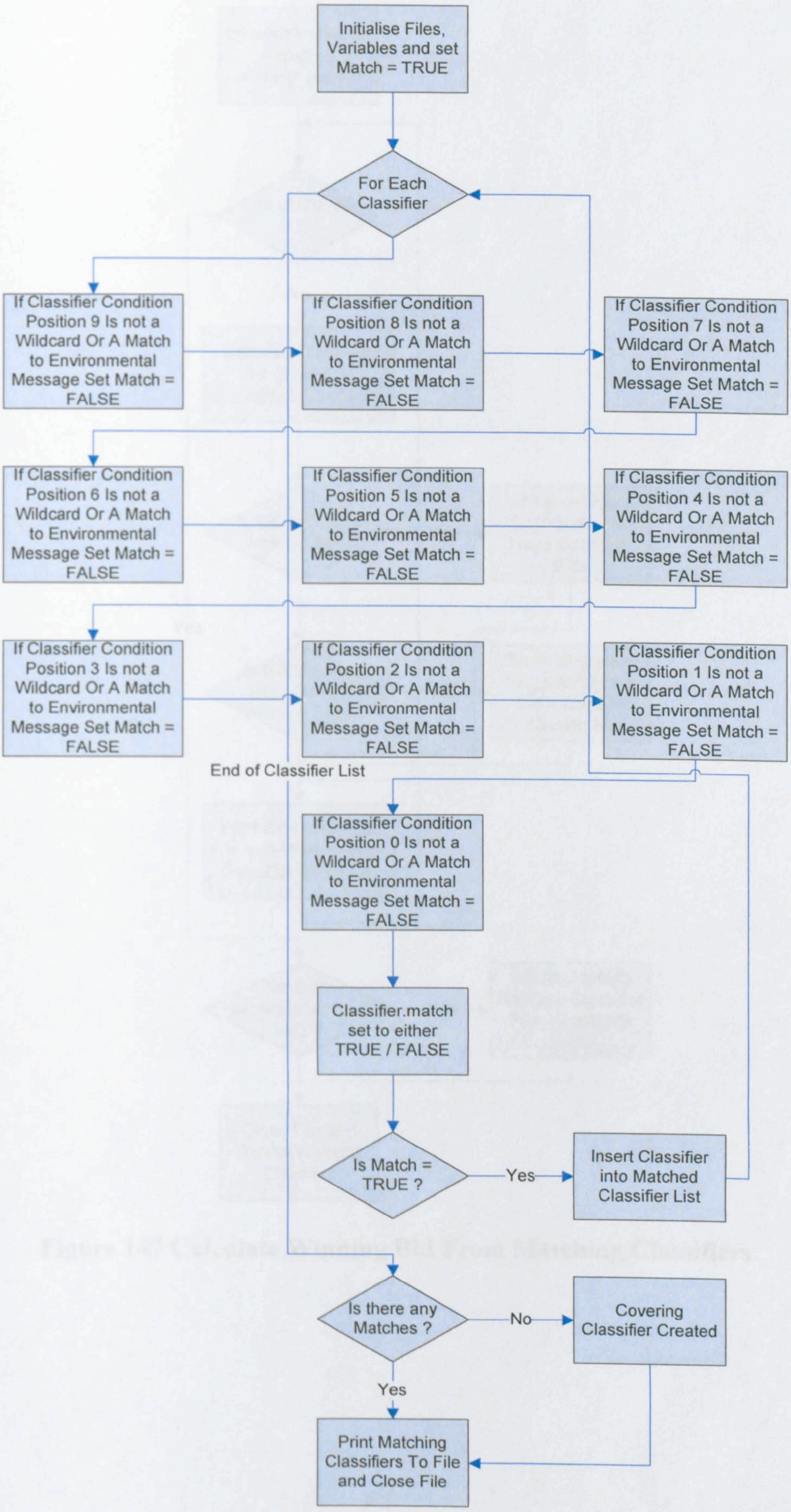

Execute PHP File (Get Reading)
Goal Request Sends Message Via XML-RPC Broadcast to "Get Reading"
XML-RPC Broadcaster transmits "Get Reading" Message Across Virtual World Region
Detector hears message and initiates FTime request process
Calculate Ftime Reading
Ftime Reading is sent via HTTP Post to PHP File
PHP File writes Ftime Goal Value to LCSReading.Txt
No
Is LCSReading.Txt Created ?
Yes
Read Ftime value from LCSReading.txt
Close FIle
Delete LCSReading.txt

Figure 145 Requesting FTime Reading



Initialise Files, Variables and set Match = TRUE
For Each Classifier
If Classifier Condition Position 9 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 8 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 7 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 6 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 5 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 4 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 3 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 2 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
If Classifier Condition Position 1 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
End of Classifier List
If Classifier Condition Position 0 Is not a Wildcard Or A Match to Environmental Message Set Match = FALSE
Classifier.match set to either TRUE / FALSE
Is Match = TRUE ?
Yes
Insert Classifier into Matched Classifier List
Is there any Matches ?
No
Covering Classifier Created
Yes
Print Matching Classifiers To File and Close File

Figure 146 Matching Message Classifiers to Environmental Messages

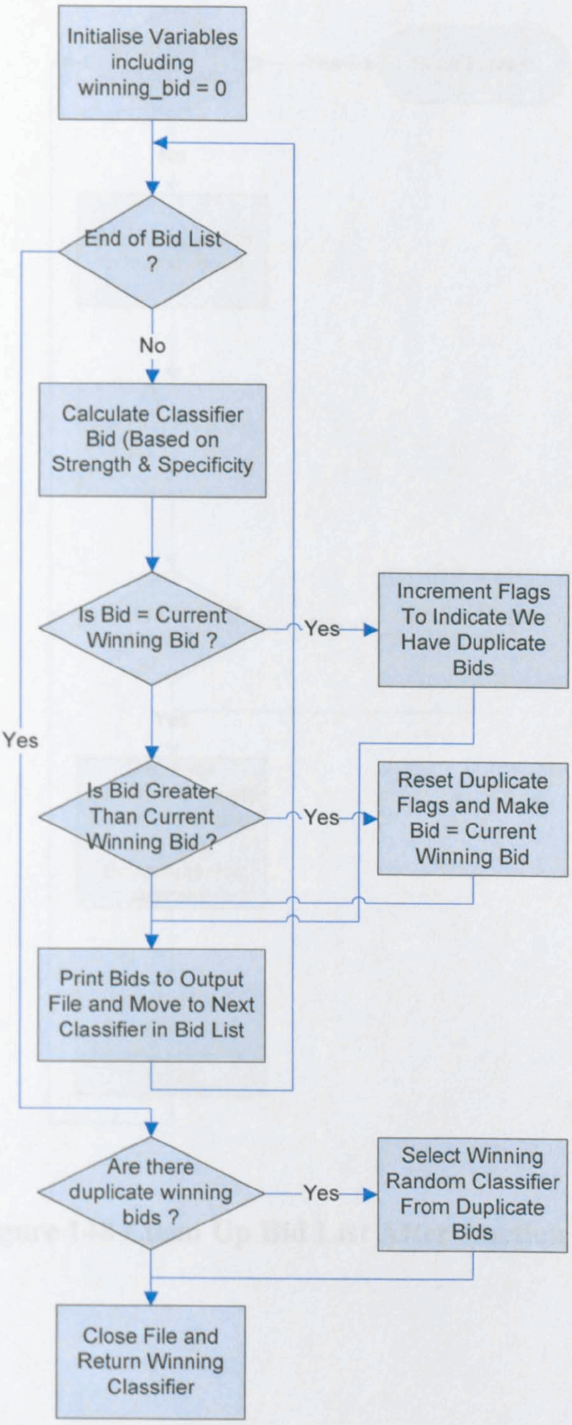

Initialise Variables including winning_bid = 0
End of Bid List ?
No
Calculate Classifier Bid (Based on Strength & Specificity
Is Bid = Current Winning Bid ?
Yes
Increment Flags To Indicate We Have Duplicate Bids
Yes
Is Bid Greater Than Current Winning Bid ?
Yes
Reset Duplicate Flags and Make Bid = Current Winning Bid
Print Bids to Output File and Move to Next Classifier in Bid List
Are there duplicate winning bids ?
Yes
Select Winning Random Classifier From Duplicate Bids
Close File and Return Winning Classifier

Figure 147 Calculate Winning Bid From Matching Classifiers

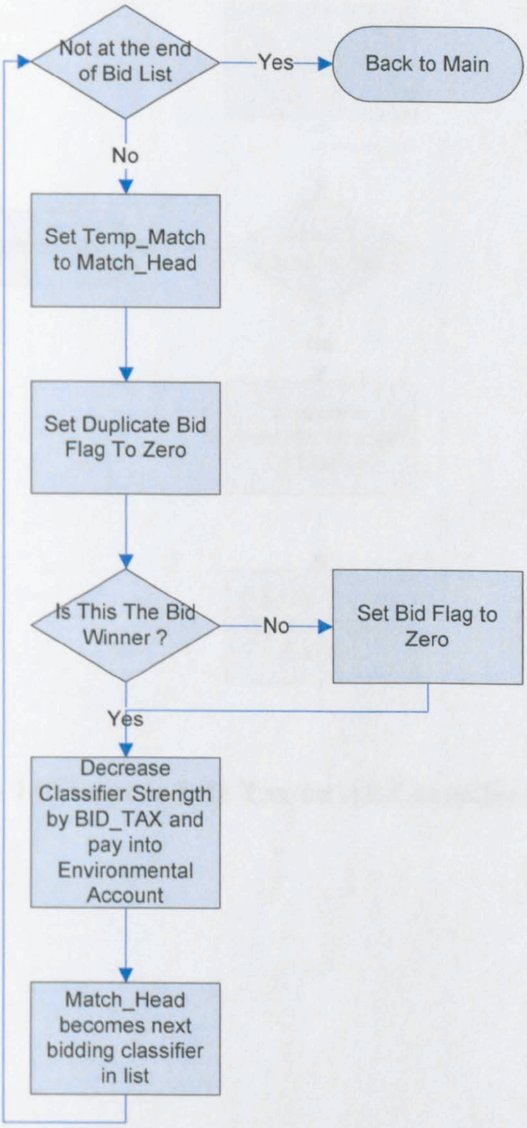

Not at the end of Bid List
Yes
Back to Main
No
Set Temp_Match to Match_Head
Set Duplicate Bid Flag To Zero
Is This The Bid Winner ?
No
Set Bid Flag to Zero
Yes
Decrease Classifier Strength by BID_TAX and pay into Environmental Account
Match_Head becomes next bidding classifier in list

Figure 148 Clean Up Bid List After Auction

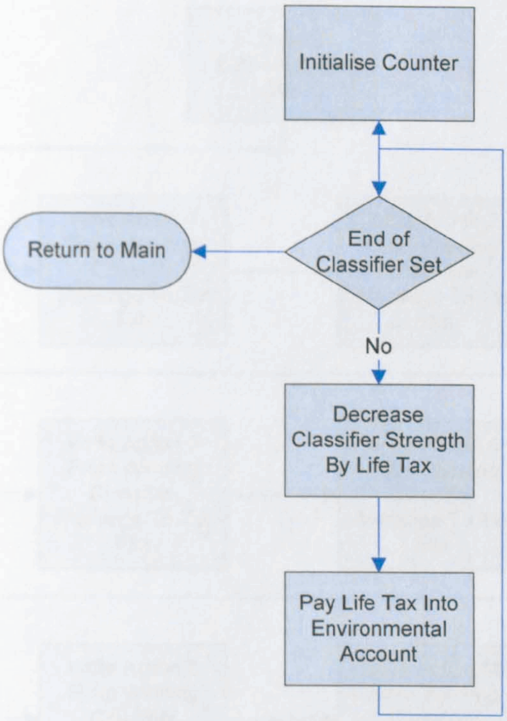

Initialise Counter
Return to Main
End of Classifier Set
No
Decrease Classifier Strength By Life Tax
Pay Life Tax Into Environmental Account

Figure 149 Impose Life Tax on All Classifiers

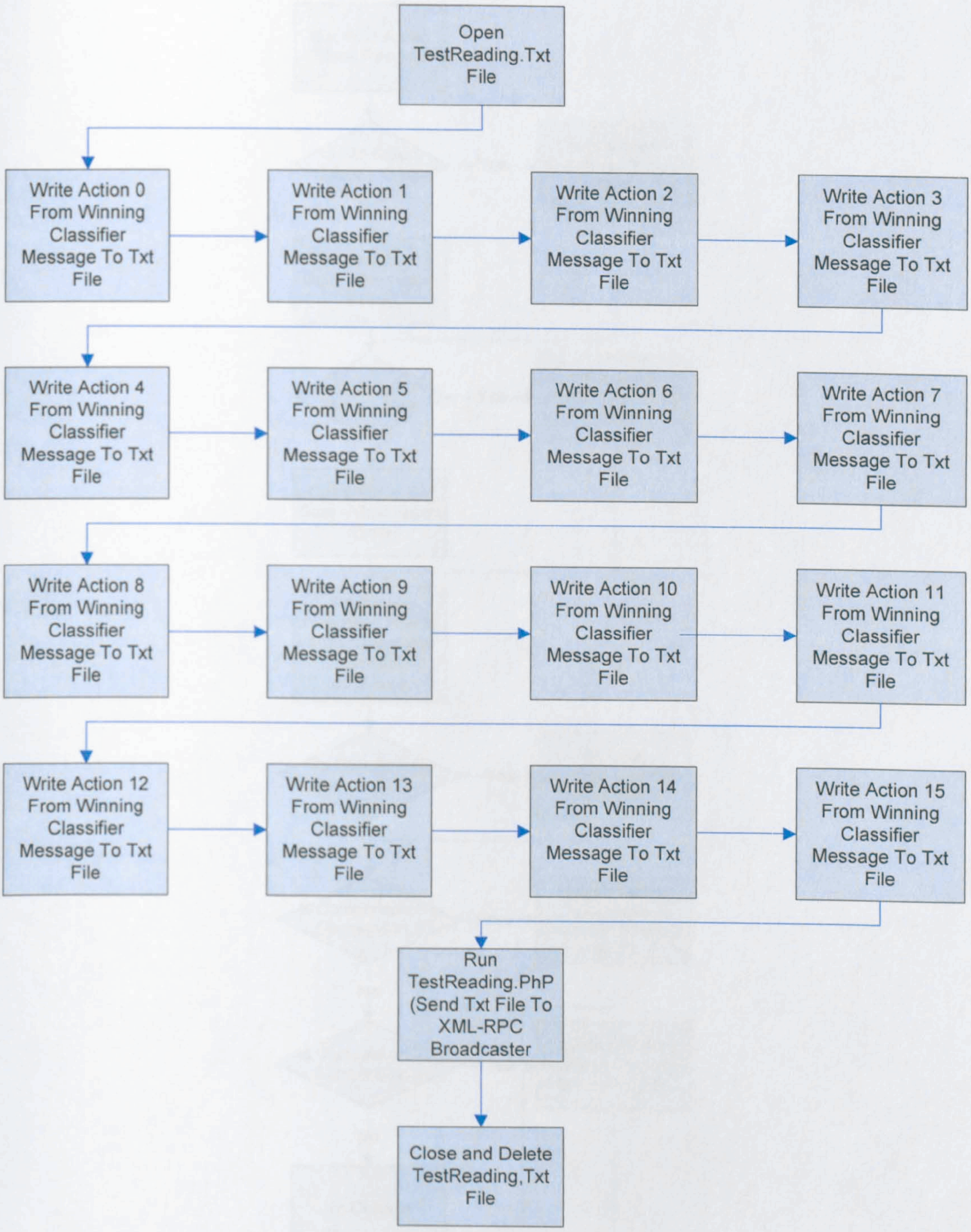

Open TestReading.Txt File
Write Action 0 From Winning Classifier Message To Txt File
Write Action 1 From Winning Classifier Message To Txt File
Write Action 2 From Winning Classifier Message To Txt File
Write Action 3 From Winning Classifier Message To Txt File
Write Action 4 From Winning Classifier Message To Txt File
Write Action 5 From Winning Classifier Message To Txt File
Write Action 6 From Winning Classifier Message To Txt File
Write Action 7 From Winning Classifier Message To Txt File
Write Action 8 From Winning Classifier Message To Txt File
Write Action 9 From Winning Classifier Message To Txt File
Write Action 10 From Winning Classifier Message To Txt File
Write Action 11 From Winning Classifier Message To Txt File
Write Action 12 From Winning Classifier Message To Txt File
Write Action 13 From Winning Classifier Message To Txt File
Write Action 14 From Winning Classifier Message To Txt File
Write Action 15 From Winning Classifier Message To Txt File
Run TestReading.PhP (Send Txt File To XML-RPC Broadcaster
Close and Delete TestReading,Txt File

Figure 150 Sending the Winning Classifier Message to the Virtual World Effectors

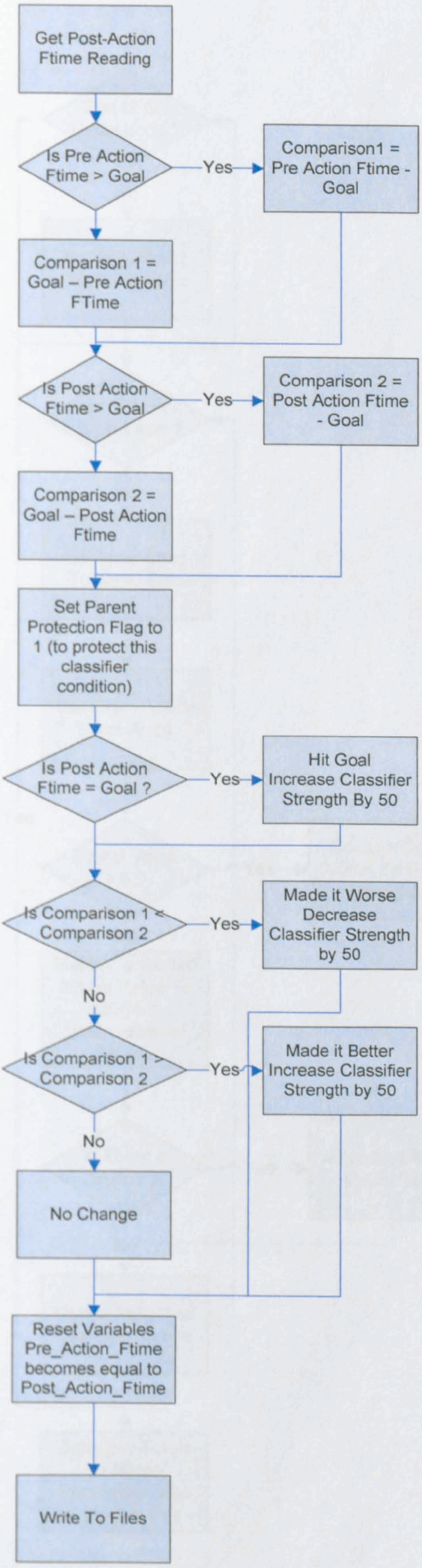

Get Post-Action Ftime Reading
Is Pre Action Ftime > Goal
Yes
Comparison1 = Pre Action Ftime - Goal
Comparison 1 = Goal – Pre Action FTime
Is Post Action Ftime > Goal
Yes
Comparison 2 = Post Action Ftime - Goal
Comparison 2 = Goal – Post Action Ftime
Set Parent Protection Flag to 1 (to protect this classifier condition)
Is Post Action Ftime = Goal ?
Yes
Hit Goal Increase Classifier Strength By 50
Is Comparison 1 < Comparison 2
Yes
Made it Worse Decrease Classifier Strength by 50
No
Is Comparison 1 > Comparison 2
Yes
Made it Better Increase Classifier Strength by 50
No
No Change
Reset Variables Pre_Action_Ftime becomes equal to Post_Action_Ftime
Write To Files

Figure 151 Reward / Punishment of Classifier

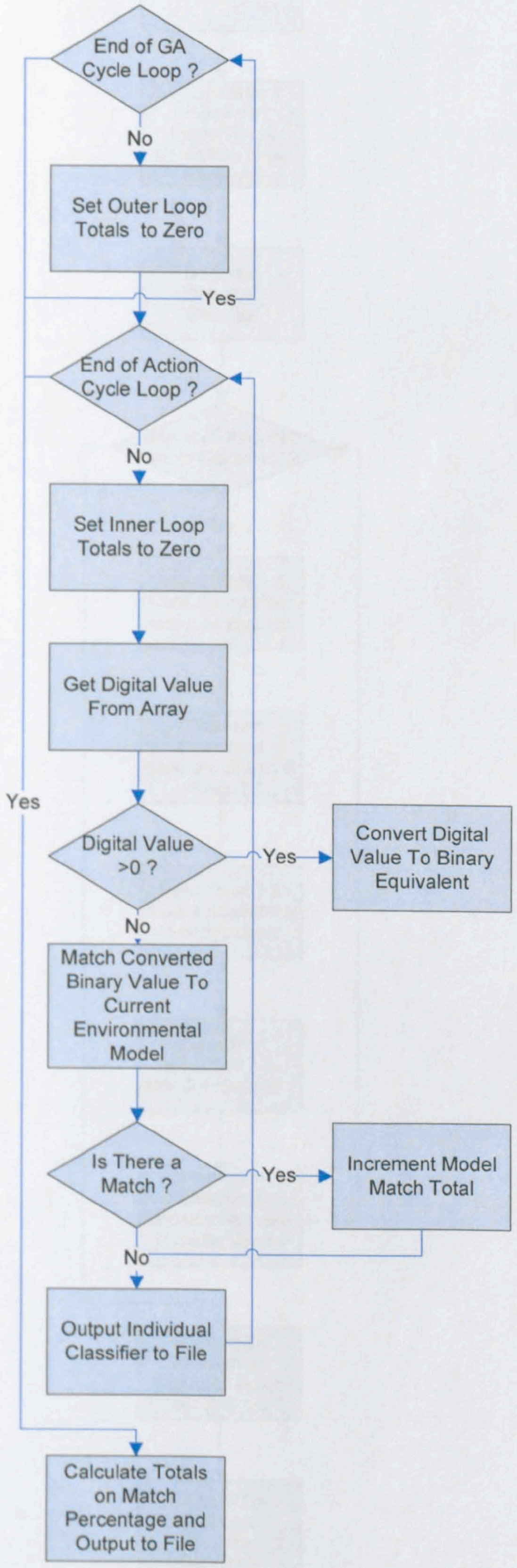

End of GA Cycle Loop ?
No
Set Outer Loop Totals to Zero
Yes
End of Action Cycle Loop ?
No
Set Inner Loop Totals to Zero
Get Digital Value From Array
Yes
Digital Value >0 ?
Yes
Convert Digital Value To Binary Equivalent
No
Match Converted Binary Value To Current Environmental Model
Is There a Match ?
Yes
Increment Model Match Total
No
Output Individual Classifier to File
Calculate Totals on Match Percentage and Output to File

Figure 152 Model Evaluation Report Generation

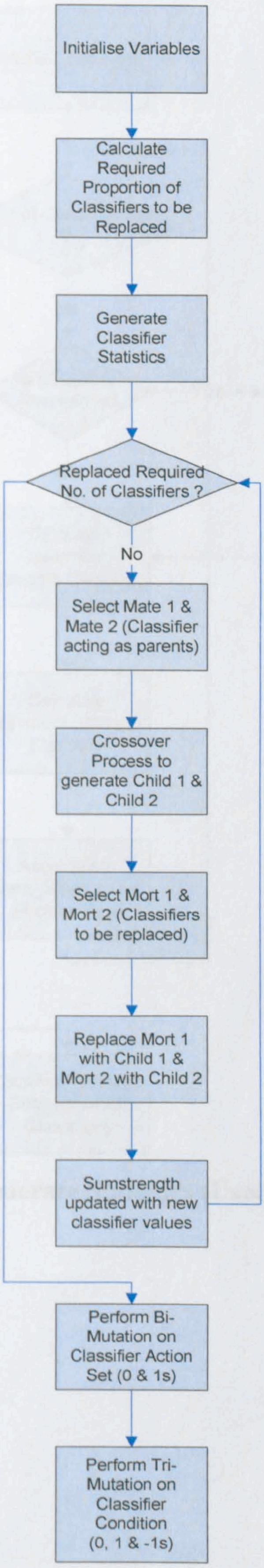
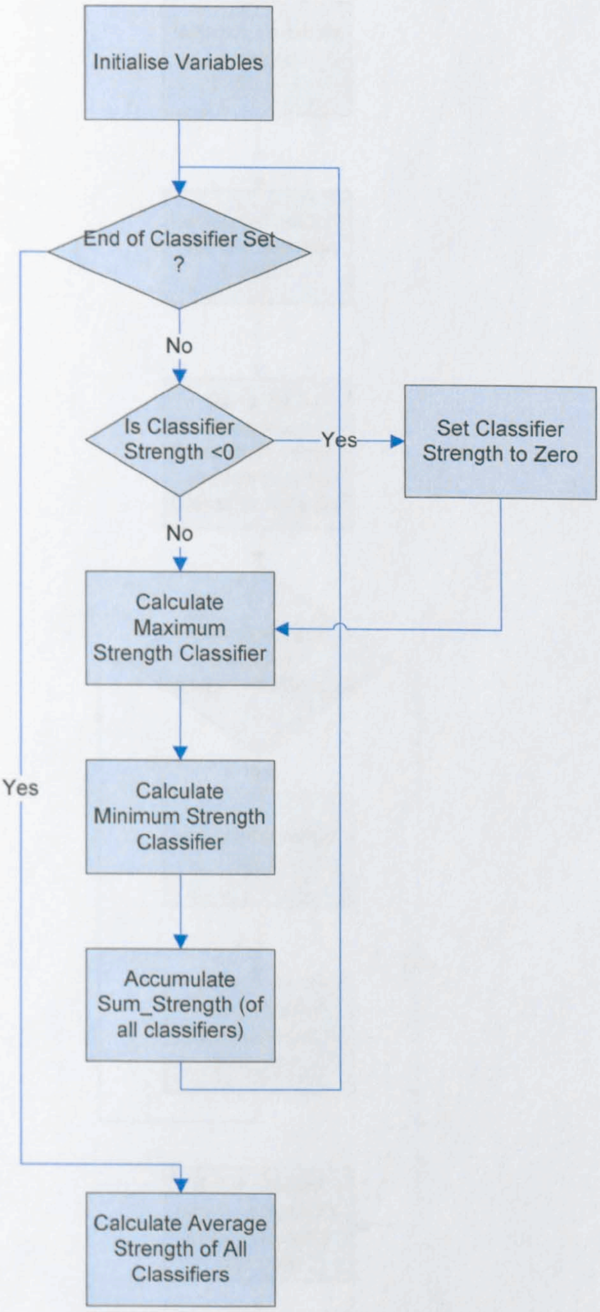

Initialise Variables
Calculate Required Proportion of Classifiers to be Replaced
Generate Classifier Statistics
Replaced Required No. of Classifiers ?
No
Select Mate 1 & Mate 2 (Classifier acting as parents)
Crossover Process to generate Child 1 & Child 2
Select Mort 1 & Mort 2 (Classifiers to be replaced)
Replace Mort 1 with Child 1 & Mort 2 with Child 2
Sumstrength updated with new classifier values
Perform Bi-Mutation on Classifier Action Set (0 & 1s)
Perform Tri-Mutation on Classifier Condition (0, 1 & -1s)

Figure 153 Genetic Algorithm Outline Design



Initialise Variables
End of Classifier Set ?
No
Is Classifier Strength <0
Yes
Set Classifier Strength to Zero
No
Calculate Maximum Strength Classifier
Yes
Calculate Minimum Strength Classifier
Accumulate Sum_Strength (of all classifiers)
Calculate Average Strength of All Classifiers

Figure 154 Generate Statistics (Used in GA Process)

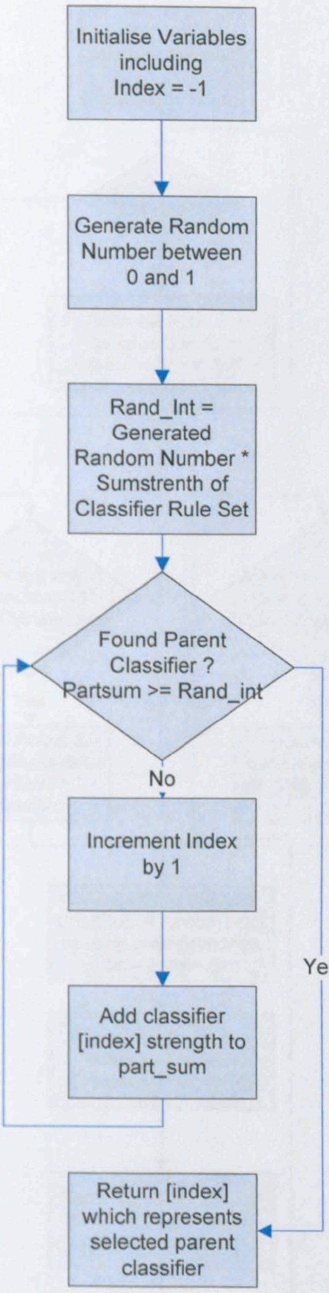

Initialise Variables including Index = -1
Generate Random Number between 0 and 1
Rand_Int = Generated Random Number * Sumstrenth of Classifier Rule Set
Found Parent Classifier ? Partsum >= Rand_int
No
Increment Index by 1
Add classifier [index] strength to part_sum
Yes
Return [index] which represents selected parent classifier

Figure 155 Select Parent Classifier (for breeding purposes)

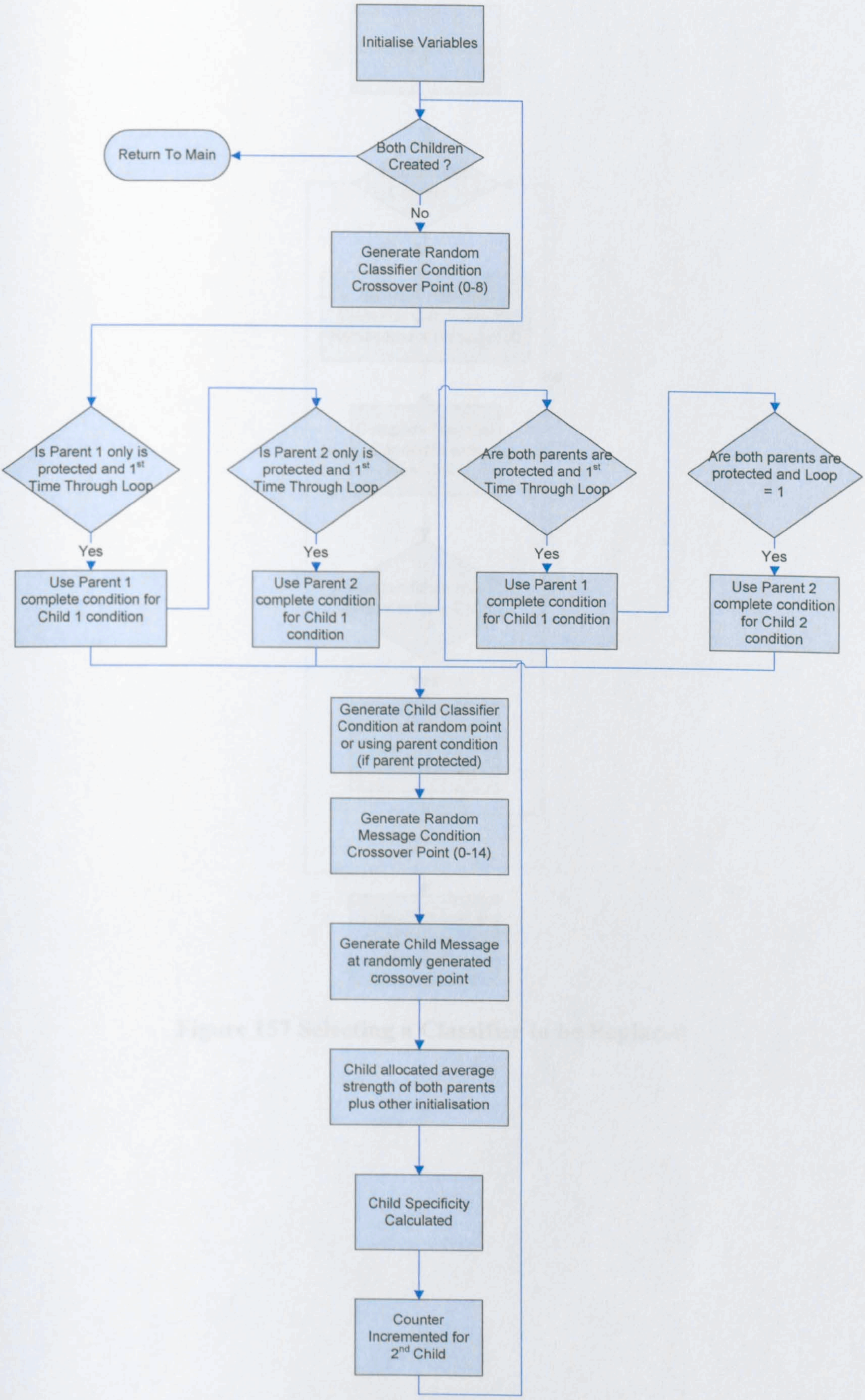

Initialise Variables
Both Children Created ?
Return To Main
No
Generate Random Classifier Condition Crossover Point (0-8)
Is Parent 1 only is protected and 1st Time Through Loop
Is Parent 2 only is protected and 1st Time Through Loop
Are both parents are protected and 1st Time Through Loop
Are both parents are protected and Loop = 1
Yes
Yes
Yes
Yes
Use Parent 1 complete condition for Child 1 condition
Use Parent 2 complete condition for Child 1 condition
Use Parent 1 complete condition for Child 1 condition
Use Parent 2 complete condition for Child 2 condition
Generate Child Classifier Condition at random point or using parent condition (if parent protected)
Generate Random Message Condition Crossover Point (0-14)
Generate Child Message at randomly generated crossover point
Child allocated average strength of both parents plus other initialisation
Child Specificity Calculated
Counter Incremented for 2nd Child

Figure 156 Generate Parent Crossover Point to Breed Two Child Classifiers

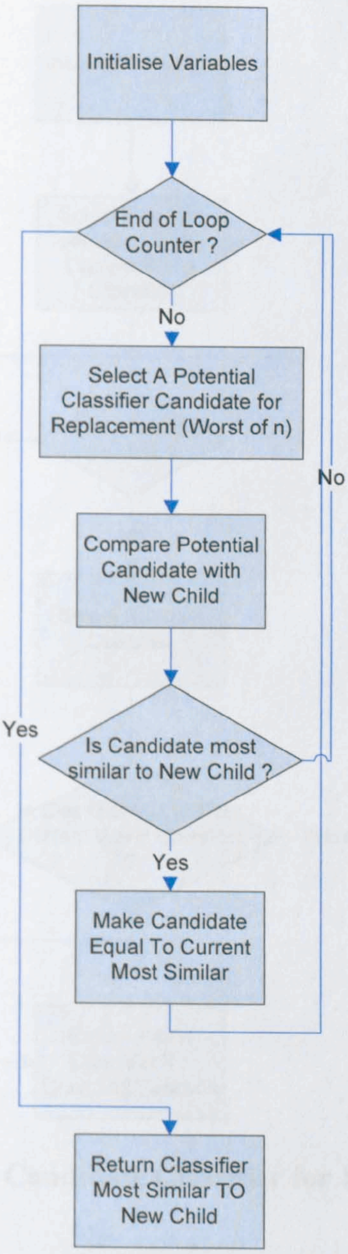

Initialise Variables
End of Loop Counter ?
No
Select A Potential Classifier Candidate for Replacement (Worst of n)
No
Compare Potential Candidate with New Child
Yes
Is Candidate most similar to New Child ?
Yes
Make Candidate Equal To Current Most Similar
Return Classifier Most Similar TO New Child

Figure 157 Selecting a Classifier to be Replaced

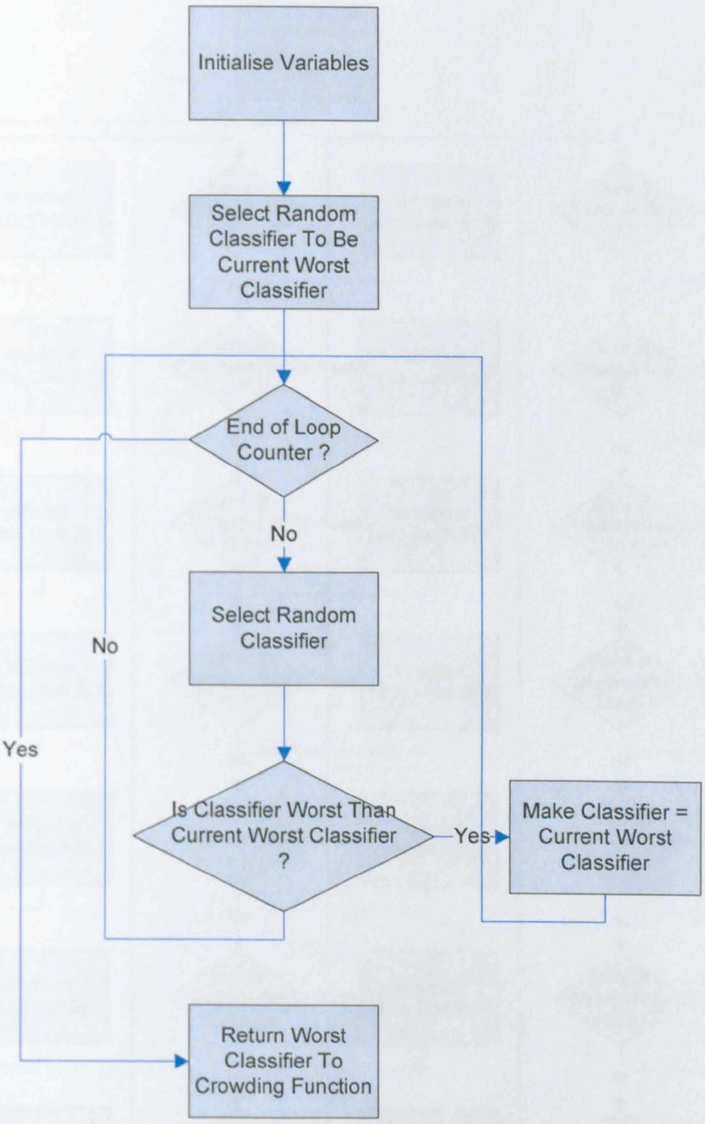

Initialise Variables
Select Random Classifier To Be Current Worst Classifier
End of Loop Counter ?
No
Select Random Classifier
No
Yes
Is Classifier Worst Than Current Worst Classifier ?
Yes
Make Classifier = Current Worst Classifier
Return Worst Classifier To Crowding Function

Figure 158 Finding a Candidate Classifier for Potential Replacement

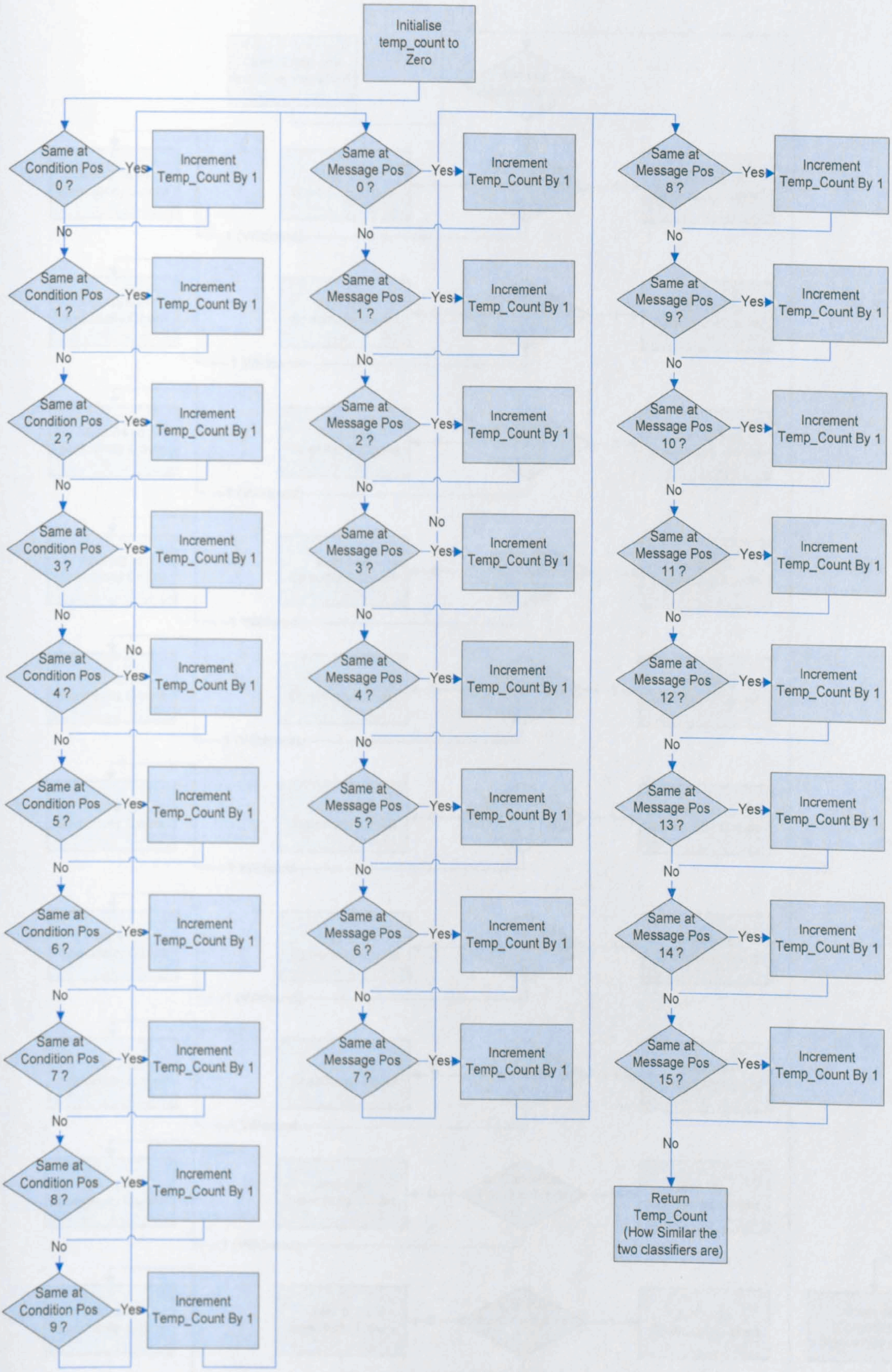


Figure 159 Comparing Classifiers to Calculate Their Similarity

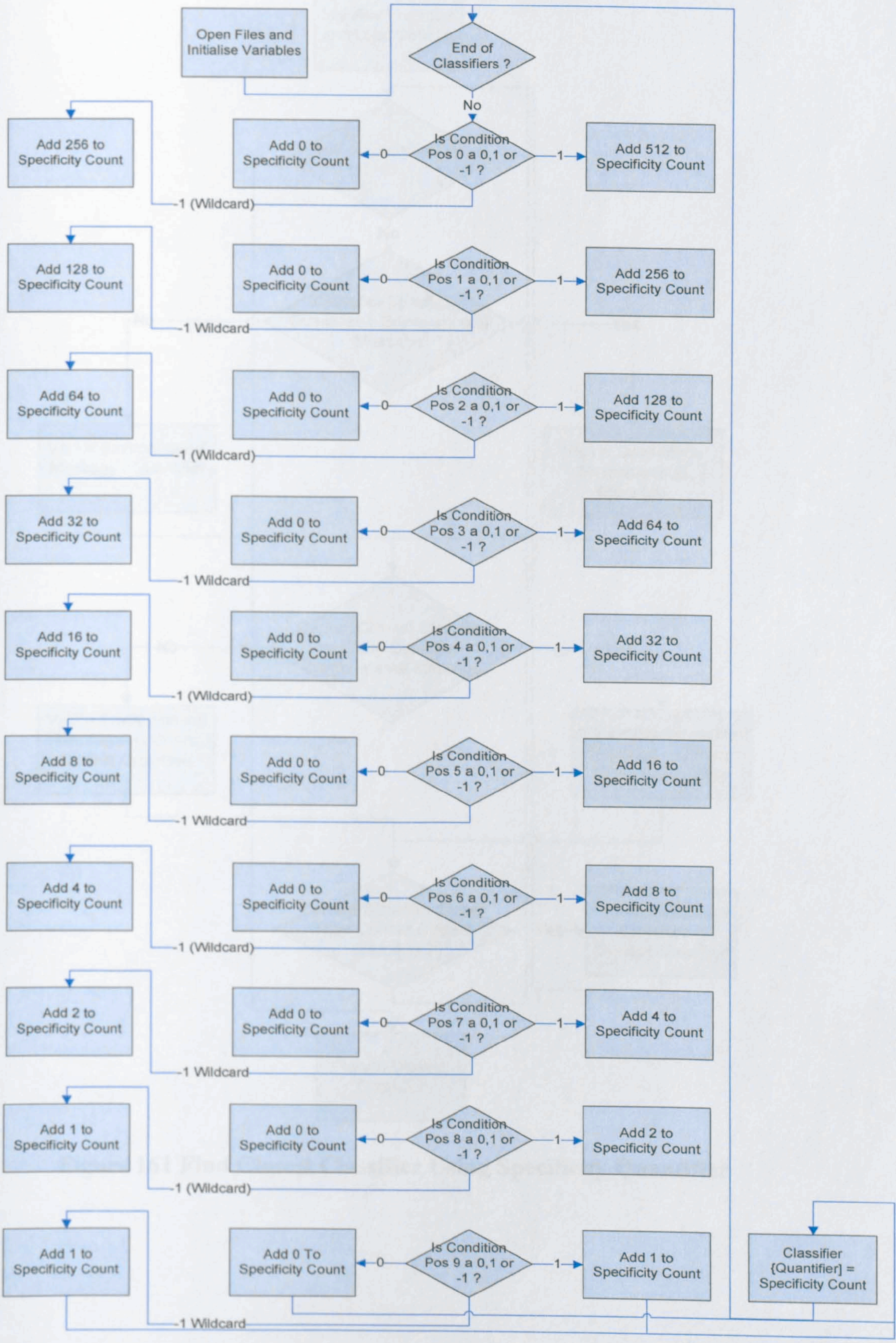

Open Files and Initialise Variables
End of Classifiers ?
No
Is Condition Pos 0 a 0,1 or -1 ?
0
1
-1 (Wildcard)
Add 0 to Specificity Count
Add 512 to Specificity Count
Add 256 to Specificity Count
Is Condition Pos 1 a 0,1 or -1 ?
-1 Wildcard
Add 0 to Specificity Count
Add 256 to Specificity Count
Add 128 to Specificity Count
Is Condition Pos 2 a 0,1 or -1 ?
-1 (Wildcard)
Add 0 to Specificity Count
Add 128 to Specificity Count
Add 64 to Specificity Count
Is Condition Pos 3 a 0,1 or -1 ?
-1 Wildcard
Add 0 to Specificity Count
Add 64 to Specificity Count
Add 32 to Specificity Count
Is Condition Pos 4 a 0,1 or -1 ?
-1 (Wildcard)
Add 0 to Specificity Count
Add 32 to Specificity Count
Add 16 to Specificity Count
Is Condition Pos 5 a 0,1 or -1 ?
-1 Wildcard
Add 0 to Specificity Count
Add 16 to Specificity Count
Add 8 to Specificity Count
Is Condition Pos 6 a 0,1 or -1 ?
-1 (Wildcard)
Add 0 to Specificity Count
Add 8 to Specificity Count
Add 4 to Specificity Count
Is Condition Pos 7 a 0,1 or -1 ?
-1 Wildcard
Add 0 to Specificity Count
Add 4 to Specificity Count
Add 2 to Specificity Count
Is Condition Pos 8 a 0,1 or -1 ?
-1 (Wildcard)
Add 0 to Specificity Count
Add 2 to Specificity Count
Add 1 to Specificity Count
Is Condition Pos 9 a 0,1 or -1 ?
-1 Wildcard
Add 0 To Specificity Count
Add 1 to Specificity Count
Add 1 to Specificity Count
Classifier {Quantifier} = Specificity Count

Figure 160 Specificity Quantified

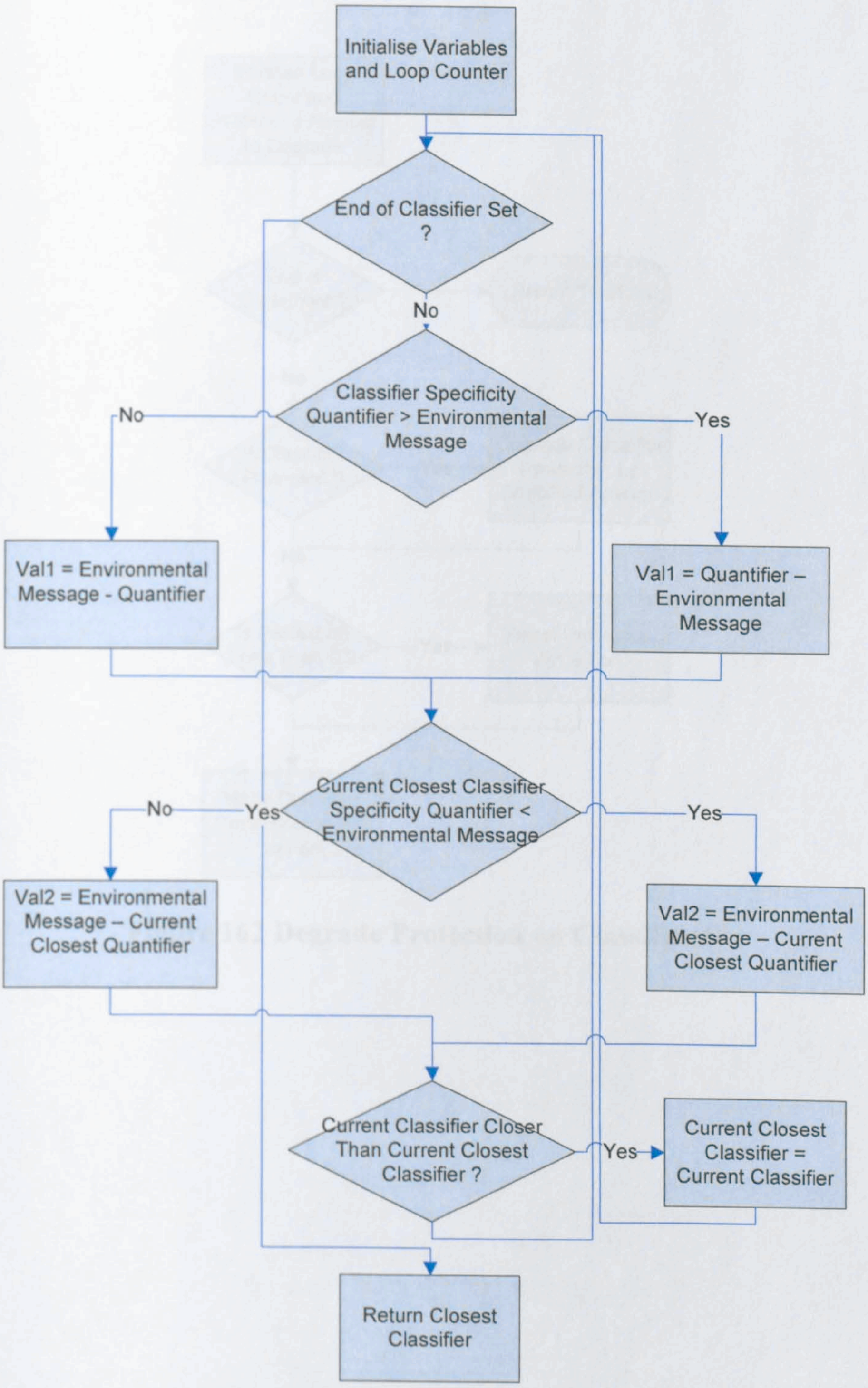

Initialise Variables and Loop Counter
End of Classifier Set ?
No
Classifier Specificity Quantifier > Environmental Message
No
Yes
Val1 = Environmental Message - Quantifier
Val1 = Quantifier – Environmental Message
Current Closest Classifier Specificity Quantifier < Environmental Message
No
Yes
Yes
Val2 = Environmental Message – Current Closest Quantifier
Val2 = Environmental Message – Current Closest Quantifier
Current Classifier Closer Than Current Closest Classifier ?
Yes
Current Closest Classifier = Current Classifier
Return Closest Classifier

Figure 161 Find Closest Classifier Using Specificity Quantifier

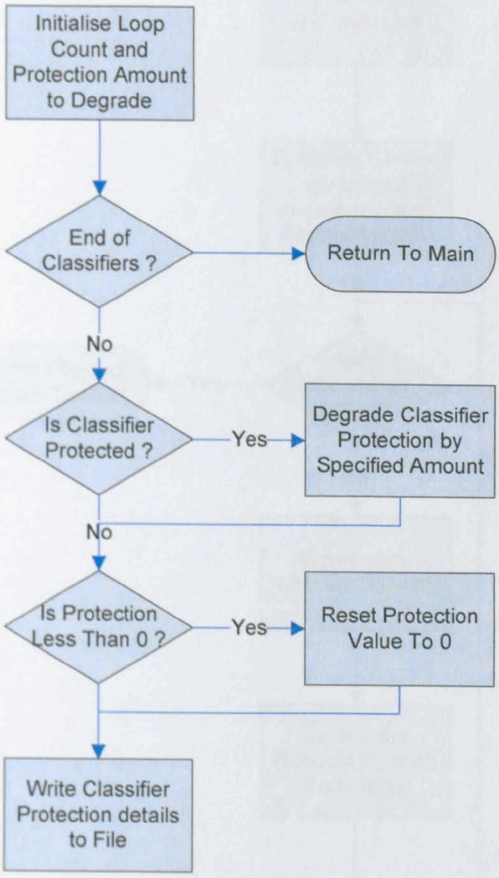

Initialise Loop Count and Protection Amount to Degrade
End of Classifiers ?
Return To Main
No
Is Classifier Protected ?
Yes
Degrade Classifier Protection by Specified Amount
No
Is Protection Less Than 0 ?
Yes
Reset Protection Value To 0
Write Classifier Protection details to File

Figure 162 Degrade Protection on Classifier Set

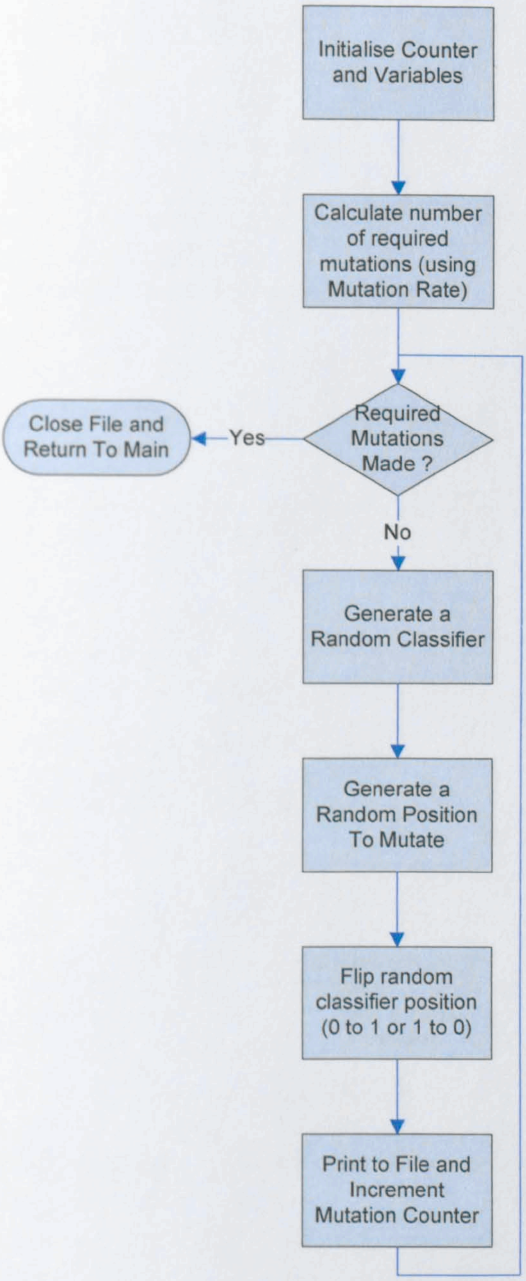

Initialise Counter and Variables
Calculate number of required mutations (using Mutation Rate)
Required Mutations Made ?
Yes
Close File and Return To Main
No
Generate a Random Classifier
Generate a Random Position To Mutate
Flip random classifier position (0 to 1 or 1 to 0)
Print to File and Increment Mutation Counter

Figure 163 Bi-Mutation Process

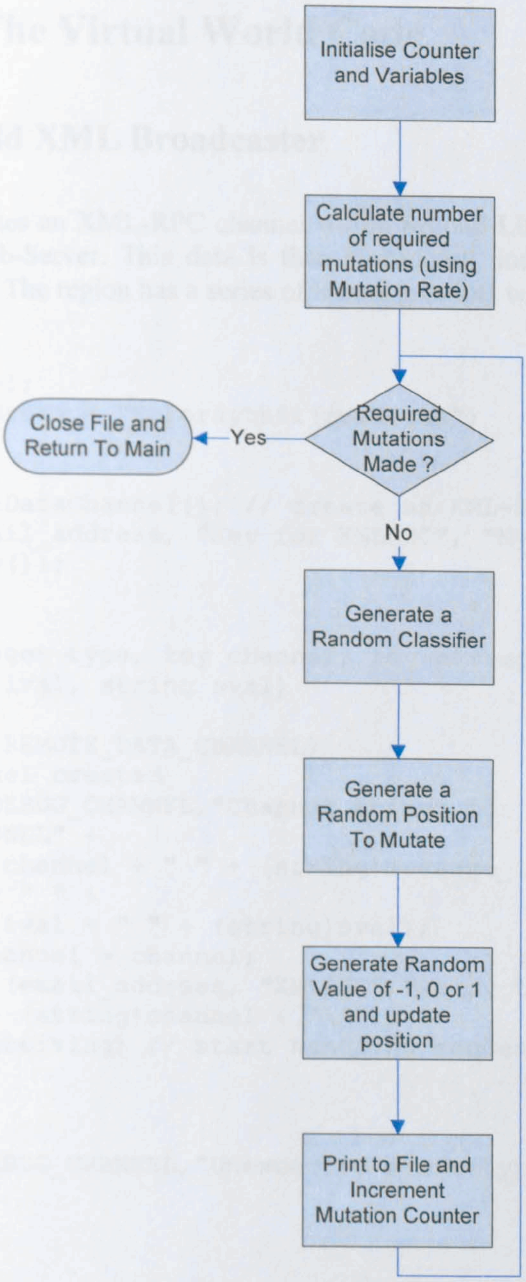

Initialise Counter and Variables
Calculate number of required mutations (using Mutation Rate)
Required Mutations Made ?
Yes
Close File and Return To Main
No
Generate a Random Classifier
Generate a Random Position To Mutate
Generate Random Value of -1, 0 or 1 and update position
Print to File and Increment Mutation Counter

Figure 164 Tri-Mutation Process

Appendix B The Virtual World Code

B1 The In-World XML Broadcaster

This LSL script creates an XML-RPC channel within Second Life which allows data to be received from a Web-Server. This data is then “broadcast” into the virtual environment region on Channel 5. The region has a series of listening scripts which then interpret the data into commands.

```
key remoteChannel;
string email_address = "H.Forsyth@ljmu.ac.uk";
init()
    {
    llOpenRemoteDataChannel(); // create an XML-RPC channel
    llEmail (email_address, "Key for XMLRPC", "My key is " +
(string)llGetKey());
    }

remote_data(integer type, key channel, key message_id, string
sender, integer ival, string sval)
    {
    if (type == REMOTE_DATA_CHANNEL)
      { // channel created
        llSay (DEBUG_CHANNEL,"Channel opened for
REMOTE_DATA_CHANNEL" +
        (string)channel + " " + (string)message_id + " " +
(string)sender + " " +
        (string)ival + " " + (string)sval);
        remoteChannel = channel;
        llEmail (email_address, "XMLRPC","Ready to receive requests
on channel \"" + (string)channel + "\"");
        state receiving; // start handling requests
        }
      else
      {
        llSay(DEBUG_CHANNEL,"Unexpected event type");
        }
    }
}


state receiving
    {
    state_entry()
      {
            llOwnerSay("Ready to receive information from outside
SL");
            }

    state_exit()
      {
            llOwnerSay("No longer receiving information from outside
SL.");
            llCloseRemoteDataChannel(remoteChannel);
      }

    on_rez(integer param)
      {
            llResetScript();
      }

    remote_data(integer type, key channel, key message_id, string
sender, integer ival, string sval)
      {
            if (type == REMOTE_DATA_REQUEST)
            { // handle requests sent to us
            llSay(DEBUG_CHANNEL,"Request received for
            REMOTE_DATA_REQUEST " + (string)channel + " " +
            (string)message_id + " " + (string)sender + " " +
            (string)ival + " " + (string)sval);
                  llRemoteDataReply(channel,NULL_KEY,"I got
it",2008);
                  llOwnerSay("I just received data in "+
llGetRegionName() +
                  " at position " + (string)llGetPos() + "\n" +
                  "The string was " +  sval + "\nThe number was " +
(string)ival + ".");
                  llRegionSay(5, sval);
                  }
      }
    }
```

B2 Converting Message From XML Broadcaster Into Commands to the Scripted Objects

This LSL script "listens" on communication channel 5 for any messages received from the XML-RPC broadcaster and translate them into commands. In this particular example a text string, consisting of binary characters, would be converted into a set of commands for the region depending on the message. Therefore 01101 would be converted to Stop 1, Start 2, Start 3, Stop 4, Start 5. This was used to switch scripted objects in the region off or on.

```
{
    state_entry()
        {
        // listen on channel five for any message received from the
XML-Broadcaster.
        llListen(5,"",NULL_KEY,"");
        }

    listen(integer channel, string name, key id, string message)
    {
        // Convert received message into a set of either start or
stop commands.
        if ((llGetSubString(message, 0, 0))=="0")
            {
            llRegionSay (5, "Stop 1");
            }
        else if ((llGetSubString(message, 0, 0))=="1")
            {
            llRegionSay (5, "Start 1");
            }

        if ((llGetSubString(message, 1, 1))=="0")
            {
            llRegionSay (5, "Stop 2");
            }
        else if ((llGetSubString(message, 1, 1))=="1")
            {
            llRegionSay (5, "Start 2");
            }
        if ((llGetSubString(message, 2, 2))=="0")
            {
            llRegionSay (5, "Stop 3");
            }
        else if ((llGetSubString(message, 2, 2))=="1")
            {
            llRegionSay (5, "Start 3");
            }

        if ((llGetSubString(message, 3, 3))=="0")
            {
            llRegionSay (5, "Stop 4");
            }
```

B3 Scripted Object Listen and Restart Script

This LSL script creates an XML-RPC channel within Second Life which allows data to be received from a Web-Server. This data is then "broadcast" into the virtual environment region on Channel 5. The region has a series of listening scripts which then interpret the data into commands.

```
listen(integer channel, string name, key id, string message)
    {
    // check if the message corresponds to a predefined string.
    // llToLower converts the message to lowercase.
    // This way, "START 1", "Start 1" or "StArT 1" will all work
    the same way.
    if (llToLower(message) == "start 1")
        {
        integer x;

        for(x=1;x<llGetInventoryNumber(INVENTORY_SCRIPT);x++)
            {
            if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
            //If the script number x is not this script

            llSetScriptState(giv(INVENTORY_SCRIPT,x),TRUE);
            //Start it
            }
        llSetColor(<0,1,0>, ALL_SIDES); // set object colour
        green.
        }

    if (llToLower(message) == "start all")
        {
        integer x;

        for(x=1;x<llGetInventoryNumber(INVENTORY_SCRIPT);x++)
            {
            if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
            //If the script number x is not this script

            llSetScriptState(giv(INVENTORY_SCRIPT,x),TRUE);
            //Start it
            }
        llSetColor(<0,1,0>, ALL_SIDES); // set object colour
        green.
        }

     if (llToLower(message) == "stop all")
        {
        integer x;

        for(x=1;x<llGetInventoryNumber(INVENTORY_SCRIPT);x++)
            {
            if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
            //If the script number x is not this script
            llSetScriptState(giv(INVENTORY_SCRIPT,x),FALSE);
            //Stop it
            }
        llSetColor(<1,0,0>, ALL_SIDES); // set object colour
        red.
        }
```

```
  if (llToLower(message) == "stop 1")
      {
      integer x;

      for(x=1;x<llGetInventoryNumber(INVENTORY_SCRIPT);x++)
             {
             if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
             //If the script number x is not this script
             llSetScripttate(giv(INVENTORY_SCRIPT,x),FALSE);
             //Stop it
             }
      llSetColor(<1,0,0>, ALL_SIDES); // set object colour
      red.}
      }
}
```

B4 The Random Experiment

If the current performance level (FTime) is not meeting the required goal then boxes (representing functionality) are randomly switched on / off to try and match the goal performance level. If the goal is met no further action is taken.

```
// Runs the random test iteration number of times
for (counter=1;counter<iterations+1; counter++)

// If current ftime reading is greater than the goal generate
another random action set
if (ftime>goal)
     {
     // For each of the scripted objects in turn generate a
     random action (on / off)
     for (counter=1;counter<17; counter++)
          {
          integer random = (integer)llFrand(2.0);
          //will return an integer 0 to 1, all with equal
          probabilities(more or less).
          llOwnerSay ((string)random);
          // Start or stop scripted object depending result of
          random calculation
          if (random==1)
               {
               llRegionSay (5, "Start " + (string)counter);
               boxstarted=boxstarted+1;
               }
          if (random==0)
               {
               llRegionSay (5, "Stop " + (string)counter);
               boxstopped=boxstopped+1;
               }
          }
     }
else if (ftime < goal)
      {
      // llRegionSay (5, "Start All");
      for (counter=1;counter<17; counter++)
          {
          integer random = (integer)llFrand(2.0);
          //will return an integer 0 to 1, all with
          equal probabilities (more or less).
          llOwnerSay ((string)random);
          if (random==1)
               {
               llRegionSay (5, "Start " + (string)counter);
               }
          if (random==0)
               {
               llRegionSay (5, "Stop " + (string)counter);
               }
          }
       }
```

B5 The Static Model

The Static Model LSL script used observational results to calculate average goal and individual script contributions. Using this data the script attempts to maintain the FTime readings at the goal (4.591) by switching objects on / off dependent on the requirement to increase / decrease FTime.

```
float goal=4.591; // Actual readings based on 10,000 plus
observations
float box1=0.499;
float box2=0.452;
float box3=0.390;
float box4=0.321;
float box5=0.290;
float box6=0.262;
float box7=0.238;
float box8=0.147;
float box9=0.140;
float box10=0.137;
float box11=0.123;
float box12=0.098;
float box13=0.039;
float box14=0.038;
float box15=0.029;
float box16=0.041;
float boundary1=0.4;
float boundary2=0.8;
float boundary3=1.2;
float boundary4=1.6;
float boundary5=2.0;

// Run the experiment iterations number of times
for (counter=1;counter<(iterations+1); counter++)
      {
      // Get current ftime reading
      float ftime = ComputeLag(MAX_LOOP*10);
      ftime = ftime/10.0;

      // If hitting goal take no action
      if (ftime == goal)
            {
            llOwnerSay("Hitting Baseline - No Action Required");
            }

      // If ftime is below goal start adding scripts based on their
      predicted contribution

      else if (ftime < goal)
            {
            llOwnerSay("Below Goal - Adding Scripts");
            // Calculate how much below the goal the ftime currently
            is so we can try to Turn on the "right" value objects
            float difference = goal - ftime;
            llOwnerSay ("Diff = "+(string)difference);
            // Switch on Scripted Objects if it wouldn't put us over
            the goal and it's currently not already running

            if (box1 <= difference && !running1)
                {
                llRegionSay (5, "Start 1");
```

```
    difference = (difference - box1);
    running1=TRUE;
    }
 if (box2 <= difference && !running2)
    {
    llRegionSay (5, "Start 2");
    difference = (difference - box2);
    running2=TRUE;
    }
if (box3 <= difference && !running3)
    {
    llRegionSay (5, "Start 3");
    difference = (difference - box3);
    running3=TRUE;
    }
if (box4 <= difference && !running4)
    {
    llRegionSay (5, "Start 4");
    difference = (difference - box4);
    running4=TRUE;
    }
 if (box5 <= difference && !running5)
    {
    llRegionSay (5, "Start 5");
    difference = (difference - box5);
    running5=TRUE;
    }
if (box6 <= difference && !running6)
    {
    llRegionSay (5, "Start 6");
    difference = (difference - box6);
    running6=TRUE;
    }
if (box7 <= difference && !running7)
    {
    llRegionSay (5, "Start 7");
    difference = (difference - box7);
    running7=TRUE;
    }
if (box8 <= difference && !running8)
    {
    llRegionSay (5, "Start 8");
    difference = (difference - box8);
    running8=TRUE;
    }
 if (box9 <= difference && !running9)
    {
    llRegionSay (5, "Start 9");
    difference = (difference - box9);
    running9=TRUE;
    }
if (box10 <= difference && !running10)
    {
    llRegionSay (5, "Start 10");
    difference = (difference - box10);
    running10=TRUE;
    }
if (box11 <= difference && !running11)
    {
    llRegionSay (5, "Start 11");
    difference = (difference - box11);
    running11=TRUE;
    }
if (box12 <= difference && !running12)
    {
    llRegionSay (5, "Start 12");
    difference = (difference - box12);
    running12=TRUE;
    }
 if (box13 <= difference && !running13)
    {
    llRegionSay (5, "Start 13");
    difference = (difference - box13);
    running13=TRUE;
    }
if (box14 <= difference && !running14)
    {
    llRegionSay (5, "Start 14");
    difference = (difference - box14);
    running14=TRUE;
    }
if (box15 <= difference && !running15)
    {
    llRegionSay (5, "Start 15");
    difference = (difference - box15);
    running15=TRUE;
    }
if (box16 <= difference && !running16)
    {
    llRegionSay (5, "Start 16");
    difference = (difference - box16);
    running16=TRUE;
    }
}

// If the ftime reading is above the goal then start
switching objects off
else if (ftime > goal)
    {
    llOwnerSay("Above Baseline - Stopping Scripts");
    // Calculate how much we are above the goal in order
    to calculate which objects to turn off
float difference = ftime - goal;
llOwnerSay ("Diff = "+(string)difference);
// If box can be switched off without going under the
goal and it's currently running then switch it off.

if (box1 <= difference && running1 && box1>0)
    {
    llRegionSay (5, "Stop 1");
    difference = (difference - box1);
    running1=FALSE;
    }
if (box2 <= difference && running2 && box2>0)
    {
    llRegionSay (5, "Stop 2");
    difference = (difference - box2);
    running2=FALSE;
    }
if (box3 <= difference && running3 && box3>0)
    {
    llRegionSay (5, "Stop 3");
    difference = (difference - box3);
    running3=FALSE;
    }
if (box4 <= difference && running4 && box4>0)
    {
    llRegionSay (5, "Stop 4");
    difference = (difference - box4);
    running4=FALSE;
    }
 if (box5 <= difference && running5 && box5>0)
    {
    llRegionSay (5, "Stop 5");
    difference = (difference - box5);
    running5=FALSE;
    }
if (box6 <= difference && running6 && box6>0)
    {
    llRegionSay (5, "Stop 6");
    difference = (difference - box6);
    running6=FALSE;
    }
if (box7 <= difference && running7 && box7>0)
    {
    llRegionSay (5, "Stop 7");
    difference = (difference - box7);
    running7=FALSE;
    }
if (box8 <= difference && running8 && box8>0)
    {
    llRegionSay (5, "Stop 8");
    difference = (difference - box8);
    running8=FALSE;
    }
 if (box9 <= difference && running9 && box9>0)
    {
    llRegionSay (5, "Stop 9");
    difference = (difference - box9);
    running9=FALSE;
    }
if (box10 <= difference && running10 && box10>0)
    {
    llRegionSay (5, "Stop 10");
    difference = (difference - box10);
    running10=FALSE;
    }
if (box11 <= difference && running11 && box11>0)
    {
    llRegionSay (5, "Stop 11");
    difference = (difference - box11);
    running11=FALSE;
    }
if (box12 <= difference && running12 && box12>0)
    {
    llRegionSay (5, "Stop 12");
    difference = (difference - box12);
    running12=FALSE;
    }
 if (box13 <= difference && running13 && box13>0)
    {
    llRegionSay (5, "Stop 13");
    difference = (difference - box13);
```

```
        running13=FALSE;
        }
if (box14 <= difference && running14 && box14>0)
        {
        llRegionSay (5, "Stop 14");
        difference = (difference - box14);
        running14=FALSE;
        }
if (box15 <= difference && running15 && box15>0)
        {
        llRegionSay (5, "Stop 15");
        difference = (difference - box15);
        running15=FALSE;
        }
if (box16 <= difference && running16 && box16>0)
        {
        llRegionSay (5, "Stop 16");
        difference = (difference - box16);
        running16=FALSE;
        }
}
```

B6 The Hybrid Model

The Hybrid Model script performs a set of initial tests to obtain baseline, maximum, goal and individual box (scripted object contributions). These calculations are then used to attempt to maintain FTime readings at the goal level but switching on / off scripted objects dependent on their estimated computational contribution.

```
   // Stop all scripted objects and measure baseline
    llRegionSay (5,"Stop All");
    baseline= Calftime ();
    llOwnerSay("Baseline = "+(string)baseline);

   // Start all scipted objects and measure baseline
    llRegionSay (5,"Start All");
    maximum=Calftime ();
    llOwnerSay("Maximum = "+(string)maximum);
    llRegionSay (5,"Stop All");

    // Calculate Goal
    goal = ((baseline + maximum) /2);

    // Start and Stop Each Scripted Objects in sequence and measure
    their individual
    contribution to the overall ftime value of the region.

    llRegionSay (5,"Start 1");
    float box1=Calftime () - baseline;
    llOwnerSay("Box 1 = "+(string)box1);
    llRegionSay (5,"Stop 1");
```

```
                 ..... Action repeated for boxes 2 - 15.....
```

```
    llRegionSay (5,"Start 16");
    float box16=Calftime () - baseline;
    llOwnerSay("Box 16 = "+(string)box16);
    llRegionSay (5,"Stop 16");

// Run the experiment for the required (counter) number of
iterations
for (counter=1;counter<101; counter++)
        {
        float ftime = ComputeLag(MAX_LOOP*10);
        ftime = ftime/10.0;

        // If ftime is equal to goal take no action
        if (ftime == goal)
            {
            llOwnerSay("Hitting Baseline - No Action Required");
            basecount=basecount+1;
            }

// If ftime is below goal add the required scripts using their
predicted contribution to attempt to hit goal
       else if (ftime < goal)
            {
            llOwnerSay("Below Goal - Adding Scripts");
            float difference = goal - ftime;
            llOwnerSay ("Diff = "+(string)difference);
            if (box1 <= difference && !running1)
        {
        llRegionSay (5, "Start 1");
        difference = (difference - box1);
        running1=TRUE;
        }
    if (box2 <= difference && !running2)
        {
        llRegionSay (5, "Start 2");
        difference = (difference - box2);
        running2=TRUE;
        }
    if (box3 <= difference && !running3)
        {
        llRegionSay (5, "Start 3");
        difference = (difference - box3);
        running3=TRUE;
        }
    if (box4 <= difference && !running4)
        {
        llRegionSay (5, "Start 4");
        difference = (difference - box4);
        running4=TRUE;
        }
     if (box5 <= difference && !running5)
        {
        llRegionSay (5, "Start 5");
        difference = (difference - box5);
        running5=TRUE;
        }
    if (box6 <= difference && !running6)
        {
        llRegionSay (5, "Start 6");
        difference = (difference - box6);
        running6=TRUE;
        }
    if (box7 <= difference && !running7)
        {
        llRegionSay (5, "Start 7");
        difference = (difference - box7);
        running7=TRUE;
        }
    if (box8 <= difference && !running8)
        {
        llRegionSay (5, "Start 8");
        difference = (difference - box8);
        running8=TRUE;
        }
     if (box9 <= difference && !running9)
        {
        llRegionSay (5, "Start 9");
        difference = (difference - box9);
        running9=TRUE;
        }
    if (box10 <= difference && !running10)
        {
        llRegionSay (5, "Start 10");
        difference = (difference - box10);
        running10=TRUE;
        }
    if (box11 <= difference && !running11)
        {
               llRegionSay (5, "Start 11");
               difference = (difference - box11);
               running11=TRUE;
               }
           if (box12 <= difference && !running12)
               {
               llRegionSay (5, "Start 12");
               difference = (difference - box12);
               running12=TRUE;
               }
            if (box13 <= difference && !running13)
               {
               llRegionSay (5, "Start 13");
               difference = (difference - box13);
               running13=TRUE;
               }
           if (box14 <= difference && !running14)
               {
               llRegionSay (5, "Start 14");
               difference = (difference - box14);
               running14=TRUE;
               }
           if (box15 <= difference && !running15)
               {
               llRegionSay (5, "Start 15");
               difference = (difference - box15);
               running15=TRUE;
               }
           if (box16 <= difference && !running16)
               {
               llRegionSay (5, "Start 16");
               difference = (difference - box16);
               running16=TRUE;
               }
            }
         // If ftime is above goal metric start switching off
scripted objects using their
         predicted contribution in order to try and hit goal reading

         else if (ftime > goal)
            {
            llOwnerSay("Above Baseline - Stopping Scripts");
            float difference = ftime - goal;
            llOwnerSay ("Diff = "+(string)difference);
            if (box1 <= difference && running1 && box1>0)
                {
                llRegionSay (5, "Stop 1");
                difference = (difference - box1);
                running1=FALSE;
                }
            if (box2 <= difference && running2 && box2>0)
                {
                llRegionSay (5, "Stop 2");
                difference = (difference - box2);
                running2=FALSE;
                }
            if (box3 <= difference && running3 && box3>0)
                {
                llRegionSay (5, "Stop 3");
                difference = (difference - box3);
                running3=FALSE;
```

```
    }
if (box4 <= difference && running4 && box4>0)
    {
    llRegionSay (5, "Stop 4");
    difference = (difference - box4);
    running4=FALSE;
    }
 if (box5 <= difference && running5 && box5>0)
    {
    llRegionSay (5, "Stop 5");
    difference = (difference - box5);
    running5=FALSE;
    }
if (box6 <= difference && running6 && box6>0)
    {
    llRegionSay (5, "Stop 6");
    difference = (difference - box6);
    running6=FALSE;
    }
if (box7 <= difference && running7 && box7>0)
    {
    llRegionSay (5, "Stop 7");
    difference = (difference - box7);
    running7=FALSE;
    }
if (box8 <= difference && running8 && box8>0)
    {
    llRegionSay (5, "Stop 8");
    difference = (difference - box8);
    running8=FALSE;
    }
 if (box9 <= difference && running9 && box9>0)
    {
    llRegionSay (5, "Stop 9");
    difference = (difference - box9);
    running9=FALSE;
    }
if (box10 <= difference && running10 && box10>0)
    {
    llRegionSay (5, "Stop 10");
    difference = (difference - box10);
    running10=FALSE;
    }
if (box11 <= difference && running11 && box11>0)
    {
    llRegionSay (5, "Stop 11");
    difference = (difference - box11);
    running11=FALSE;
    }
if (box12 <= difference && running12 && box12>0)
    {
    llRegionSay (5, "Stop 12");
    difference = (difference - box12);
    running12=FALSE;
    }
 if (box13 <= difference && running13 && box13>0)
    {
    llRegionSay (5, "Stop 13");
    difference = (difference - box13);
    running13=FALSE;
    }
```

```
        if (box14 <= difference && running14 && box14>0)
            {
            llRegionSay (5, "Stop 14");
            difference = (difference - box14);
            running14=FALSE;
            }
        if (box15 <= difference && running15 && box15>0)
            {
            llRegionSay (5, "Stop 15");
            difference = (difference - box15);
            running15=FALSE;
            }
        if (box16 <= difference && running16 && box16>0)
            {
            llRegionSay (5, "Stop 16");
            difference = (difference - box16);
            running16=FALSE;
            }

        }

}
```

B7 The Self Managing Controller

This LSL script calculates the baseline, maximum and goal metrics. It then “broadcasts” the goal into the Second Life Region on communication channel 6.

```
Main()

{
llEmail (email_address, "Starting","Self Managing Test");
llRegionSay (6,"restart");
llSleep(5.0);

llRegionSay (6,"Stop All");
baseline= Calftime ();
llOwnerSay("Baseline = "+(string)baseline);

llRegionSay (6,"Start All");
maximum=Calftime ();
llOwnerSay("Maximum = "+(string)maximum);

goal = ((baseline + maximum) /2);
llOwnerSay("Goal = "+(string)goal);
llRegionSay (6,(string)baseline);

llEmail (email_address, "Self Managing Stats are", "Baseline " +
(string)baseline + "\nMaximum " + (string)maximum + "\nGoal " +
(string)goal);

}
```

B8 Example of Self Managing Scripted Object (Box 1)

The self-managing scripted objects “listen” on communication channel 6 for the goal FTime being broadcast by the self-managing controller. Once a goal is received the scripted objects turn their own scripts on / off depending on environmental readings which indicate the need to lower / higher FTime readings.

```
integer MAX_LOOP = 1695; // Gives 1 second in lag free sim LSL
integer startTime;
integer endTime;
integer timeTaken;
float goal;
integer running=TRUE;
integer counter;
integer count1running=0;
integer count1paused=0;
string email_address ="H.Forsyth@ljmu.ac.uk";
string giv(integer type,integer i) {return
llGetInventoryName(type,i);}
integer goalcount=0;
integer loop_counter;


default

{
state_entry()
        {
        // listen on channel six for any messages broadcast by the
        Self Managing
        Controller
        llListen(6,"",NULL_KEY,"");
        }
listen(integer channel, string name, key id, string message)
        {
        if (llToLower(message) == "restart")
            {
            llResetScript();
            }
        else if (llToLower(message) == "start all")
            {
            integer x;
            for(x=1;x<llGetInventoryNumber(INVENTORY_SCRIPT);x++)
                   {
                   if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
                   //If the script number x is not this script
                   llSetScriptState(giv(INVENTORY_SCRIPT,x),TRUE);
                   //Start it
                   }
            llSetColor(<0,1,0>, ALL_SIDES); // set object colour
            green.
            }

        else if (llToLower(message) == "stop all")
            {
            integer x;

            for(x=0;x<llGetInventoryNumber(INVENTORY_SCRIPT);x++)
                   {
```

```
          //If the script number x is not this script

          if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
          llSetScriptState(giv(INVENTORY_SCRIPT,x),FALSE);
          //Stop it
          }
    llSetColor(<1,0,0>, ALL_SIDES); // set object colour
    red.
    }
else
    {

    for (loop_counter=1;loop_counter<101;loop_counter++)
       {
       integer startTime=llGetUnixTime();
       num1=3.142;
       num2=9.189;
       n=1;
       do
       {
       num3=num1*num2;
       num4=num3/num1;
       n=n+1;
       }
       while(n<=(10*MAX_LOOP));

       integer endTime=llGetUnixTime();
       float timeTaken=endTime-startTime;
       ftime=timeTaken/10.0;

       if (ftime == goal)
          {
          goalcount=goalcount+1;
          }

       if (goal < ftime && running)
          {
          integer x;
          for(x=0;x<llGetInventoryNumber(INVENTORY_SCRIPT);x
          ++)
            {
            if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
              {
              //If the script number x is not this script
              llSetScriptState(giv(INVENTORY_SCRIPT,x),
              FALSE);
              //Pause it
              }
            }
         running = FALSE;
         count1paused=count1paused+1;
         llSetColor(<1,0,0>, ALL_SIDES); // set object colour
         red
         }
   else if (goal < ftime && !running)
         {
         count1paused=count1paused+1;
         }

   else if (goal > ftime && !running)
         {
                 //If ftime< goal and the Script is not running
                       integer x;

                 for(x=0;x<llGetInventoryNumber(INVENTORY_SCRIPT);x
                 ++)
                   {
                   if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
                    {
                    llSetScriptState(giv(INVENTORY_SCRIPT,x),TRUE);
                    //Enable all scripts
                    }
                   }
            llSetColor(<0,1,0>, ALL_SIDES);
            // set object colour green.
            running = TRUE;
            count1running=count1running+1;
            }

      else if (goal >= ftime && running)
           {
           count1running=count1running+1;
           }
        } // end of For loop

integer x;
for(x=0;x<llGetInventoryNumber(INVENTORY_SCRIPT);x++)
         {
         if(giv(INVENTORY_SCRIPT,x) != llGetScriptName())
         //If the script number x is not this script
         llSetScriptState(giv(INVENTORY_SCRIPT,x),FALSE); //Stop it
         }
      llSetColor(<1,0,0>, ALL_SIDES); // set object colour red.
      } // end of else loop
    } // end of listen
} // end of default
```

Appendix C EMMA Prototype System Code

The header file defines the number of classifiers (MAXCLASS), the structures of the condition element (signed to allow -1 representing wildcards) and the action element (0 or 1). The structure of the classifier is also defined in the header file.

```
/* Header file for the VW Sim Control LCS system */
/* 4th Aug, 2009 */

#if !defined(_SIMLCS)
#define _SIMLCS
#include <stdio.h>

#define MAXCLASS     200

enum MATCHED {FALSE, TRUE} ; //FALSE = 0
enum INITFLAG {NO, YES} ; //NO = 0

typedef struct condition
   {
   signed pos_9 : 2 ;
   signed pos_8 : 2 ;
   signed pos_7 : 2 ;
   signed pos_6 : 2 ;
   signed pos_5 : 2 ;
   signed pos_4 : 2 ;
   signed pos_3 : 2 ;
   signed pos_2 : 2 ;
   signed pos_1 : 2 ;
   signed pos_0 : 2 ;
   } CONDITION ;

typedef struct action
   {
   unsigned act_15 : 1 ;
   unsigned act_14 : 1 ;
   unsigned act_13 : 1 ;
   unsigned act_12 : 1 ;
   unsigned act_11 : 1 ;
   unsigned act_10 : 1 ;
   unsigned act_9  : 1 ;
   unsigned act_8  : 1 ;
   unsigned act_7  : 1 ;
   unsigned act_6  : 1 ;
   unsigned act_5  : 1 ;
   unsigned act_4  : 1 ;
   unsigned act_3  : 1 ;
   unsigned act_2  : 1 ;
   unsigned act_1  : 1 ;
   unsigned act_0  : 1 ;
   } ACTION ;

typedef struct classifier
   {
   CONDITION    condition    ;
   ACTION       action       ;
   float        strength     ,
                cbid         ,
                bid          ,
                 bid1         ,
                 bid2         ,
                 ebid         ,
                 ebid1        ,
                 ebid2        ,
                 protection   ;
   enum MATCHED match         ;
   int           specificity ;
   int           duplicate    ;
   } CLASSIFIER ;

/* typedef struct matches
   {
   int clist[MAXCLASS] ;
   int nactive ;
   } MATCHES ;    */

#endif
```

Main Program: The main prototype program initialises relevant global declarations, variables and structures. The initial classifier ruleset is generated. An initial goal FTime is obtained (from the virtual simulator region). Environmental metrics are obtained and converted into binary strings (to match the format of the classifier condition).

Function calls to match classifier strings, creating a bid list (from the matching classifiers) and calculating a winning bid are made. The winning action is broadcast into the region and then the effect on the region is measured. The winning classifier is subsequently rewarded / punished depending on whether its actions have moved the FTime metric closer to the goal FTime.

At the end of an action cycle (typically consisting of between 10 – 100 observations), the genetic algorithm is invoked which replaces a proportion of the ruleset using the process of crossover from selected parents. At the end of each generation, mutation is applied before another action cycle begins.

```
/* A Learning Classifier System for Virtual World Sim Control */
/* Version 2.1 - 24th June 2010*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <dos.h>
#include "lcs.h" /* Type definitions for the learning classifiers */

#define OBSLENGTH 1
#define FLOATINT  100
#define INITIAL_STRENGTH 1000.0
#define WILD_CARD -1
#define CROWDINGFACTOR 3
#define CROWDINGSUBPOP 5
#define NUMBER_OF_COVERING 3
#define NPOSITION_CONDITION 10
#define NPOSITION_ACTION  16
#define MUTATION_RATE 0.005
#define ACTION_CYCLES 25
#define GA_TRIGGER    1000


/* Environmental message to binary struct */
struct bidigit
   {
   int value ;
   struct bidigit *next ;
   } ;
typedef struct bidigit *LINK ;
LINK topbidigit = NULL ;

/* Matched Classifier Struct */
struct matchmember
   {
   int    listindex ;
   struct matchmember *next ;
   } ;
typedef struct matchmember *MATCHLINK ;
MATCHLINK matchhead = NULL ;
```

```
int  calculate_specificity(CLASSIFIER [MAXCLASS], int) ;
int  closest_match(CLASSIFIER [MAXCLASS]);
enum INITFLAG initflag = YES   ;

/* Global variable declarations */
float life_tax = 0,
      bid_tax = 0,
      sumstrength = 0.0,
      maxstrength = 0.0,
      avgstrength = INITIAL_STRENGTH,
      minstrength = 0.0 ;
int   nposition  = 0,
      ncrossover = 0,
      nmutation  = 0 ;
int   previous_winner[NPOSITION_ACTION] = {0} ;
int   random_bid_flag = 0 ;
float proportion_select ;
int   specificity_quantifier[MAXCLASS] = {0} ;
int   ftime_history[GA_TRIGGER+1][ACTION_CYCLES+1] = {{0,0}} ;
float boundary1 = 0.40,
      boundary2 = 0.80,
      boundary3 = 1.20,
      boundary4 = 1.60,
      boundary5 = 2.00;

int   global_random_index  = 0 ;
float environmental_goal = 0.0 ;
float ftime_reading      = 0.0 ;
FILE *goal = NULL ;
FILE *ftimereading = NULL ;

/* Historic Matching */
void  put_digit_to_message(MESSAGE *, int, int) ;
void  match_message_classifiers(CLASSIFIER [MAXCLASS], MESSAGE *,
FILE *, int) ;
MESSAGE message                                 ;
MESSAGE *message_ptr                            ;

void main()
{
CLASSIFIER classifier_array[MAXCLASS]           ;

FILE  *ga_action_cycle       = fopen("ga_action.txt",  "w")      ;
FILE  *classifier_on         = fopen("class_evol.txt", "w")      ;
FILE  *classifier_strength   = fopen("classifier_st.txt", "w")   ;
FILE  *random_classifier     = fopen("random_classifiers.txt", "w")
;
FILE  *ga_activity           = fopen("GAStuff.txt", "w")         ;
FILE  *bid_history           = fopen("Bid_History.txt", "w")     ;
FILE  *spec_history          = fopen("Specificity_History.txt","w")
;
FILE  *protection_history    = fopen("Protection_History.txt", "w")
;
FILE  *classifier_model_file  = fopen("Classifier_Model.txt", "w") ;
FILE  *ftime_history_report   = fopen("FTime_History.txt", "w")    ;
FILE  *ftime_history_model    = fopen("FTime_History_Model.txt",
"w")    ;
FILE  *condition_model_report = fopen("Condition_Model.txt", "w")  ;
FILE  *classifier_report           = fopen("classifier.txt", "w") ;

void  init_message(MESSAGE *) ;
void  init_classifiers(CLASSIFIER [MAXCLASS]) ;
void  put_digit_to_message(MESSAGE *, int, int) ;
void  print_env_message(MESSAGE *) ;
void  print_classifier_array(CLASSIFIER [MAXCLASS],FILE *) ;
void  print_matched_classifiers(CLASSIFIER [MAXCLASS], MESSAGE *) ;
void  report_classifier_strength(CLASSIFIER [MAXCLASS], FILE *) ;
int   calculate_winning_bid(CLASSIFIER [MAXCLASS], float *) ;
void  clean_up_bid_list(CLASSIFIER [MAXCLASS], float *, int) ;
void  impose_life_tax(CLASSIFIER [MAXCLASS], float *) ;
void  effector(CLASSIFIER [MAXCLASS], int) ;
void  degrade_protection(CLASSIFIER [MAXCLASS], FILE *) ;
void  push_bidigit(int) ;
void  pop_bidigit(void) ;
void  put_digit_to_message(MESSAGE *, int, int) ;
void  match_message_classifiers(CLASSIFIER [MAXCLASS], MESSAGE *,
FILE *, int) ;
int   classifier_model(CLASSIFIER classifiers[MAXCLASS], MESSAGE
*message_ptr, FILE *)   ;
void  condition_model(CLASSIFIER classifiers[MAXCLASS], FILE
*condition_model_report) ;

int   select(CLASSIFIER [MAXCLASS])     ;
void  statistics(CLASSIFIER [MAXCLASS]) ;
void  crossover(CLASSIFIER parent1, CLASSIFIER parent2, CLASSIFIER
*child1, CLASSIFIER *child2) ;
int   crowding(CLASSIFIER child, CLASSIFIER[MAXCLASS]) ;
int   worstofn(CLASSIFIER[MAXCLASS]) ;
int   matchcount(CLASSIFIER classifier1, CLASSIFIER classifier2) ;
void  ga(CLASSIFIER[MAXCLASS], FILE *)          ;
void  specificity_quantified(CLASSIFIER[MAXCLASS]) ;
float avg(float, float)                 ;
void  get_goal(void)                    ;
float get_reading(void)                 ;

int     obs_array[OBSLENGTH]              ;
int     input_count                       ;
int     digit_value                       ;
int     bi_conversion_count               ;
int     bid_winner                        ;
int     message_value =  0          ;
float   bid_payment                       ;
int     old_winner                        ;
int     ga_cycle_count                    ;
int     action_count                      ;
float   environment_account = 10000 ;
float   input_value                       ;
float   pre_action_ftime  = 0.0           ;
float   post_action_ftime = 0.0           ;
float   comparison_1,
        comparison_2                      ;
int     boundarycount1,
        boundarycount2,
        boundarycount3,
        boundarycount4,
        boundarycount5,
        the_rest                                      ;
float   the_score = 0.0                         ;
int     g_cycle_model               ;
int     a_cycle_model               ;
int     ga_model_total                          ;
```

```
/* Temporary place for message_ptr initialisation */
message_ptr = &message ;
srand(time(NULL)) ;
init_classifiers(classifier_array) ;
get_goal() ;
goal = NULL ;
printf("Environmental goal is %0.2f...\n", environmental_goal) ;
pre_action_ftime = get_reading() ;

for(ga_cycle_count = 0; ga_cycle_count <= GA_TRIGGER;
ga_cycle_count++)
   {
   bid_tax = avgstrength / 100.0;
   life_tax = avgstrength / 200.0;
   the_score = 0.0      ;
   boundarycount1 = 0;
   boundarycount2 = 0;
   boundarycount3 = 0;
   boundarycount4 = 0;
   boundarycount5 = 0;
   the_rest = 0         ;
   specificity_quantified(classifier_array) ;
   for(action_count = 0; action_count < ACTION_CYCLES;
   action_count++)
      {
      printf("Ftime reading is... %0.2f...\n", ftime_reading) ;
      for(input_count = 0; input_count < OBSLENGTH; input_count++)
         {
         obs_array[input_count] = message_value
         =(int)((pre_action_ftime + 0.001)* FLOATINT) ;
         ftime_history [ga_cycle_count][action_count] = obs_array
         [input_count] ;
         printf ("\n Ftime Value at array row position %d action
         cycle %d is %d\n", ga_cycle_count, action_count,
         ftime_history [ga_cycle_count][action_count]) ;
         }
      for(input_count = 0; input_count < OBSLENGTH; input_count++)
         {
         init_message(message_ptr) ;
         bi_conversion_count = 0 ;
         digit_value = obs_array[input_count] ;
         while(digit_value > 0)
            {
            put_digit_to_message(message_ptr, bi_conversion_count,
            (digit_value % 2)) ;
            bi_conversion_count++ ;
            push_bidigit(digit_value % 2) ;
            digit_value = digit_value / 2 ;
            }
         while(topbidigit != NULL)
            {
            printf("%2d", topbidigit->value) ;
            pop_bidigit() ;
            }
         printf("\n") ;
         print_env_message(message_ptr) ;
         }
      printf("End of observation conversion sequence...\n") ;

      for(input_count = 0; input_count < OBSLENGTH; input_count++)
         {
```

```
      printf("%d\n", obs_array[input_count]) ;
      }
    printf("End of observation array values...\n\n") ;
    // print_classifier_array(classifier_array, classifier_report)
;
    match_message_classifiers(classifier_array, message_ptr,
    random_classifier, message_value) ;
    print_matched_classifiers(classifier_array, message_ptr) ;
    bid_winner = calculate_winning_bid(classifier_array,
    &environment_account) ;
    printf("\n *** Winning Bid is %d ***", bid_winner) ;
    clean_up_bid_list(classifier_array, &environment_account,
    bid_winner) ;
    impose_life_tax(classifier_array, &environment_account) ;
    effector(classifier_array, bid_winner) ;
    post_action_ftime = get_reading() ;

    if(pre_action_ftime > environmental_goal)
      {
      comparison_1 = pre_action_ftime - environmental_goal ;
      }
    else
      {
      comparison_1 = environmental_goal - pre_action_ftime ;
      }
    if(post_action_ftime > environmental_goal)
      {
      comparison_2 = post_action_ftime - environmental_goal ;
      }
    else
      {
      comparison_2 = environmental_goal - post_action_ftime ;
      }
    /* The line below protects all bid_winners which promotes
    their condition during GA mutation */
    classifier_array[bid_winner].protection = 1.0 ;

    if (post_action_ftime == environmental_goal)
         {
         printf("\nGreat, it hit the goal!...") ;
         bid_payment += 0.5 * classifier_array[bid_winner].bid ;
         classifier_array[bid_winner].strength += bid_payment ;
         environment_account -= bid_payment ;
         bid_payment = 0.0 ;
         }

    if(comparison_1 < comparison_2)
         {
         printf("\nIt made it worse!\n") ;
         bid_payment += 0.5 * classifier_array[bid_winner].bid ;
         classifier_array[bid_winner].strength -= bid_payment ;
         environment_account += bid_payment ;
         }
    else if(comparison_1 > comparison_2)
         {
         printf("\nIt made it better!...") ;
         bid_payment += 0.5 * classifier_array[bid_winner].bid ;
         classifier_array[bid_winner].strength += bid_payment ;
         environment_account -= bid_payment ;
         }
    else
```

```
                {
                printf("\n*** No change... ***\n") ;
                }
        fprintf(bid_history,"Cycle Count: %d Random Flag: %d
        Classifier %d Initial Bid %0.4f Bid Payment %0.3f Env Acc:
        %0.4f\n", action_count, random_bid_flag, bid_winner,
        classifier_array[bid_winner].bid, bid_payment,
        environment_account) ;
        bid_payment = 0.0 ;
        random_bid_flag = 0 ;
        classifier_array[bid_winner].bid = 0.0;
        printf("\nGoal = %0.2f Pre = %0.2f Post = %0.2f\n\n",
        environmental_goal, pre_action_ftime, post_action_ftime) ;
        if (post_action_ftime>=(environmental_goal- boundary1) &&
        post_action_ftime<=(environmental_goal+ boundary1))
                {
                boundarycount1=boundarycount1+1;
                }
        else if (post_action_ftime>=(environmental_goal- boundary2) &&
        post_action_ftime<=(environmental_goal+ boundary2))
                {
                boundarycount2=boundarycount2+1;
                }
        else if (post_action_ftime>=(environmental_goal- boundary3) &&
        post_action_ftime<=(environmental_goal+ boundary3))
                {
                boundarycount3=boundarycount3+1;
                }
        else if (post_action_ftime>=(environmental_goal- boundary4) &&
        post_action_ftime<=(environmental_goal+ boundary4))
                {
                boundarycount4=boundarycount4+1;
                }
        else if (post_action_ftime>=(environmental_goal- boundary5) &&
        post_action_ftime<=(environmental_goal+ boundary5))
                {
                boundarycount5=boundarycount5+1;
                }
        else
                {
                the_rest=the_rest+1;
                }

        if (post_action_ftime > environmental_goal)
                {
                the_score += (post_action_ftime - environmental_goal)*
                10.0;
                }
        else if (environmental_goal > post_action_ftime)
                {
                the_score += (environmental_goal - post_action_ftime)*
                10.0;
                }
        fprintf(ga_action_cycle,"ACycle Count: %d Goal = %0.2f Pre =
%0.2f Post = %0.2f Performance is %.2f \n", action_count,
environmental_goal, pre_action_ftime, post_action_ftime, the_score)
;
        old_winner = bid_winner ;
        pre_action_ftime = post_action_ftime ;
        printf("\n *** Environmental Account is %f ***",
environment_account) ;
```

```
      // report_classifiers(classifier_array, classifier_on) ;
      }
   for(g_cycle_model = 0; g_cycle_model <= ga_cycle_count;
g_cycle_model++)
         {
         float ga_model_total = 0;
         float ga_total_percentage;
         for(a_cycle_model = 0; a_cycle_model < ACTION_CYCLES;
a_cycle_model++)
            {
            int     match_model_total = 0 ;
            init_message(message_ptr) ;
            bi_conversion_count = 0 ;
            digit_value = ftime_history
[g_cycle_model][a_cycle_model] ;
            fprintf(ftime_history_report, "GA Cycle : %d  Act Cycle
: %d Digital Value : %d\n", g_cycle_model, a_cycle_model,
digit_value);
            while(digit_value > 0)
               {
               put_digit_to_message(message_ptr,
bi_conversion_count, (digit_value % 2)) ;
               bi_conversion_count++ ;
               push_bidigit(digit_value % 2) ;
               digit_value = digit_value / 2 ;
               }
            match_model_total = classifier_model(classifier_array,
message_ptr, ftime_history_model) ;
            if (match_model_total == 1)
                  {
               ga_model_total += 1;
               }
            fprintf(classifier_model_file, "GA Cycle: %d: %d\n",
g_cycle_model, match_model_total) ;
            match_model_total = 0;
            while(topbidigit != NULL)
               {
               printf("%2d", topbidigit->value) ;
               pop_bidigit() ;
               }
            }

            ga_total_percentage = (ga_model_total / ACTION_CYCLES) *
            100;
            fprintf(classifier_model_file, "GA Total: %f: \n\n",
            ga_model_total) ;
                        fprintf(classifier_model_file, "GA Model
            Percentage : %f \n\n", ga_total_percentage)  ;
            fprintf (ftime_history_model, "End of GA Cycle %d
            \n\n",g_cycle_model);
         }
   condition_model(classifier_array, condition_model_report) ;
   if (the_score >=0 && the_score<ACTION_CYCLES*2)
            {
         proportion_select = 0.05;
         }
   else if (the_score >= ACTION_CYCLES*2 && the_score <
ACTION_CYCLES*4)
            {
         proportion_select = 0.10;
         }
   else if (the_score >=ACTION_CYCLES*4 &&
            the_score<ACTION_CYCLES*6)
            {
         proportion_select = 0.15;
            }
   else if (the_score >= ACTION_CYCLES*6 && the_score
            <ACTION_CYCLES*8)
            {
         proportion_select = 0.20;
            }
   else
         {
         proportion_select = 0.25;
         }
   print_classifier_array(classifier_array, classifier_report) ;
   ga(classifier_array, ga_activity) ;
   degrade_protection(classifier_array, protection_history) ;
   report_classifier_strength(classifier_array, classifier_strength)
   ;
   fprintf(ga_action_cycle,"Boundary 1 Count: %d\nBoundary 2 Count:
%d\nBoundary 3 Count: %d\nBoundary 4 Count: %d\nBoundary 5 Count:
%d\n OoB Count      : %d  Proportion Selected is %f \n",
boundarycount1, boundarycount2, boundarycount3, boundarycount4,
boundarycount5, the_rest, proportion_select) ;
   fprintf(ga_action_cycle,"GA Count: %d\n", ga_cycle_count) ;
   }
fprintf(ga_action_cycle,"The Score is : %f\n", the_score) ;
fclose(ga_action_cycle)          ;
fclose(classifier_on)            ;
fclose(random_classifier)        ;
fclose(ga_activity)              ;
fclose(bid_history)              ;
fclose(spec_history)             ;
fclose(protection_history)       ;
fclose(classifier_model_file)    ;
fclose(ftime_history_report)     ;
fclose(ftime_history_model)      ;
fclose(condition_model_report)   ;
fclose(classifier_report)        ;
}
```

Degrade Protection: This function degrades parent protection on all "currently" protected classifiers (which promotes the reproduction of recently matched classifiers) before the Genetic Algorithm is run.

```
void  degrade_protection(CLASSIFIER classifiers [MAXCLASS], FILE
*output)
 {
 int count ;

 degrade_amount = ACTION_CYCLES / 100 ;
 for(count = 0; count < MAXCLASS; count++)
    {
    if(classifiers[count].protection > 0.0)
       {
       classifiers[count].protection -= degrade_amount ;
       }
    if(classifiers[count].protection < 0.0)
       {
       classifiers[count].protection = 0.0 ;
       }
    fprintf(output,"%d: %0.2f\t", count,
classifiers[count].protection) ;
    }
  fprintf(output,"\n") ;
}
```

Initialise Classifiers: This function is used in the initialisation of the classifiers. It randomly generated either a -1,0 or 1 in each of the classifier condition positions and also either a 0 or 1 in each of the action positions. It then calls a function to calculate the specificity of the classifier as well as initialising a set of initialisation values (such as strength, protection etc).

```
void init_classifiers(CLASSIFIER classifiers [MAXCLASS])
{
 int init_count ;
 FILE *specificities = fopen("specificities.txt", "w") ;
 FILE *sumstrengths = fopen("sumstrength.txt", "w") ;
 for(init_count = 0; init_count < MAXCLASS; init_count++)
    {
     classifiers[init_count].condition.pos_9 = (rand() % 3) - 1 ;
----------------------------------------------------------------------
                Code Repeated for conditions 8 to 1
----------------------------------------------------------------------
     classifiers[init_count].condition.pos_0 = (rand() % 3) - 1 ;

     classifiers[init_count].specificity =
     calculate_specificity(classifiers, init_count) ;
     fprintf(specificities, "%d\n",
     classifiers[init_count].specificity) ;
     classifiers[init_count].action.act_15 = rand() % 2 ;
----------------------------------------------------------------------
                 Code Repeated for actions 14 to 1
----------------------------------------------------------------------
     classifiers[init_count].action.act_0  = rand() % 2 ;
     classifiers[init_count].strength      = INITIAL_STRENGTH ;
     classifiers[init_count].cbid          = 0.10 ;
     classifiers[init_count].bid1          = 1.0  ;
     classifiers[init_count].bid2          = 0.0  ;
     classifiers[init_count].ebid1         = 1.0  ;
     classifiers[init_count].ebid2         = 0.0  ;
     classifiers[init_count].protection    = 0.0  ;
     classifiers[init_count].duplicate     = 0 ;
     fprintf(sumstrengths, "%0.3f\n",
     classifiers[init_count].strength) ;
    }
 fclose(specificities) ;
 fclose(sumstrengths) ;
}
```

Calculate Specificity: This function is used to calculate the specificity of a classifier. Specificity is the total of 0s and 1s in a classifier condition. Any positions containing the WILD_CARD value (-1) are omitted.

```
int  calculate_specificity(CLASSIFIER classifiers[MAXCLASS], int
index)
{
int specificity_total = 0 ;

   if(classifiers[index].condition.pos_9 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_8 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_7 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_6 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_5 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_4 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_3 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_2 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_1 != WILD_CARD)
      specificity_total += 1 ;
   if(classifiers[index].condition.pos_0 != WILD_CARD)
      specificity_total += 1 ;
   return(specificity_total) ;
}
```

Matching Message Classifiers: This function seeks to find all classifiers in the current rule set which match the environmental message being received. Matching classifiers are subsequently inserted into the bid list (using insert_new_matchmember function). If there are no matching classifiers then a “covering classifier” is generated and inserted into the rule-set (replacing one of the “worst” classifiers in the current rule-set).

```
void match_message_classifiers(CLASSIFIER classifiers[MAXCLASS],
MESSAGE *message_ptr,FILE *random_classifier_file, int
message_value)
{
FILE *output = fopen("matchs.txt", "w") ;
int  find_closest_match(int message_val) ;
void insert_new_matchmember(int) ;
enum MATCHED match = TRUE   ;
int match_count ;
int matches = 0 ;    // checks for no matches on this environment
message
int random_classifier ;
MATCHLINK temp_match ;

for(match_count = 0; match_count < MAXCLASS; match_count++)
   {
   if(classifiers[match_count].condition.pos_9 != -1)
      {
      if(classifiers[match_count].condition.pos_9 != message_ptr-
      >mess_9)
      match = FALSE ;
      }
```

```
              Code Repeated for condition checks 8 to 1
```

```
   if(classifiers[match_count].condition.pos_0 != -1)
      {
      if(classifiers[match_count].condition.pos_0 != message_ptr-
      >mess_0)
      match = FALSE ;
      }
   classifiers[match_count].match = match ;
   if(match == TRUE)
      {
      matches++ ;
      }
   match = TRUE ;
   if(classifiers[match_count].match == TRUE)
      {
      insert_new_matchmember(match_count) ;
      }
   }
if(matches == 0)
   {
   int   count ;
   int   covering_count;
   int   worst ;
   int   candidate ;
   float worst_strength ;

   for(covering_count = 0; covering_count < NUMBER_OF_COVERING;
   covering_count++)
      {
      worst = rand() % MAXCLASS ;
      worst_strength =  classifiers[worst].strength ;
      for(count = 1; count < CROWDINGSUBPOP; count++)
            {
            candidate = rand() % MAXCLASS ;
            if(worst_strength > classifiers[candidate].strength)
                  {
                  worst = candidate ;
                  worst_strength = classifiers[worst].strength ;
                  }
            }

      if (((rand()%3)-1) == -1)
            {
            classifiers[worst].condition.pos_9 = - 1 ;
            }
      else
            {
            classifiers[worst].condition.pos_9 = message_ptr-
            >mess_9;
            }
```

Code Repeated for Condition Check 8 to 1

```
      if (((rand()%3)-1) == -1)
            {
            classifiers[worst].condition.pos_0 = - 1 ;
            }
      else
            {
            classifiers[worst].condition.pos_0 = message_ptr-
            >mess_0;
            }

      classifiers[worst].action.act_15 = rand() % 2 ;
```

Code Repeated for Action Initialisation at Positions 14 to 1

```
      classifiers[worst].action.act_0  = rand() % 2 ;
      classifiers[worst].strength      = avgstrength ;
      classifiers[worst].cbid          = 0.10 ;
      classifiers[worst].bid1          = 1.0  ;
      classifiers[worst].bid2          = 0.0  ;
      classifiers[worst].ebid1         = 1.0  ;
      classifiers[worst].ebid2         = 0.0  ;
      classifiers[worst].protection    = 0.0  ;
      classifiers[worst].duplicate     = 0 ;
      classifiers[worst].specificity =
      calculate_specificity(classifiers, worst) ;
      insert_new_matchmember(worst) ;
      random_bid_flag = 0 ;
      }
   }
temp_match = matchhead ;
while(temp_match != NULL)
   {
   fprintf(output,"%d ", temp_match->listindex) ;
   temp_match = temp_match->next ;
   }
fclose(output) ;
}
```

Classifer_Model: This function is used to compare all previous environmental readings to the current classifier conditions. This outputs a text file which shows how effective the current model would be in terms of matching previous environmental messages. An example of this may be that the current model (10th Generation) can now only match 80% of the messages received in the first generation.

```
int classifier_model(CLASSIFIER classifiers[MAXCLASS], MESSAGE
*message_ptr, FILE *ftime_history_model)
{

enum MATCHED match = TRUE   ;
int match_count ;
int matches = 0 ;    // checks for no matches on this environment
message
fprintf (ftime_history_model, "New Digital Value Sent to
Classifier_model function \n");
for(match_count = 0; match_count < MAXCLASS; match_count++)
   {
   if(classifiers[match_count].condition.pos_9 != -1)
      {
      if(classifiers[match_count].condition.pos_9 != message_ptr-
      >mess_9)
      match = FALSE ;
      }
```

```
                 Code Repeated for Condition Checks 8 to 1
```

```
   if(classifiers[match_count].condition.pos_0 != -1)
      {
      if(classifiers[match_count].condition.pos_0 != message_ptr-
      >mess_0)
      match = FALSE ;
      }

   if(match == TRUE)
      {
      matches = 1 ;
      fprintf (ftime_history_model, "Match at Classifier Position
      %d\n", match_count);
      }
   match = TRUE ;
   }

if (matches == 0)
      {
      fprintf (ftime_history_model, "No Match Found in Current
      Model\n");
      return(0);
      }
if (matches == 1)
      {
      return (1);
      }
}
```

Condition_Model: This function tests all potential environmental values against the current classifier set to enable analysis of the current model of external environment. It checks whether a potential environmental message has a match in the condition element of the current classifier set.

```
void condition_model(CLASSIFIER classifiers[MAXCLASS], FILE
*condition_model_report)
{
enum MATCHED match = TRUE      ;
int  condition_count           ;
int  match_count               ;
int  bi_conversion_count       ;
int  digit_value               ;

for(condition_count = 1; condition_count <= 1024; condition_count++)
   {
   fprintf (condition_model_report, "%d:\t", condition_count);
   init_message(message_ptr) ;
   bi_conversion_count = 0 ;
   digit_value = condition_count ;
   while(digit_value > 0)
      {
      put_digit_to_message(message_ptr, bi_conversion_count,
      (digit_value % 2)) ;
      bi_conversion_count++ ;
      push_bidigit(digit_value % 2) ;
      digit_value = digit_value / 2 ;
      }
   for(match_count = 0; match_count < MAXCLASS; match_count++)
      {
      if(classifiers[match_count].condition.pos_9 != -1)
         {
         if(classifiers[match_count].condition.pos_9 != message_ptr-
         >mess_9)
         match = FALSE ;
         }
              Code Repeated for Condition Checks 8 to 1

      if(classifiers[match_count].condition.pos_0 != -1)
         {
         if(classifiers[match_count].condition.pos_0 != message_ptr-
         >mess_0)
         match = FALSE ;
         }
   if(match == TRUE)
      {
      fprintf (condition_model_report, " %d ", match_count);
      }
   match = TRUE ;
   while(topbidigit != NULL)
         {
         pop_bidigit() ;
         }
    }
  fprintf (condition_model_report, "\n");
 }
}
```

Calculate Winning Bid: This function goes through the bid list (classifiers matching the environmental message) and calculates the winning classifier to be enacted. The bid is calculated using a classifier strength and specificity as major components of the bid amount. If there are duplicate bids then a random winnder is selected.

```
int calculate_winning_bid(CLASSIFIER classifiers[MAXCLASS], float
*env_account)
{
/* Attempts to implement random selection of duplicate bids if they
occur */

float     winning_bid          = 0.0   ,
          random_bid_element           ;
int       winning_classifier           ;
int       duplicate_flag       = 0     ;
int       duplicate_count      = 0     ;
int       random_duplicate     = 0     ;
MATCHLINK temp_match                   ;
FILE      *output = fopen("bids.txt", "w") ;

temp_match = matchhead ;
while(temp_match != NULL)
   {
   random_bid_element = (float)(rand() % 1001) / 1000 ;
   classifiers[temp_match->listindex].bid = 0.1 *
   ((classifiers[temp_match->listindex].specificity) *
   (classifiers[temp_match->listindex].strength));
   classifiers[temp_match->listindex].bid += random_bid_element;
   if(classifiers[temp_match->listindex].bid == winning_bid)
      {
      duplicate_flag   = 1 ;
      duplicate_count += 1 ;
      classifiers[temp_match->listindex].duplicate = duplicate_count
      ;
      }
   if(classifiers[temp_match->listindex].bid > winning_bid)
      {
      duplicate_flag = 0 ;
      duplicate_count = 0 ;
      classifiers[temp_match->listindex].duplicate = duplicate_count
      ;
      winning_bid = classifiers[temp_match->listindex].bid ;
      winning_classifier = temp_match->listindex ;
      }
   temp_match = temp_match->next ;
   }
  if(duplicate_flag)
      {
      random_duplicate =  rand() % (duplicate_count + 1) ;
      temp_match = matchhead ;
      while((classifiers[temp_match->listindex].duplicate !=
      random_duplicate) && temp_match != NULL)
            {
            printf("\nRandom Duplicate Unmatched (not selected) = %d
            Classifier Duplicate count = %d\n", random_duplicate,
            classifiers[temp_match->listindex].duplicate) ;
            temp_match = temp_match->next ;
            }
      printf("\nRandom Duplicate Selected = %d Classifier Duplicate
      count = %d", random_duplicate, classifiers[temp_match-
      >listindex].duplicate) ;
      if(classifiers[temp_match->listindex].bid == winning_bid)
```

```
            {
            winning_classifier = temp_match->listindex ;
            }
      }
fclose(output) ;
}
```

Impose Life Tax: After each action cycle a life tax is deducted from the strength of all classifiers. This is to "punish" non-bidding classifiers.

```
void  impose_life_tax(CLASSIFIER classifiers[MAXCLASS], float
*env_account)
{
int index ;

for(index = 0; index < MAXCLASS; index++)
   {
    classifiers[index].strength -= life_tax ;
   *env_account += life_tax ;
   }
}
```

Select Classifier: The selection of a classifier is performed using a randomised "roulette wheel" approach which is weighted according to classifier strength (therefore stronger classifiers are more likely to be selected).

```
int select(CLASSIFIER classifiers[MAXCLASS])
{
float  rand_int    ,
       holdfloat   ,
       partsum     ;
int    index       ;

partsum = 0.0      ;
index = -1         ;
holdfloat = (float)(rand() % 1001) / 1000 ;
rand_int = sumstrength * holdfloat ;
printf ("Sumstrength = %f holdfloat = %f partsum= %f rand_int =
%f",sumstrength, holdfloat, partsum, rand_int) ;
do
   {
   index = index++ ;
   partsum = partsum + classifiers[index].strength ;
   }
while((partsum <= rand_int) && (index < MAXCLASS - 1)) ;

return(index) ;
}
```

WorstofN: This function is used to select "weak" classifiers amongst the rule-set. It begins by choosing a random classifier as the "current worst". It then compares the current worst to a number (CROWDINGSUBPOP) of randomly chosen classifiers. At the end of the function the classifier with the lowest strength is returned as the "worst". This function is mainly used for selection of classifiers to be candidates for deletion from the ruleset.

```
int   worstofn(CLASSIFIER classifiers[MAXCLASS])
{
 int   count ;
 int   worst ;
 int   candidate ;
 float worst_strength ;

worst = rand() % MAXCLASS ;
worst_strength =  classifiers[worst].strength ;
for(count = 1; count < CROWDINGSUBPOP; count++)
   {
   candidate = rand() % MAXCLASS ;
   if(worst_strength > classifiers[candidate].strength)
      {
      worst = candidate ;
      worst_strength = classifiers[worst].strength ;
      }
   }
return(worst) ;
}
```

Crowding: This function implements DeJong's crowding factor by choosing the classifier to be deleted (from the candidate list) that is the most similar to new classifier being inserted into the rule-set. This functions helps to maintain diversity in the ruleset.

```
int   crowding(CLASSIFIER child, CLASSIFIER classifiers[MAXCLASS])
{
int popmember ;
int count ;
int match ;
int matchmax ;
int most_similar ;

matchmax = -1 ;
most_similar = 0 ;
for(count = 0; count < CROWDINGFACTOR; count++)
   {
    popmember = worstofn(classifiers) ;
    match = matchcount(child, classifiers[popmember]) ;
    if(match > matchmax)
       {
       matchmax = match ;
       most_similar = popmember ;
       }
    }
return(most_similar) ;
}
```

Crossover: The crossover function is used to generate child classifiers as part of the Genetic Algorithm process. If a selected parent classifier is "protected" its condition element is used, in entirety, for the child. Otherwise a random cross over point in selected and the child is produced using a combination of both parents. The action element of the classifier is generated using a cross over point (parent protection does not affect this element).

```
void  crossover(CLASSIFIER parent1, CLASSIFIER parent2, CLASSIFIER
*child1, CLASSIFIER *child2)
{
int   get_cross_over_point(int) ;
float inheritance ;
int   cross_point ;
int   count1 = 0 ;
int   specificity_total ;
CLASSIFIER *child ;

child = child1 ;
while(count1 < 2)
      {
      cross_point = get_cross_over_point(NPOSITION_CONDITION) ;
      if(parent1.protection > 0.0 && parent2.protection == 0.0 &&
      count1 == 0)
            {
            cross_point = 9 ;
            }
      if(parent2.protection > 0.0 && parent1.protection == 0.0 &&
      count1 == 0)
            {
            cross_point = 10 ;
            }
      if(parent1.protection > 0.0 && parent2.protection > 0.0 &&
      count1 == 0)
            {
            cross_point = 9 ;
            }
      if(parent2.protection > 0.0 && parent1.protection > 0.0 &&
      count1 == 1)
            {
            cross_point = 10 ;
            }
      switch(cross_point)
            {
            case 0 :
            {
            child->condition.pos_0   = parent1.condition.pos_0   ;
            child->condition.pos_1   = parent2.condition.pos_1   ;
            child->condition.pos_2   = parent2.condition.pos_2   ;
            child->condition.pos_3   = parent2.condition.pos_3   ;
            child->condition.pos_4   = parent2.condition.pos_4   ;
            child->condition.pos_5   = parent2.condition.pos_5   ;
            child->condition.pos_6   = parent2.condition.pos_6   ;
            child->condition.pos_7   = parent2.condition.pos_7   ;
            child->condition.pos_8   = parent2.condition.pos_8   ;
            child->condition.pos_9   = parent2.condition.pos_9   ;
            break ;
            }
```

Code Repeated with different condition crossover points of
Parents 1 & 2

```
      case 10 :
      {
      child->condition.pos_0    = parent2.condition.pos_0   ;
      child->condition.pos_1    = parent2.condition.pos_1   ;
      child->condition.pos_2    = parent2.condition.pos_2   ;
      child->condition.pos_3    = parent2.condition.pos_3   ;
      child->condition.pos_4    = parent2.condition.pos_4   ;
      child->condition.pos_5    = parent2.condition.pos_5   ;
      child->condition.pos_6    = parent2.condition.pos_6   ;
      child->condition.pos_7    = parent2.condition.pos_7   ;
      child->condition.pos_8    = parent2.condition.pos_8   ;
      child->condition.pos_9    = parent2.condition.pos_9   ;
      break ;
      }
 }

cross_point = get_cross_over_point(NPOSITION_ACTION) ;
 switch(cross_point)
      {
      case 0 :
            {
            child->action.act_0    =  parent1.action.act_0   ;
            child->action.act_1    =  parent2.action.act_1   ;
            child->action.act_2    =  parent2.action.act_2   ;
            child->action.act_3    =  parent2.action.act_3   ;
            child->action.act_4    =  parent2.action.act_4   ;
            child->action.act_5    =  parent2.action.act_5   ;
            child->action.act_6    =  parent2.action.act_6   ;
            child->action.act_7    =  parent2.action.act_7   ;
            child->action.act_8    =  parent2.action.act_8   ;
            child->action.act_9    =  parent2.action.act_9   ;
            child->action.act_10   =  parent2.action.act_10 ;
            child->action.act_11   =  parent2.action.act_11 ;
            child->action.act_12   =  parent2.action.act_12 ;
            child->action.act_13   =  parent2.action.act_13 ;
            child->action.act_14   =  parent2.action.act_14 ;
            child->action.act_15   =  parent2.action.act_15 ;
            break ;
            }
```

Code Repeated with different action crossover points of
Parents 1 & 2

```
      case 14 :
            {
            child->action.act_0    =  parent1.action.act_0   ;
            child->action.act_1    =  parent1.action.act_1   ;
            child->action.act_2    =  parent1.action.act_2   ;
            child->action.act_3    =  parent1.action.act_3   ;
            child->action.act_4    =  parent1.action.act_4   ;
            child->action.act_5    =  parent1.action.act_5   ;
            child->action.act_6    =  parent1.action.act_6   ;
            child->action.act_7    =  parent1.action.act_7   ;
            child->action.act_8    =  parent1.action.act_8   ;
            child->action.act_9    =  parent1.action.act_9   ;
            child->action.act_10   =  parent1.action.act_10 ;
            child->action.act_11   =  parent1.action.act_11 ;
            child->action.act_12   =  parent1.action.act_12 ;
            child->action.act_13   =  parent1.action.act_13 ;
            child->action.act_14   =  parent1.action.act_14 ;
```

```
                  child->action.act_15  =  parent2.action.act_15 ;
                  break ;
                  }
         }
      inheritance       = (parent1.strength + parent2.strength) /2 ;
      child->strength   = inheritance ;
      child->match      = 0 ;
      child->cbid       = 0.10 ;
      child->bid        = 0.0 ;
      child->bid1       = 1.0 ;
      child->bid2       = 0.0 ;
      child->ebid1      = 1.0 ;
      child->ebid2      = 0.0 ;
      child->protection  = 0.0 ;
      child->duplicate  = 0 ;
      specificity_total = 0 ;
      if(child->condition.pos_9 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_8 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_7 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_6 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_5 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_4 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_3 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_2 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_1 != WILD_CARD)
         specificity_total += 1 ;
      if(child->condition.pos_0 != WILD_CARD)
         specificity_total += 1 ;
      child->specificity = specificity_total ;
      child = child2 ;
      count1++ ;
      }
}


int get_cross_over_point(int number_of_positions)
{
int cross_point ;

cross_point = rand() % (number_of_positions - 1) ;
return(cross_point) ;
}
```

Genetic Algorithm: At the end of each cycle, the genetic algorithm is run. A proportion of the ruleset is selected for replacement. The select function produces parent classifiers and the crossover function is used to produce children from these parents. The crowding function is used to select learning classifiers to be deleted (whilst maintaining diversity in the population).

```
void  ga(CLASSIFIER classifiers[MAXCLASS], FILE *ga_activity)
{
void bin_mutation(CLASSIFIER classifiers[MAXCLASS]) ;
void tri_mutation(CLASSIFIER classifiers[MAXCLASS]) ;
CLASSIFIER child1, child2 ;
int nselect,
    index  ,
    mate1  ,
    mate2  ,
    mort1  ,
    mort2  ;

nselect =  (int) (proportion_select * MAXCLASS * 0.5) ;
statistics(classifiers) ;
for(index = 0; index < nselect; index++)
   {
   mate1 = select(classifiers) ;
   mate2 = select(classifiers) ;
   printf ("Mate 1 = %2d Mate 2 = %2d", mate1, mate2);
   crossover(classifiers[mate1], classifiers[mate2], &child1,
   &child2) ;
   mort1 = crowding(child1, classifiers) ;
   sumstrength -= classifiers[mort1].strength ;
   classifiers[mort1] = child1 ;
   sumstrength += classifiers[mort1].strength ;
   mort2 = crowding(child2, classifiers) ;
   sumstrength -= classifiers[mort2].strength ;
   classifiers[mort2] = child2 ;
   sumstrength += classifiers[mort2].strength ;
   fprintf(ga_activity,"Index No: %2d\t Mate 1 = %2d\t Mate 2 =
   %2d\t Mort 1 = %2d\t Mort 2 = %2d\t Sum Strength = %f\n", index,
   mate1, mate2, mort1, mort2, sumstrength) ;
   }

printf("\n\nWell! it didn't crash!") ;
bin_mutation(classifiers) ;
tri_mutation(classifiers) ;
}
```

Bi Mutation and Tri Mutation: As part of the GA process, a number of classifier chromosomes are mutated in both the action (bi-mutation) and condition (tri-mutation) positions. This is achieved by selecting a random classifier / position and altering the value at that position. This allows the classifier rule set to potentially backtrack from the issue of convergence.

```
void bin_mutation(CLASSIFIER classifiers[MAXCLASS])
{
int no_of_mutations     ,
    mutation_count = 0  ,
    random_classifier   ,
    random_action       ;
FILE *binmutation = fopen("bin_mutation.txt", "w") ;

no_of_mutations = (int) NPOSITION_ACTION * MAXCLASS * MUTATION_RATE;
printf("\nMutating action chromosomes...") ;
while(mutation_count < no_of_mutations)
   {
   random_classifier = get_random_value(MAXCLASS) ;
   random_action = get_random_value(NPOSITION_ACTION) ;
   switch(random_action)
      {
      case 0  :
            {
            classifiers[random_classifier].action.act_0 =
            ~classifiers[random_classifier].action.act_0 ;
            break ;
            }
      Code Repeated for cases 1 to 14 mutating at that position
      case 15  :
            {
            classifiers[random_classifier].action.act_15 =
            ~classifiers[random_classifier].action.act_15 ;
            break ;
            }
      }
   fprintf(binmutation,"Classifier No: %3d was mutated at action
position %2d\n", random_classifier, random_action) ;
   mutation_count++ ;
   }
fclose(binmutation) ;
}
```

```
void tri_mutation(CLASSIFIER classifiers[MAXCLASS])
{
int no_of_mutations     ,
    mutation_count = 0  ,
    random_classifier   ,
    random_condition    ;

FILE *tri_mutation = fopen("trimutate.txt", "w") ;

no_of_mutations = (int) NPOSITION_CONDITION * MAXCLASS *
MUTATION_RATE ;
printf("\nMutating condition chromosomes...") ;
while(mutation_count < no_of_mutations)
   {
   random_classifier = get_random_value(MAXCLASS) ;
   random_condition  = get_random_value(NPOSITION_CONDITION) ;
   switch(random_condition)
      {
      case  0  :
            {
            classifiers[random_classifier].condition.pos_0 = (rand()
            % 3) - 1 ;
            break ;
            }
      case  9  :
            {
            classifiers[random_classifier].condition.pos_9 = (rand()
            % 3) - 1 ;
            break ;
            }
      }
   classifiers[random_classifier].specificity =
   calculate_specificity(classifiers, random_classifier) ;
   mutation_count++ ;
   fprintf(tri_mutation,"Classifier No: %3d was mutated at condition
   position %2d - specificity = %2d\n", random_classifier,
   random_condition, classifiers[random_classifier].specificity) ;
   }
fclose(tri_mutation) ;
}
```