Determining the Feasibility of Achieving Graceful System Degradation Through The Use of Automatic Reconfigurable Designs and Software

Mark D Cornwall

30 September, 2006

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk
Determining the Feasibility of Achieving Graceful System Degradation Through The Use of Automatic Reconfigurable Designs and Software

Open University

M801

Mark D Cornwall (R0161105)

8-Feb-07

A dissertation submitted in partial fulfilment of the requirements for the Open University’s Master of Science in Computing for Commerce and Industry.

mark.cornwall1@btopenworld.com

07931 934635

Word Count (19,534)
Preface

This thesis makes reference on a number occasions to biological systems, in particular molecular genetics. While this thesis has placed some of these concepts in context of component software development, the breadth of the subject means that it can only be treated at a relatively high level. Consequently, software equivalents of 'messenger RNA' or 'RNA genes' were considered outside of the immediate scope, although they will undoubtedly making interesting research topics in their own right.

I would like to thank Andrew Fitzmaurice and John Connelly for their invaluable review comments.

Last, but always first in thought, I would like to thank my wife Karin, for both her patience in listening to my ideas, and tolerance while I was scribbling away.
Contents

Preface .................................................................................................................. 2
Contents ................................................................................................................ 3
List of Figures ........................................................................................................ 5
Abstract ................................................................................................................ 6
1 Problem Overview .............................................................................................. 8
   1.1 Problem Domain Description ..................................................................... 8
   1.2 Aim & Proposed Research ......................................................................... 11
      1.2.1 Aim .................................................................................................. 11
      1.2.2 Proposed Research .......................................................................... 11
2 Research Methods ............................................................................................. 13
   2.1 Literature Review ...................................................................................... 13
   2.2 Review Areas ............................................................................................ 15
   2.3 Peer Discussions ...................................................................................... 15
   2.4 Approach Limitations ............................................................................... 16
3 Review of Resilience Body of Knowledge ....................................................... 17
   3.1 Computer System Failure Context .......................................................... 18
   3.2 Failure Mode ‘Types’ ............................................................................... 20
      3.2.1 Physical Failures ............................................................................. 21
      3.2.2 Design Failures .............................................................................. 22
      3.2.3 Malicious Failures ......................................................................... 24
      3.2.4 Environmental Induced Failures ................................................... 25
   3.3 Consequences of Failure .......................................................................... 25
      3.3.1 Changing Environment ................................................................... 26
      3.3.2 Changing Resources ...................................................................... 26
      3.3.3 Degraded Service .......................................................................... 27
   3.4 Response to Failures ................................................................................ 28
      3.4.1 Fault Prediction .............................................................................. 29
      3.4.2 Fault Detection ............................................................................... 30
      3.4.3 Fault Isolation ................................................................................ 32
      3.4.4 Fault Recovery and Repair ............................................................. 33
      3.4.5 Degraded Service .......................................................................... 34
   3.5 Summary of Resilience Body of Knowledge ............................................. 35
4 Review of GA and Fractal Architecture Research ........................................... 36
   4.1 Alternative Architectures .......................................................................... 37
      4.1.1 Neuro-Fuzzy Based Architectures .................................................. 38
   4.2 Fractal Architecture .................................................................................. 39
      4.2.1 Cell Formation Requirements ........................................................ 39
      4.2.2 Fractal Cell Structure .................................................................... 41
      4.2.3 Fractal Interaction .......................................................................... 45
      4.2.4 Service Provision and Design ........................................................ 47
      4.2.5 Response to Failure ........................................................................ 48
      4.2.6 FPGAs within a Cell ....................................................................... 49
   4.3 Agent Based Operating Systems ............................................................... 50
   4.4 GA Based Operating Systems .................................................................. 52
      4.4.1 GA Properties ............................................................................... 52
      4.4.2 GA Cell Design .............................................................................. 54
      4.4.3 GA Cell Life Cycle ......................................................................... 56
      4.4.4 Cell ‘Bio-Diversity’ ........................................................................ 58
   4.5 GA and Fractal Failure Modes ................................................................. 60
   4.6 Summary of GA And Fractal Architecture Research ............................... 61
5  Review of GA Resilience in Fractal Architectures .......... 62
  5.1  GA Resilience .......................................................... 62
  5.1.1  Hardware & Software Partitioning .................................. 63
  5.1.2  Software Rejuvenation ................................................. 64
  5.1.3  Re-tasking And Changing Functionality ........................... 65
  5.1.4  Resource Allocation .................................................... 67
  5.1.5  Mutual Interaction ....................................................... 68
  5.2  Fractal Architecture Resilience ..................................... 70
  5.2.1  FPGAs ........................................................................ 70
  5.2.2  Self-Repair ................................................................. 71
  5.2.3  Cell Formation & Interaction ........................................... 73
  5.2.4  Dynamic Reconfiguration ................................................. 74
  5.3  GA Control in a Fractal Architecture ............................... 75
  5.3.1  Multiple Layers ........................................................... 75
  5.3.2  Service Level Agreements and Prioritisation ..................... 77
  5.3.3  System Functional Equivalency ......................................... 77
  5.3.4  Topological Responsiveness ............................................. 78
  5.4  Problem Space Summary ................................................. 78

6  Analysis ................................................................. 79
  6.1  Literature Review Effectiveness ..................................... 79
  6.1.1  Elementary Literature Review ........................................ 80
  6.1.2  Originality ................................................................. 81
  6.1.3  Evidence Quality ........................................................ 81
  6.2  Detailed Analysis ........................................................ 82
  6.2.1  Approach Advantages .................................................... 82
  6.2.2  Approach Disadvantages ............................................... 83
  6.2.3  Approach Comparison ................................................... 84
  6.3  Proposed System Map ...................................................... 85
  6.4  Analysis Summary ........................................................ 88

7  Conclusions ......................................................... 89
  7.1  Summary ................................................................. 89
  7.2  Additional Observations ................................................. 91
  7.3  Areas of Future Research .............................................. 93
  7.3.1  Suggested Tools and Devices ......................................... 94
  7.3.2  Proposed Research Programme ....................................... 95
  7.3.3  Additional Research Areas ............................................ 96

References ............................................................. 97
Index ................................................................. 107
List of Figures

Figure 1: Research Approach ................................................................. 14
Figure 2: Literature Review ................................................................. 17
Figure 3: Computer Architecture Design ............................................. 18
Figure 4: Computer Architecture (OU M881 Unit 3 Fig 1.2) .................. 19
Figure 5: Sample Distributed Network .................................................... 20
Figure 6: Computer Failure Modes ....................................................... 21
Figure 7: System Failure Consequences ................................................. 25
Figure 8: System Response To Failure .................................................. 28
Figure 9: Fractal Architecture Concept (Ryu & Young, 2003: Fig 1 Variation) .......... 39
Figure 10: Fractal Cell Structure (Ryu & Jung, 2003: Fig 5 Variation) .......... 42
Figure 11: Fractal Architecture Arrangement (Ryu & Jung, 2003: Fig 11 Variation) ..... 43
Figure 12: Fractal Interaction ................................................................. 45
Figure 13: GA Process (Beyer, 2001: Algorithm 1) .................................... 53
Figure 14: Mutation Rate ..................................................................... 53
Figure 15: Reproduction & Recombination ............................................. 54
Figure 16: GA String Sequence ............................................................. 55
Figure 17: GA Controlled Cell Life Cycle .............................................. 56
Figure 18: Operating System Fractal Functions ....................................... 62
Figure 19: Fractal Cell Re-Tasking & Reconfiguration ............................ 66
Figure 20: Fractal Architecture Considerations .................................... 70
Figure 21: SLA Equivalency ................................................................. 76
Figure 22: Architecture Comparisons .................................................... 76
Figure 23: Proposed System Cellular Architecture Meta-Map .................. 86
Figure 24: Proposed System Fractal Architecture Meta-Map ..................... 87
Figure 25: Proposed Future Research Programme .................................. 95
Abstract

All systems can and do fail, and often a relatively ‘small’ failure has a disproportionate effect, as it prevents access to parts of a system that are operating correctly, or prevents their usage. Hence, the traditional method of maintaining the intended service, is to provide sufficient redundancy, in combination with an appropriate repair strategy; although this is inherently expensive and is not always feasible.

However, design and economic pressures, together with engineering’s innate conservatism, effectively prevent this approach from being re-evaluated. The net result is, at best, a step drop in capability and at worse, a total loss of a system.

Allowing a system to undergo ‘graceful’ service degradation, instead of a step drop, significantly improves its ‘resilience’; with a direct impact on both availability and cost. A resilient system is therefore capable of absorbing the impact of an interruption, disruption or loss; and will continue to provide a minimum ‘acceptable’ level of service.

This thesis is a literature survey, and represents a synthesis of ideas from a number of sources to suggest an inherently resilient system. It presents a new way for looking at system failures that occur within a traditional computer-based architecture, by considering:

- An architecture based upon the ‘fractal concept’, i.e. a number of identical components, capable of self-repair, self-configuration, self-optimisation, and self-protection; which through mutual interaction, can provide high level services equivalent to that of a ‘normal’ computer;

- An operating system, which utilises the principles of evolutionary strategies in the form of genetic algorithms, to prioritise and reconfigure the components; which
form the pool of available resources for a given operating scenario and environmental circumstances.

By combining these two concepts, to create a ‘heterogeneous adaptive system’, this thesis outlines how the proposed approach creates an emergent self-optimising capability; which uses temporal based structures for causal elimination and automatic ‘work around’. This creates an inherently resilient system, capable of automatically adapting to internal and external environmental changes i.e. failures, while maintaining its intended service provisions.

This thesis consists of six sections: the introduction; research methods employed; current architecture resilience concepts and responses; the proposed fractal architecture; an examination of the proposed architecture’s ability to address resilience issues; an analysis of the method employed and the results identified; conclusions and future research.
1 Problem Overview

1.1 Problem Domain Description

All systems can and do fail, i.e., the non-performance or inability to perform the intended function for a specified time under specified environmental conditions, (Leveson, 1995). Failures can be:

- Physical, the device breaks through excessive or prolonged usage, (NCS, 1982);
- Environmentally induced, hardware and software can be affected by the physical environment e.g. radiation induced bit changes (Actel, 2002);
- Design errors, which affect both hardware and software (Perraju, 2001);
- ‘Malicious’, which are a subset of ‘design’ errors (M881, 2002).

Often, a relatively ‘small’ failure has a disproportionate effect by preventing ‘access’ to the remaining working parts, or prevents their effective use. The net result is typically a step drop in capability or total system loss. The failure’s significance is based upon its (rated) impact and frequency of occurrence (Camargo, 2001), i.e. its tolerability; consequently small regular outages, may have a greater economic impact than a ‘one off’. Failures often result in (a combination of):

- Safety impacts (people and / or the environment are placed at risk);
- Mission impacts (inability to perform intended function);
- Economic impacts (loss of work, delays resulting in lost revenue).
A truly ‘resilient’ system is therefore capable of absorbing the impact of an interruption, disruption or loss, and will continue to provide a minimum acceptable level of service (British Standards Institute, 2003). Hence the system can provide a gracefully degradable service, while reporting to users the extent of the situation.

Hence, how a system fails (and its ability to withstand those failures), are by necessity, important design characteristics for many safety critical and high availability systems (Reed, 2006). Failure avoidance or mitigation therefore requires an understanding of both ‘fault type’ and the design itself. Traditionally, this also involves an appropriate repair strategy; although this is inherently expensive e.g. stockpiling spares and is therefore not always feasible.

In addressing resilience based issues, the options typically considered in the literature, reflect the success of historic engineering approaches, and the economic compromises necessary to solve a given problem, e.g.:

- Availability of the system;
- Detection, isolation and ‘go around’ during operation;
- Better ways of determining failure probability for a given system;
- Avoiding design ‘type’ failures, (especially in software given the general high level of physical availability).

As with any ‘assumption’, these ‘accepted options’ should be tested regularly to determine whether they remain valid; however, design and economic pressures, together with engineering’s inherent conservatism, normally prevents this from occurring.
Hence, some aspects of a system’s design are very rarely questioned, and are generally accepted as a ‘given’.

Consequently, only limited attempts at redesigning the fundamental basic concepts have occurred, effectively precluding consideration of:

- Autonomic response to faults and errors (Xu, 2003);
- Self-repair in response to fault identification (Bower, 2005; and Lala, 2005);
- Probabilistic fault anticipation and response (Cai, 1996);
- Dynamic functional partitioning (Fröhlich, 2001);
- Fault treatment granularity (Bobbio, 2001).

Two other related areas that are not well addressed by the literature, but impact system degradation, are re-tasking and reconfiguration:

- Re-tasking is the prioritisation of various aspects of the service provision based upon available resources. While part of normal computing, especially in distributed systems, in this context, re-tasking refers to the re-allocation of physical functions within the system. Consequently, a failure within one area can be counteracted by using alternative ‘parts’, to provide the same function;
- Re-designating component functionality through the physical reconfiguration of resources significantly enhances re-tasking. Hence reconfiguration provides an inherent resilience capability; however, the various components need to determine when, how, and into what they should be reconfigured.

Closely allied to re-tasking and reconfiguration, is the ability to repair components, when both the potential problem and the available resources are initially unknown.
1.2 Aim & Proposed Research

1.2.1 Aim

The aim of this thesis is to demonstrate that a computer’s fundamental design is not conducive to graceful degradation; and that an automatic configurable design, capable of continuously responding to adverse internal and external environment changes is more robust, while providing the same intended service level for longer.

1.2.2 Proposed Research

The proposed research is split into three parts, and will consider:

• ‘Design’ as a fundamental contributor to overall failure;

• Development of a ‘fractal based’ architectural solution composed of Field Programmable Gate Arrays (FPGA) and ‘double helix’ Genetic Algorithms (GA).

(Throughout the document, ‘system’ may refer to either a single ‘computer’ or multiple computers linked together as a ‘system of systems’);

• GAs operating in fractal architecture in a resilience context.

Williams (1997), defined a fractal as “a pattern that repeats the same design and detail of definition over a broad range of scale.”

A ‘fractal-based’ architecture therefore, consists of a number of functions repeated at different ‘hierarchical’ levels (Tharumarajah, 2003). At the lowest level, this consists of a number of identical ‘cells’, which through mutual interaction, are capable of providing the functionality of the next higher level of service.

While “GAs are a group of stochastic search algorithms inspired by evolutionary biology….the GA searches a solution space defined by the representation of the
application at hand, by iterating three basic steps...evaluation of a group of search points, called the population, against an objective function...based upon their evaluated fitness, some are selected as promising candidate solutions...which undergo genetic operations to generate a new population." (Eklund, 2004).

Focusing on several inter-related areas, the research will demonstrate how:

- Physical and design induced failures impact system operation (Actel, 2002);
- Systems can respond to environment and resource changes, using reconfiguration and re-tasking management (Oh, 2005; Sklyarov, 2002);
- Alternative architectures can provide ‘service equivalency’ through re-formation and mutual interaction (Ryu, 2003; Younis, 2004);
- FPGAs dynamic reconfiguration capacity can provide different functionality; (Sterritt, 2004).

Using this research to create a synthesised approach based upon disparate ideas, the thesis will identify and assess the effectiveness of several alternative solutions. It will show how the proposed approach addresses the issues presented by traditional methods; and that it is inherently resilient, by looking at the different types of failure and their consequences. For example, the solution has a number of apparent inherent advantages when compared to a ‘traditional’ design, such as a dynamic self-organisation; allowing it to implement actions based upon available resources and environmental circumstances, (Levitin, 2005; Gen, 2001).

Hence to demonstrate the aim, the proposed research question is therefore:

"Is it possible for a fractal based system to achieve graceful degradation; i.e. continue to provide some services in a prioritised order, reducing its commitments according to a previously agreed priority of functionality?"
2 Research Methods

The primary research method employed within this thesis is a keyword search to identify suitable journal articles and case studies. The suitability of the research subject is demonstrated by the paucity of published literature.

2.1 Literature Review

The literature review supports the approach outlined in Figure 1; while direct examples may be identified by the review in support of the thesis, the majority of the concepts will require a synthesis of ideas; and the extrapolation of ideas from one area of research into another. This has necessitated an extensive (automated) search across a number of research areas utilising keyword combinations, these have included:

- Adaptive
- Algorithm
- Autonomic
- Autonomous
- Availability
- Bionic
- Degradation
- Dependability
- Dynamic
- Evolutionary
- Failure
- FPGA
- Fractal
- Genetic
- Heterogeneous
- Hierarchy
- Holonic
- Polymorphic
- Reconfiguration
- Reliability
- Resilience
- Scalable
- Self-stabilisation
- Shared
- Topology

The act of conducting the search has itself, suggest additional areas and keywords for further examination. Subsequent sifting and cutting of information, has enabled concepts to be clarified, linked, and subsequently reinforced.
The review has only been able to assess publicly available information; however, given the potential economic impacts that could result from different aspects of the proposed research, it is recognised that many areas may be addressed by as yet unpublished R&D programmes.

Given the speed with which technology changes, the search has been limited to the more recent papers to provide relative data freshness; although cognisance has been taken of older papers where appropriate. Also, an examination of the relevant citations within selected papers has been undertaken (OU, 2005).

In addition, appropriate industrial standards and text books have also been reviewed for applicable techniques and concepts. Standards represent a good source of current consolidated information, i.e. they have been updated or are discarded.
Similarly, textbooks address many standard issues and concepts within a given domain, although they are not as recent as the research papers.

### 2.2 Review Areas

In order to demonstrate the proposed approach, the thesis has examined aspects of a ‘traditional’ system in a logical sequence, and contrasted them with fractal cell and architecture design equivalents. Consequently, the review builds a logical argument for demonstrating the feasibility of achieving graceful system function degradation, through the automatic reconfiguration of components, software, and architecture; the review has therefore considered:

- Overall concept – nearest equivalent (holonic, bionic, fractal);
- FPGA properties, including self healing and dynamic reconfiguration;
- Cell formation for producing higher level functions;
- GA’s, and their properties;
- Using GAs as a resource manager to control and task the FPGA cells through reconfiguration;
- Using GA based operating systems to provide a ‘level of service’ equivalence;
- Explaining how the approach can address the various failures;
- GA prioritisation of the overall service in light of various failures;
- Problems with the potential design – theoretical and practical.

### 2.3 Peer Discussions

There are a number of inherent limitations with a literature survey based thesis, e.g. currency of information. Hence, in order to corroborate the approach, the thesis was discussed with a number of researchers from the Software Engineering Institute
during the European Software and Systems Engineering Process Group annual meeting (ESPI, 2006).

This was undertaken in the context of requirements effectiveness and the development of ‘system of systems’, where operational control is shared between a number of entities who are not necessarily known to each other. It was agreed that the proposed approach had merit and that it addressed many issues; although the consensus was that successful delivery required a further five years of research.

### 2.4 Approach Limitations

By definition a literature survey is based upon existing work, which in turn reflects research ‘fashion’, political dictates, and funding requirements. Consequently, it may not be possible to achieve a fully balanced or objective assessment of the proposed idea; unless alternative architectures and operating system approaches are also considered.

The other main limitation with a ‘pure’ survey is that results are inevitably qualitative in nature. While this can be addressed in part by the provision of a proposed system map; this issue was recognised during the early course of this study. As such, the decision was taken to make this thesis sufficiently robust, that it could be used as a substantive ‘business case’ for undertaking future PhD research. Hence, ‘Future Research’ discusses both generic research and the initial next stages for validating the proposed architectural solution.
3 Review of Resilience Body of Knowledge

This Section demonstrates how the computer’s fundamental design is a major contributor to a lack of graceful degradation, Section 1.2.

The literature review itself consists of three main sections with structured links, to demonstrate the proposed approach:

- While Section 1 addressed the nature of the problem; this part of the review looks broadly across the current body of knowledge, and highlights various aspects of the present research undertaken in the area of computer architecture resilience; see Figure 2. Given the breadth of the overall subject area, citations are used in this Section to support the main argument’s theme. Hence, each citation represents either an example of a particular problem or it addresses a specific issue, often in part, which is relevant to the main theme. Where there are multiple sources with respect to a single aspect, they have been summarised;

- Section 4 considers a possible solution, using a more limited number of citations. This involves the weaving together of several key concepts regarding fractal architectures, GA’s and reconfigurable components;

Figure 2: Literature Review
• Section 5 looks at how the proposed solution addresses the issues raised during
  the search of the current body of knowledge. In addition to cross-references,
citations are used for emphasis.

By using such an approach, it is the intention that the synthesis of ideas should
progressively demonstrate its own viability.

3.1 Computer System Failure Context

With the exception of some types of specialised complex architectures, the basic
structure of the majority of computing systems, has not fundamentally changed in
concept since the work of Von Neumann (M881, 2002), Figure 3. This reflects in
part, engineering’s innate conservatism.

*Engineering conservatism means using something that has worked (well) in
the past; however, this also results in the design unwittingly inheriting its
constraints and assumptions.*

Hence, although
technological changes,
such as replacing
valves with IC’s, have
improved overall
availability; the ‘basic
architecture’ has
effectively remained the
same, Figure 4.
Typically, each architectural component provides a specific function, either singularly or in combination, which are integrated at various levels of abstraction, until a user service is defined (M881, 2002).

On a larger scale, these and additional functions may be provided by various systems, all in turn, requiring assorted forms of management (M881, 2002: T823, 2004). This approach often means that a single failure can result in the loss of a whole function e.g. access to memory, and therefore effectively the entire system.

A relatively small failure can therefore have a disproportionate effect upon overall (operational) resilience.

Furthermore, as in many areas, engineering is subjected to opportunity and design fashions. For example, the ability to connect within a network has facilitated the distribution of computer functions themselves.
This creates additional design problems (and opportunities) such as deadlocks; depending on whether the architecture is open or closed, forms part of a distributed network or not, is well or ill-defined (M881, 2002; T882, 2003; Liu and Chen, 2004; Avresky and Natchev, 2005).

Hence, from a failure contextual perspective, simply treating a network as a ‘system of systems’; means that a node develops the ‘characteristics’ previously attributed to a component (as identified in Figure 4); e.g. in Figure 5, the loss of the A-B link means part of the network is isolated and will longer able communicate. Similarly, a failure of the A-C link means the A-D link has to undertake twice as much work, leading to potential congestion issues etc, (T823, 2004; Duato et al, 2005).

### 3.2 Failure Mode ‘Types’

Failure modes are discussed within the literature quantitatively at a generic level, using various statistical techniques; or at a lower level, where specific failures and consequences are discussed, (Perraju, 2001; Carmargo et al, 2001; Littlewood et al, 2002; Lai et al, 2002). The literature indicates that failures can be broadly divided into four categories; Section 1 and Figure 6:
Figure 6: Computer Failure Modes

- **Physical**: The device fails through poor material quality, heavy or excessive usage, or simply old age, (NCS, 1982);

- **Design**: various compromises for good (and bad) reasons, or poor understanding; result in the component / system behaving in an unexpected manner (Zhang and Pham, 2000; Khang and Sung 2002);

- **Malicious**: the system is subjected to malicious intent (attack), causing it to act in an inappropriate manner, (Madan et al, 2004);

- **Environment**: the environment itself causes a device to physically breakdown or respond inappropriately, (Actel, 2002).

### 3.2.1 Physical Failures

Physical failures are arguably the easiest to detect, and the history of computer design reflects this; e.g. ‘plug and play’ is an extension of value rack replacement.
Fault conditions are often set during manufacturing, e.g. diffusion faults or glassivation defects, which result in failures occurring during normal operation (early burn out) or through long-term degeneration; (NSC 1982; Sze, 1988; Grovenor, 1989; Bower et al, 2005).

Hence, the collection of defect data, and relating it to component environmental stress, has provided sufficient information to allow statistical analysis; ensuring that failure mechanisms are well understood. This has resulted in standards and guidance documents such as US Military-Standard 470 and Military-Handbook 217, being developed (Smith 1988).

More recent, non-process orientated literature, has focused on availability issues associated with large complex systems, or those providing specialist functions, (Perraju, 2001; Reed and Mendes, 2006). Hence, a suitable architectural design (and appropriate quality policy) can cater to the majority of physical failures.

3.2.2 Design Failures

Design trends, reflecting opportunity and customer demand, have moved away from discrete i.e. physically independent functions, to a more integrated approach. In part this reflects softwares' ability to provide greater flexibility than hardware; but also a reduction in overall cost coupled with increased capacity (i.e. ‘Moores’ Law'; McGraw Hill online encyclopaedia, accessed 14/05/06). This is shown in the relevant literature by an increased focus on design failures.

‘Failures’ typically occur when the system has exceeded a tolerance parameter; or something has occurred, which was never originally considered when the design was initially envisaged. Consequently, design failures may be split into two rough groups: those that look at the overall system design, and those that more specifically address software development. System design issues cover a number of areas ranging from trade-off considerations (Younis et al, 2004); hardware and software component
interactions (Lai et al 2002); to the general availability of various modules (Carmargo, 2001; Reed, et al 2006; Park and Kim, 2002). The increased prevalence of software at the system level and its contribution to these types of failure, is also reflected within the literature (Levitin, 2005; Yeung and Schneider, 2005; Özekici and Soyer, 2003).

In the literature, design ‘robustness’ is considered quantitatively through improving the design (and testing) processes, or by removing errors through alternative design comparisons and their selection during operation, (Levitin, 2005; Yeung and Schneider, 2005; Zalewski et al, 2003; Zhang and Pham 2000). The impact upon performance, especially in large complex software based systems, also represents a major area of research (Lai et al, 2002; Goševa-Popstojanova and Trivedi, 2001; Perraju, 2001).

Software does not ‘break’ in the same sense as hardware; rather errors accumulate such that tolerances are exceeded, and a ‘design’ failure occurs. While it is possible to physically induce software errors (e.g. cosmic radiation or alpha particles from device packaging; Acetel, 2002), the nature of software means that the majority are introduced through the design process accidentally or via poor specification. The net result is a ‘fragile’ product, having narrow tolerance limits on its operating parameters and environment. Furthermore, given the expense and difficulty associated with validating and verifying software performance, not least because it can be undertaken inappropriately and induce errors; the literature also considers the reuse of tried and tested software, (Younis et al, 2004).

Also software can have a product life measured in decades, and can be subjected to a number of changes over that period. Each time software is changed, including attempts to fix previous problems, new (undetected) errors are introduced. In addition, during use, slight ‘operational environmental changes’ occur over time,
which were not originally anticipated during the design phase. As a consequence, error accumulation occurs during operation, a process known as ‘software aging’ which has to be addressed either by a human or system controller (Özekici and Soyer 2003; Liu et al, 2005, Pfening et al, 1996). While there are always design trade-offs, an inadequate designed architecture is less able to respond effectively; and may actually induce failures under specific circumstances.

### 3.2.3 Malicious Failures

Although often considered separately by the literature, a specialised form of design failure (typically inadequate specification) occurs when the system is compromised (attacked), i.e. a known (architectural) weakness is exploited; effectively resulting in a ‘malicious’ failure. Such incidents are classified depending upon whether the focus is on data or some form of operating system implementation flaw (Garfinkel and Spafford 1997). Ultimately, the attackers’ intent is to induce information disclosure, compromise data integrity, prevent resource utilisation (denial of use) or repudiate an action (M881, 2002; M886, 2004).

Consequently, the design needs to consider effective security, which involves multiple layers of prevention, detection, protection etc (M886, 2004). Within the system itself, there are a number specific mechanisms such as access control, traffic padding etc which are applied at various layers within the OSI model to mitigate the effects of an attack (M881, 2002; Stallings, 2003). As a potential attacker, their resources, and motives, can range from a hostile government to that of a small child (M886, 2004), the literature in this area is quite extensive; covering such subjects as cryptographic techniques to adaptive secure software (Stallings, 2003; Tak and Park, 2003).
3.2.4 Environmental Induced Failures

While devices may be subjected to excessive environmental stress and so accelerate fault occurrence, Section 3.2.1; some unique environments may induce system failures. Typical examples include those that contain hard ration, such as inside a nuclear reactor or upon a satellite, (Actal, 2002). These often represent a combined physical and or design failure.

3.3 Consequences of Failure

Although various classification systems, methodologies, and failure consequences, have been extensively categorised within the literature, from a global perspective, failure impact can be simply classified, as a change to the operating environment, the available resource, or the level of service provided; Figure 7. However, the extent of any impact is conditional upon what occurs, where, and when, i.e. it is architecturally dependent.

Figure 7: System Failure Consequences.
3.3.1 Changing Environment

The definition of what constitutes a ‘system’, and therefore its operating environment, is somewhat arbitrary, often this reflects a designer’s decision (T837, 2003).

A failure impacts how a system interacts with both its ‘internal’ and ‘external’ operating environments, and is therefore dependent upon the architectures various boundaries and interfaces.

A failure can affect the ‘external’ environment through incorrect operation or the absence (or permanence) of a function. Similarly, as the internal operating environment is subjected to the same types of failure, particular capabilities may no longer be available in whole, or in part.

3.3.2 Changing Resources

Another consequence of failure is that available resources for performing existing or future services will have changed, Section 1.2.

Although often thought simply as a physical entity to which there is limited access, e.g. a printer; a wider definition of resources is typically used in (distributed) systems:

- Spatial;
- Temporal;
- Data;
- A combination of all the above.
Hence, depending on where or how a failure occurs, it can effectively render the remaining undamaged areas, as either inaccessible or incapable of operation by the rest of the system.

However, due to design economies, very few systems have the appropriate architectural capabilities (i.e. redundancy) to address this situation; rather, they reflect optimisation considerations for ‘normal’ operations, (T822, 2003; T823, 2004). As a consequence, unless sufficient spare capacity is available at the appropriate location and time (many functions are often ‘one’ deep), then a relatively small failure can have a disproportionate impact.

3.3.3 Degraded Service

Degraded service is also a potential consequence of failure.

Degradation occurs when the operational environment is adversely impacted, or resources are no longer available in sufficient quantity for the service to be provided at the appropriate level, i.e. only an intermittent or reduced service is available. Hence, the extent of degradation is dependent upon fault location and therefore, the system’s architecture.

Consequently, service impacts can be typically categorised as:

- ‘No impact’;
- ‘Minor’ - nominal degradation;
- ‘Major’ - significant degradation;
- ‘Catastrophic’ - total loss.
As the number of failures increases, a system’s performance is often degraded disproportionately, Section 3.2.2.

This reflects failure ‘criticality’, and the cumulative nature with respect to disproportionate workloads and associated increased management overheads; e.g. “system calls scale poorly when the number of connections is significantly increased” (Liu and Chen, 2004).

Hence, size and complexity have a major cost impact on maintenance, and it has been proposed that ‘systems’ should be capable of identifying problems and healing themselves, (Whiteson and Stone, 2004), Section 3.2.2.

3.4 Response to Failures

In responding to the various types of failure and their consequences, the literature looks at making a system ‘fault tolerant’ through a combination of: prediction, detection, isolation, recovery, and restoration, Section 1.1 and Figure 8.

Figure 8: System Response To Failure
‘Fault tolerance’ is typically defined as “the ability of a system to continue satisfactory operation in the presence of one or more non-simultaneously occurring hardware or software faults” (Spitzer, 1993).

Fault tolerance requires a robust design, developed using such techniques as hardware and software redundancy (i.e. contingency); albeit there is a trade-off with cost and performance. This allows either the same level of service, or a degraded capability based upon some form of pre-selected criticality, to be provided. Both design processes and operational trade-off considerations are discussed extensively, (Gutjahar and Uchida, 2000; Perraju, 2001; Levitin, 2005; Zhang and Pham, 2000; Özekici and Soyer 2003; Reed and Mendes 2006; Zalewski et al, 2003; Yeung and Schneider, 2005).

Hence, fault tolerance is dependent upon the system boundaries, available resource, and its design; i.e. its architecture (physical, functional and logical).

3.4.1 Fault Prediction

As mentioned in Section 3.2.1, statistical analysis has been used to predict failures based upon component stressing and probability distribution, (Smith, 1988). This consequently influences system design, in terms of component redundancy; repair cycles / maintenance down time etc.

Assessing failure impacts has resulted in various methods for calculating system reliability (Chang et al, 2004), in particular, attempts to predict where and how failures will occur; have been made in large complex systems or those that are heavily software biased, (Chang et al, 2004; Camargo et al, 2001).
Real-time prediction of potential failures based on current environmental and operational conditions, would enable the system to anticipate situations, and respond accordingly (Reed et al, 2006; Xu et al, 2003).

Such capabilities, where they are discussed within the literature, are generally based upon fuzzy logic, which enables a ‘maybe’ option to be considered; instead of a simple hard ‘yes’ or ‘no’. Consequently, “vague phenomenon, vague relations in the modelling of problems….information vagueness….heuristic algorithms..” can be used to predict fault occurrence (Cai, 1996). While there are obvious advantages to prediction over detection, from an operational perspective, its use is currently limited, e.g. to avionic ‘health and usage monitoring’.

3.4.2 Fault Detection

System fault response is dependent upon (correct) failure detection (Rish et al, 2005; Krantis et al, 2005).

Hence, detection typically involves some form of management system, which is responsible for monitoring performance and undertaking action if a ‘threshold’ is exceeded (T823, 2004).

Detection and trade-off consequences are well documented (Spitzer, 1993; Leveson, 1995), and are typically split to reflect the fault’s ‘origin’, i.e. hardware, software or during transmission:

1. Hardware failure detection typically involves:

   - Replication and voting: Here, a fault tolerant ‘voting’ circuit compares the (same) output values from multiple systems, and discards that one which does not agree with the majority decision;
• **Duplication and comparison:** Typically this involves regular cross comparisons between two circuits, each having a capability (under specific circumstances) to de-activate the other (incorrect) system. Often there is some form of Master / Slave / Back-up arrangement in place;

• **Self-checking:** Reasonableness checks on intermediate and or final results, can detect internal errors, without reference to another system. If detected, the system can simply switch itself off and or activate an alternative;

• **Built in test:** As opposed to a self-check, an ‘externally’ requested check using either a dedicated testing sub-system and or software within the main unit, detects the error.

2. Although using similar concepts to those employed in hardware design, (Spitzer, 1993; Levitin, 2005), software failure detection typically reflects design error origins, Section 3.2.2:

• **Multi-version (N-version) programming:** Multiple software versions are developed by separate teams, which are run either in parallel or sequentially; and undertake a number of synchronised comparisons at specified points;

• **Recovery Blocks:** One version of the software is executed, and if it is ‘non-complaint’, an alterative version is invoked. This new version undergoes the same test and if it passes, the previous version (or part) is deemed to have failed. Recovery can either involve re-running the failed module (backward recovery), or ignoring the failed responses and only taking the correct values from the invoked module (forward recovery);

• **Exception Handling:** The system only reports situations which deviate from the expected standards – an ‘exception’ - while results falling within expectations are not.
3. Transmission failures may not necessarily be due to system software or hardware fault e.g. dropped packets due to system timeout or corruption. Hence, a combination of hardware and software design techniques is employed to detect transmission errors, e.g. checksums, (T822, 2003; T823, 2004).

Many (safety critical) systems require extensive testing to identify the problem source. While many of the above techniques are well established, demonstrating their effectiveness remains complex (Littlewood et al, 2002). This is further complicated when distributed systems or high data volumes need to be assessed. Hence, new approaches are being identified, such as proactive fault diagnosis using graph theory (Yu et al, 2005), or ‘adaptive selective active probing’, to determine the fault source (Rish et al, 2005).

While the more ‘obvious’ faults are often detected quickly, disproportionate amounts of resources are often required to identify those that are more obscure or occur over a longer period. Hence, large amounts of research effort are being spent looking at the design process, to identify where and how failures are introduced, (Neema et al, 2004; de Lemos, 2004).

### 3.4.3 Fault Isolation

In many fault scenarios, a loss is preferable to an incorrect output; as the latter can propagate throughout the remainder of the system inducing further errors and spurious responses, creating a potential cascade situation, Section 1.1.

Therefore once detected, faults are often isolated or deactivated, with the remainder (of the system) informed of the problem (Yu et al, 2005). “Currently, static reconfiguration techniques with predefined alternative paths, based on redundant hardware, are used in many contemporary high-speed networks” (Avresky and Natchev, 2005), to perform this function. However, isolation may induce other
failures, e.g. deadlock or livelock, as additional dependencies are created prior to old link removal; (T823, 2004; Avresky and Natchev, 2005).

Unfortunately, fault isolation may also require other potentially ‘good’ systems to be isolated; e.g. due to connectivity issues, or the inability of various systems to resolve mutual dependencies (Kon et al, 2005), Section 3.1.

Hence, “reconfiguration and anonymity of the active elements to the user are considered basic attributes of advanced fault tolerant systems”, (Spitzer, 1993).

3.4.4 Fault Recovery and Repair

Recovery is the ability of a system to return to the same level of service or capability after a failure has occurred; while, repair is the ability to ‘self-heal’ or have a faulty part replaced.

If possible, the system may attempt to recover from the fault, either working around using ‘alternatives routes’; restarting; using back-up components; or calling an engineer to replace the faulty part; (Bobbio, 2001), Section 1.1. Of course, a single severe failure may cause the entire system to fail outright. In many instances, the system can recover from the last known good state or undertake a complete reboot (M881, 2002; Van Moorsel and Wolter, 2004). Ideally, it should recover to a legal state, without the aid of human intervention (Datta et al, 2004).

Unlike biological systems, computers typically do not possess sufficient granularity of design, or function, to effect true self-repair; rather they simply implement a ‘lower level of available redundancy’ (Lala et al, 2005; Bower et al, 2005). Hence, true repair of a physically damaged device requires human intervention.
The main methods employed for recovery (and repair) are discussed above; however, due to inherent computer design limitations, two mechanisms which are not readily scalable at the system level, are:

- **Automatic re-assigning of functionality between hardware and software** (Finc and Zemva, 2005);
- **Dynamic system reconfiguration, as typified by a FPGA** (Zhang and Ng, 2000; Lala et al, 2005).

An alternative mechanism to repair and recovery, is simulation of the damaged part; e.g. system software may ‘recreate’ a correct output using the remaining available data and others in close proximity which are still functioning; a process known as analytical redundancy (Spitzer, 1993; Clouqueur et al, 2004).

### 3.4.5 Degraded Service

A valid failure response to a failure is to change the level of service provided, i.e. degrade it. Services are often prioritised to reflect ‘criticality’, and therefore provide a natural order for their preferred loss. Furthermore, the level offered may be reduced in line with existing agreements (T823, 2004); reflecting their use or ‘abuse’ as appropriate.

Although briefly mentioned in Section 1, there are a number of service prioritisation schemes reflecting fault consequences or tolerability; e.g. ‘catastrophic’ (SAE, 1996; MoD, 2004; Railtrack, 2000). Also, as service boundaries are often ‘grey’ with a number of overlaps, an acceptable level of service is conditional upon the operational scenario; e.g. peace or war time (MoD, 2004). In general, services are prioritised as follows, (most important first):
Safety Critical: a ‘loss’ would result in death, a large number of injuries, or a major environmental disaster;

Mission Critical: the system or business would be unable to undertake its primary intent, e.g. loss of weapon targeting on a warplane;

Business Critical: those failures which have an impact upon an organisation but not directly its mission, e.g. a company payroll system;

‘Other’: those which do not fall into the above categories i.e. are ‘nice to have’.

3.5 Summary of Resilience Body of Knowledge

This Section has looked broadly across the current body of knowledge, highlighting various aspects of the current research in computer architecture resilience:

- Basic computer architecture;
- Different types of system failure;
- Associated consequences;
- Typical design response.

It was established that when a failure occurs, the ability to respond is architecturally dependent, resulting in dedicated functions, and therefore capability, being lost; which can not necessarily be re-introduced into the system, Section 1.2.2.
4 Review of GA and Fractal Architecture Research

Section 3, determined that failure response is governed by the system’s architecture, which has not changed essentially since computers were initially developed. Current designs still rely on the same approach, albeit packaged differently, e.g. ‘having two of something’. Hence, any proposed solution has to be cognoscente of those architectural issues, as well as adapting to environmental and resource changes.

The available literature typically considers isolated issues together with ‘immediate’ solutions; however, similar concerns in non-related fields (and potential solutions) are often over looked. Consequently, this Section considers alternative architecture and differing forms of operating system, in addition to a potential solution to the issues raised by Section 3; by weaving together several diverse key concepts which exhibit phylogenetic, ontogenetic and epigenetic behaviours, (Lala et al, 2005):

- FPGAs, which have specific properties to enable them to be reconfigured;
- A ‘fractal architecture’ that uses arrangements of ‘cells’, which are temporally grouped to provide resource functionality;
- Darwinian selection in the form of GAs, which create an adaptive service provision through emergent behaviours.

Hence, this Section will consider the feasibility of using a fractal architecture, based upon FPGA cells, to provide an equivalent level of service of a ‘normal’ computer. It will also show how, in responding to a failure, a GA based ‘operating system’, can reconfigure the fractal architecture based upon circumstances and available resources.
4.1 Alternative Architectures

There are a number of possible architectures, which do not use the traditional hierarchical (command based) control approach, Section 1.2.2:

- A ‘bionic’ system is considered as an ‘organ’, and the equipment or components in the system as ‘cells’, using a biological viewpoint;
- A ‘holonic’ system consists of autonomous and cooperative entities defined by functional decomposition using object orientated concepts;
- A ‘fractal’ system uses a set of ‘self-similar agents’, which through cooperation, coordination, and negotiation with others, can reorganise and reconfiguration itself to be more efficient and effective.

Tharumarajah (1996) and subsequently Ryu and Jung (2003) compared fractal systems with those based upon traditional control hierarchy, bionic and holonic approaches. The basic principles of fractal, bionic and holonic systems are the same; utilising autonomous and dynamic entities to achieve their aims. However, major difference occur with respect to the negotiation schemes employed (to obtain resources and provide services), and their approach to reconfiguration.

Although the Authors undertook these reviews from a manufacturing perspective; the principles are transposable to computer systems, e.g. storage.

Ryu and Young identified the advantages of fractals over the other architectural systems:

- Their ability to dynamically change at any time, according to individual cell and system goals;
• New fractals can be constructed from available resources or re-assigning functions to existing fractals;

• Fractals use mobile agents, which reduces communication loads and maximises the ability of an agent.

While a fractal-based approach provides an ability to respond to environmental changes, this can be further enhanced if they are hybridised, and are constructed ‘bionically’.

That is, each component within a cell, “is basically similar, but differentiated by function, and are capable of multiple operations. These components are built up in hierarchical layers and are combined to form ‘organs’ with a particular function, which are in turn grouped together to form systems”, (Tharumarajah et al, 1996).

4.1.1 Neuro-Fuzzy Based Architectures

In addition to the architectures described by Tharumarajah, another major approach is based upon neuro-fuzzy logic. The advantage of fuzzy systems, is their ability to deal with non-binary responses, i.e. they have a ‘maybe’ option. Hence, as many systems utilise sensor inputs as part of their operation, and the ability to anticipate potential (internal and external) environmental change offered by a neuro-fuzzy interface offers significant advantages. However, while a neuro-fuzzy network approach would readily cope with external environmental variations, it would still experience internal failures, compromising ongoing operations, Sections 3.2 and 3.3. Hence, the architecture would need to be capable of some form of adaptation and regeneration (Glackin et al, 2004), i.e. was fractal based.
4.2 Fractal Architecture

This thesis proposes that a fractal-based architecture can address the issues identified in Section 3. This architecture consists of a number of functions repeated at different ‘hierarchical’ levels (Tharumarajah, 2003). Each function is provided by a number of identical ‘cells’, which through mutual interaction, are capable of providing the functionality of the next higher level of service; see Figure 9. Hence, to understand the proposed solution, it is necessary to understand cell formation, structure, interactions, and service provision; as well as responses to failure.

Figure 9: Fractal Architecture Concept (Ryu & Young, 2003: Fig 1 Variation)

4.2.1 Cell Formation Requirements

The ability to respond to change has traditionally relied on the designer incorporating sufficient (and potentially costly) contingency, (Section 3.3). Furthermore, most systems rely on stability and repeatability to operate efficiently; hence “apart from a few simple cases, command-based hierarchy can not solve the adaptation problems with which it is confronted in complex situations”. The alternative, especially in an
unpredictable environment, is for minimum specification and a capacity to redefine the systems’ structure, i.e. a fractal (Tharumarajah et al, 2003).

This in turn requires architectural building blocks, i.e. cells that are scalable and reusable. Furthermore, at this ‘cellular’ level, there is a requirement for mutual inherent problem recognition, and some form of response mechanism.

This necessitates double loop learning, (Tharumarajah et al, 2003), i.e. continuous improvement, where the cells self-diagnose a situation, taking action that will revise operational norms.

Tharumarajah et al, stated that a fractal and therefore its constituent cells, should exhibit autonomy, ‘circularity’, and ‘self-reference’; to enable their interaction to create double loop learning:

- Autonomy is a condition or quality of self determination, i.e. an ability to choose what to think and what to do. An ‘autonomous’ system therefore has some ability to alter its preferences and so affect its own actions;

- Cooperation amongst components is essential for self-organisation, this is achieved through self-control and coordination of (their) action with other units i.e. circularity; e.g. perpetuation of minor irregularities such as timing errors, can cause an entire system to crash, if they are not adequately controlled. Furthermore, “different rates at which events occur inhibit direct interactions between processes at different levels”. However, exercising self-control and self-coordination between components, ensures that these issues are ‘ironed out’;

- A ‘living system’ closes in on itself to maintain stable patterns of relationships, and it is this pattern of ‘self-reference’ which ultimately determines the system.

“Interactions among the entities both determine their actions and in turn, are
determined by the actions of the entities. When these interactions don’t leave the system, then the system evolves as a whole set of relations”…..the system (i.e. the society of entities) “interacts in a way that facilitates its organisation to maintain autonomy and unit, and in this sense the environment is actually part of the system and not separate to it”. (i.e. epigenetic behaviour, Lala et al, 2005)

Hence, a unit is adaptive, as its internal capabilities maintain its coherence. Therefore, these properties enable cells to (re)produce themselves, their own organisation, and identity. This necessitates (the appearance of) a hierarchy between fractal processes, which should be nested as ‘part – whole relationships’, ensuring that the full range of interaction is possible. Also, some form of ‘unity of action’ is required due to the level of internal interaction between cells; as there is a need to avoid (self-destructive) actions that are hazardous to the whole.

**Consequently, fractal formation involves the re-arrangement and reconfiguration (i.e. self-regulation) of cells through the exchange of resources and functions. This ensures harmony with other units, allowing the system functions to be maintained or changed as a whole (Tharumaražah et al 2003).**

Although Tharumaražah et al, used the term ‘autopoiesis’ to describe this situation, i.e. self production through mutual influences, each cell within a fractal contains minor variations. Furthermore, the fractal has a GA operating component capable of self-generated change. Hence, a fractal is actually a heterogeneous complex adaptive system, rather than a true homogeneous autopoitic system, (Stacy, 2003).

### 4.2.2 Fractal Cell Structure

As fractals can represent both a part and an ‘entire’ system, the definition provided by Tharumaražah et al and Williams, Section 1.2; require expanding to facilitate effective formation requirements, and hence cell structure. Ryu and Jung (2003)
provided the following definition, and addressed Tharumarajah’s (2003) requirements, by proposing that a fractal should consist of five ‘modules’ using facilitated coordination, see Figure 10, (Figure 11, shows sub-module components in architectural format):

Figure 10: Fractal Cell Structure (Ryu & Jung, 2003: Fig 5 Variation)

- Self-similar, providing services;
- Self-organising in terms of its operation (procedural optimisation); technically and strategically, (it determines and formulate goals in a dynamic process, restructuring, regenerating and dissolving itself accordingly);
- Integrated to facilitate information exchange;
- Monitored to ensure correct operation.
The five modules are:

1. **Resolver**: this produces job profiles, goal formation and decision making processes; and involves schedule, goal and task generation. After different resource profiles have been considered in the context of the fractals objectives, the most appropriate is selected and scheduled as part of the management activity with the appropriate tasks being generated. The fractal's objective(s) are based upon partial or incomplete goals sent by a higher fractal to those at a lower level, which will attempt to complete them, utilising the available resources.

This may involve accessing a knowledge base as part of its decision making process; while some knowledge is inherent (supplied by the designer), some the system has to ‘learn’ and ‘recollect’. To complete its assigned goals, a fractal may have to negotiate with neighbours for functionality and resources. Similarly, others will enter negotiation for this fractal’s resources. While the ‘negotiator’ can
reject some requests automatically; 'reasonable' requests will need to be sent to
the resolver for appropriate consideration;

2. **Observer:** monitors the fractal network state, receives and transmits composite
information from other peer, higher, and lower level fractals; advising the resolver
or analyser. As most systems are attached to 'something', this equipment will
need to be monitored. However, this may be indirect, as it may not necessarily
occur at the lowest level;

3. **Analyser:** this determines the real time system 'work loading'; involving schedule
evaluation, dispatching and simulation;

4. Depending upon the operating environment, a system would normally control
resource access, i.e. undertake schedule management and evaluation. Also,
given an unknown resource profile or multiple failures, it may need to re-optimise;
which could require a simulation with the results being passed to the analyser.

**Organiser:** fractal status needs to be managed during operations and when
‘negotiation positions’ are established. Similarly, the address location of adjacent
fractals, both physical and logical, needs to be known; so that information is sent
correctly and resource assigned etc;

5. **Reporter:** literally, reports process status results in the fractal to others. Most
messages are commands to control the system, either: sub-goals for sub-fractals
and status request messages; negotiation replies and status reports of the
‘super-fractal’; or the ‘best’ job profile tasking. As part of its network and
equipment command functions; it assigns addresses, controls general
communication (with other fractals), and tasks equipment.

Correct operation, requires fractals to be monitored and their performance evaluated,
with sub-fractals being generated or deleted as required. Also some form of system
and network control is required, as the fractal will ultimately run physical devices and
undertake address management; with updates communicated to other fractals. This is achieved through module ‘sub-element’ interaction, see Figure 11.

### 4.2.3 Fractal Interaction

The fractal serves the system by “interactively correcting its relations and goals, cooperating and negotiating with others” (Ryu and Jung, 2003). To provide services, fractals exchange information, negotiate functional roles and resource access etc., see Figure 12. However, the fractals multi-dimensional, multi-facetted, and self-symmetrical nature, means that size and ability to interact both internally and with ‘others’ is relative; being dependent upon the reference point, Figure 12.

**Figure 12: Fractal Interaction**
Interaction is vital to the fractal’s ability to self-organise for both internal and overall system optimisation. This can involve (part) service/resource transfer to another fractal; or better internal organisation to provide the same service for less resource.

**Fractals are therefore characterised by high individual dynamics, and an ability to adapt to a rapidly changing environment (i.e. epigenetic behaviour).**

Fractal interaction is also dependent upon a decisive characteristic known as ‘vitality’, (Tharumarajah et al, 2003). A fractal is only maintained for as long as it proves useful for overall system success; with an operating ‘life’ varying from a one-time-only need to that of a constant requirement throughout the system’s service life.

From an interaction perspective, Ryu and Jung acknowledge that a fractal has specific characteristics, but also exhibits ‘agent’ properties. For example, ‘a fractal can move’, i.e. the fractal configuration can be superimposed elsewhere in the system where conditions are ‘more’ optimal, and the old components (cells) subsequently become resource for constructing another, Section 4.1.

**While Ryu and Jung’s architecture (Figure 11) is based upon a series of interacting agents, it is the capability to be able to perform these functions, and not the mechanism itself, which is important.**

For example, a fractal can elect a ‘proxy’ to act upon its behalf with another; without it having to send its own ‘agent’. Similarly, another mechanism would be for cells to use GA’s as their operating paradigm, and exchange information in the form of ‘DNA’. Consequently, there are a number of approaches available, to enable fractals to interact and provide the various system peer and hierarchical services.
4.2.4 Service Provision and Design

Through internal functional co-ordination and collective cooperation externally with others, fractals progressively combine; to initially provide ‘low-level’, and subsequently ‘high-level’ services, Figure 12.

Therefore, it is appropriate to treat a fractal as a ‘hierarchical network’ providing a series of inter / intra related services.

This has implications for architectural design when addressing divergent performance requirements (M881, 2002; T823, 2004). As Venkatasubramanian et al (2004) identify, both natural and artificial ‘network’ structures and organisations are related to survival (performance objectives), and therefore service provision over time. A complex system, either through design or selection, maximises its ‘overall fitness’ through a combination of short and long-term survival requirements; where efficiency is related to short-term survival, and robustness is concerned with the longer term. These objectives often conflict, requiring a trade-off between efficiency (functional performance) and robustness (the ability to perform during adversity). This subsequently necessitates a balance between ‘average’ and ‘worse case’ survival requirements.

Consequently, this impacts goal setting within fractals and hence their service provision. Furthermore, as each develops differently, depending upon its instructions, resources and those of its neighbours; the ability to change goals, results in local (low-level) services and ultimately system performance optimisation, (ontogenetic behaviour) in response to environmental changes. (High-level goals (services) require an inheritance mechanism to ensure consistency).
This capability to adapt to local current surroundings, while retaining long-term (service) goals, provides a major component of a robust system design.

4.2.5 Response to Failure

“Most real world problems are not fixed but rather change with time….and practical experience has demonstrated that the goal of building totally fault free systems …is impossible to achieve”. (Glackin et al, 2004). The fractal architecture’s ability to optimise itself, and respond to a resource or environmentally induced failure; requires cells capable of adaptation / reconfiguration, Section 3.3. This naturally requires the system to identify its current status and potential response, (i.e. its future configuration). Glackin et al identified that this can be undertaken either internally or externally; with the results ‘presented’ to the system:

- **Extrinsic**: fitness is determined through simulation, with the final version being downloaded into the system;

- **Intrinsic**: fitness is directly evaluated within the system.

Their work established that an intrinsic approach is generally faster, as there is no need for additional overhead i.e. software simulation; and although failures still occur, self-diagnosis and self-healing mechanisms within the adaptive system minimise their impact. While some self-repair mechanisms were proposed, the consequences of where and how failures occurred, was not explored thoroughly, e.g. the failure may prevent the system from (initially) establishing the revised system configuration. However, they do state that:
“The real attractiveness and power of evolvable hardware comes from its potential as an adaptive hardware that can change its behaviour and improve its performance while executing in a real physical environment”.

4.2.6 FPGAs within a Cell

The fractal’s self-organising ability is a vital aspect of self-optimisation, (Section 1.1). Hence, at the lowest level (hardware implementation), components need to exhibit the same properties as the cells; i.e. they are capable of organising into self-similar structures and reconfiguring. This requires ability to:

- Change the functional partitioning between hardware and software;
- Change or transfer functionality between system areas;
- ‘Self-heal’ by working around damaged areas;
- Store the ‘functional requirements’ and operate at the lowest level, i.e. a system-on-chip approach, as part of a system-of-systems design.

A FPGA exhibits these capabilities, and the advantages have long been recognised and applied in several, but limited areas (Zhang and Ng, 2000). As such, these properties provide Tharumarajah with his ‘vitality’ characteristic (2003).

Furthermore, these properties enable a FPGA to act as a bionic building block in a fractal architecture, creating a ‘true’ fractal.

Although a fractal is ‘traditionally’ considered to be agent-based, and a bionic system is created from cells (Tharumarajah, and Ryu and Jung; 2003); the above proposed approach, is more consistent with the standard definition of a fractal, and is also more appropriate from an architectural viewpoint (Williams, 1997). However, the mass usage of FPGAs as basis for a building an entire architecture in the proposed
context of an inherently resilient system, has not been previously considered based upon the literature review.

4.3 Agent Based Operating Systems

A fractal-based architecture can potentially utilise either GA or agent based operating systems, and these two approaches are discussed below, Section 4.2.3.

(Software) agents are connected over a network, with each controlling at least one resource, and with one or more ‘sales’ agents connected to the network representing ‘customers’. Placed in a hierarchy, the higher agents can set the constraints for those lower down resulting in a ‘market’, i.e. a “system composed by a set of components – agents – which interact locally, directing the system towards a coherent global behaviour” (Bastos et al, 2005).

This approach doesn’t require a central coordinator, as the systems’ operation is reliant on the buying and selling behaviour of its composite agents. Simply put, the functional entities combine together to undertake a particular activity and offer a ‘bid’ to the enterprise agent – this selects the most economical in order to provide the requested activity, i.e. system service.

Bastos et al (2005), proposed the use of a multi-agent model for resource allocation taking into account temporal and synchronisation aspects; and disturbance impacts, such as functional failure or external events. Multi-agent systems can consequently address emergent (instead of planned) and concurrent (instead of sequential) issues, creating a self-governing solution. While this creates advantages, e.g. a multi-agent dynamic resource allocation and trading process can manage disturbances in real-time, and therefore short-term survival requirements; this is not unique.

GA’s can exchange code block ‘DNA’ (and resources) with neighbouring cells, resulting in an equivalent agent-like behaviour.
Furthermore, an agent based approach has a number of limitations, which are not shared by GA’s:

- A GA can adapt itself through ‘mutation’ and ‘cross over’ (i.e. ‘sexual’ reproduction). This offers an opportunity for long-term survivability when the system needs to adapt outside of its original requirements (paradigm). However, agents can only ‘buy’ what is available, they cannot create a new option from themselves (agents can clone themselves, but not have ‘children’ and so evolve).

- Neither Ryu and Jung (2003) or Bastos et al (2005) address the scenario of an error occurring in the agent, e.g. a ‘false sales request’, Section 3.2. For an agent there is no effective means of correcting it; creating a situation whereby a systematic or systemic, failure may occur. In a GA based fractal system, not every cell will have the same ‘DNA’; as each only needs to be ‘good enough’ to produce its intended service reasonably efficiently, (a process known as local optimisation), Section 4.2.3. This ensures sufficient ‘bio-diversity’, which in the event of a disaster, enables a surviving cell to provide the service by reproducing itself; or a previously suppressed mutation becomes prevalent and colonises the system, addressing the situation.

**While a GA may be more appropriate as an operating system, agents (behaviour) call still play a significant roll within the system by acting as ‘B-cells’ or ‘T-cells’.**

B-cells recognise and lock with specific antigens within the body while the T-cells destroy them. Once detected by the immune system, the body remains permanently immune to that type of antigen. The equivalent scenario can occur in a silicon based system, i.e. recognition and identification, repair of a (transient) fault, and future
recollection if it occurs again. Agents are ideal for performing this type of ‘anti-body’ roll. (As in a biological system, new or mutated agents will need to be created for new faults).

4.4 GA Based Operating Systems

Section 4.2 outlined how FPGAs could be arranged in a fractal architecture. While this provides the physical basis of the proposed solution, the system still needs to know where, when, and how to respond to inherently unknown changes in the environment. This necessitates an adaptive operating system capable of configuring both itself and the physical architecture. This sub-section therefore considers how a GA based operating system would work in a fractal based architecture by considering basic GA properties, cell design, life-cycle, the impact of multiple different GAs (bio-diversity), and potential failure modes.

4.4.1 GA Properties

Throughout this document, the term ‘GA’ has been used in a general context to describe an evolutionary strategy based upon (phylogenetic) Darwinian principles, Figure 13. Simply put, a GA considers a targeted population which is represented by a data string of ‘solution option parameters’, which is subjected to a set of interdependent ‘operators’; to establish a potential solution space. The GA generates a number of possible ‘local’ solutions, which are gradually optimised until the single best fit is established. A number of authors describe this, such as Li et al, (2005); Eklund, (2004); and de Toro Negro, (2004); although Beyer (2001), provides a more comprehensive treatment.

While there are differences in evolutionary strategies; the main components consider the principle of ‘variation’ and ‘selection’, combined with a change in the generation (‘reproduction’), see Figure 13:
• **Selection:** The best individuals from a given set are deterministically selected to be carried into the next generation based upon their ‘fitness’ value, i.e. they exhibit (specific) characteristics which are deemed to be ‘more important’ or ‘better’ from an ‘environmental pressure’ perspective;

• **Mutation:** this operator is problem dependent, and is selected according to various rules (Beyer, 2001); basically, it determines the amount of ‘change’ between one generation to the next, see Figure 14;

• **Recombination:** affects the amount of similarity between parents and offspring, controlling the range of diversity in a population, providing a basis for ‘inbred – thoroughbred’;
• **Reproduction:** determines which parents are required to produce offspring; effectively it controls the degree of ‘polygamy’ in the parent population, and therefore the number of children each is capable of producing, see Figure 15.

**Figure 15: Reproduction & Recombination**

Several authors have proposed variations on this model to reflect that “most real world optimisation problems are multi-objective in nature since, they normally have several (usually conflicting) objectives that must be satisfied at the same time”. (De Tero Negro et al, 2004). A typical example occurs where “a parallel GA is an evolution of simple GA in order to solve problems encountered when a multimodal function is optimised….and the word parallel….refers to a partitioning of the population in subpopulations” (Fröhlich et al, 2001).

**4.4.2 GA Cell Design**

A single GA DNA string consists of a series of parameters (‘allele’), each of which may be represented by a bit string. This is similar in concept to horizontal architecture microcode (M881, 2002), although more ‘combinations’ are possible.
Furthermore, each cell initially contains the same DNA containing the parameters, but with different ‘values’. These can interact by changing individual parameters or their values (random mutation), or an entire string section may be swapped to produce a new variation (offspring).

Each subsequent generation is tested for its relative ‘fitness’, with only the most promising candidates included in the subsequent breeding population.

Consequently, if each cell within a fractal could exchange its ‘DNA’ with that of its neighbours, it would be possible to improve the overall operation of its service provision or adapt to environmental changes, see Figure 12.

Furthermore, the DNA could be represented by a ‘double helix’, i.e. ‘one side’ controls the day-to-day operations, while the other the parameters by which those operations are undertaken (phylogenetic behaviour).

Also, if correctly designed, fitness parameters could control the swapping of small code blocks, i.e. a variation of component based software engineering, (M880, 2000; Nordin et al, Accessed 08th October 2006); Figure 15. Once formulated into ‘chromosomes’ sequences providing specific services; subsequent variations could be controlled by GA operators, Figure 16.

Figure 16: GA String Sequence
Therefore GA properties can be used to form the basis of an operating system within a fractal cell. As the cells operate collectively to provide various services, the operating system of each cell must interact with other cells.

**Hence, performing operations on both sides of the double helix means that the GA based fractal architecture will change over the long-term (i.e. its a heterogeneous complex adaptive system); consequently it cannot be regarded as a true autopoietic system, Section 4.2.1 (Stacey, 2003).**

### 4.4.3 GA Cell Life Cycle

Unless a fractal provides an ongoing benefit (service), it represents a resource drain, Section 4.2.3. Consequently, both the cell and its associated GA experience a lifecycle, from initial request to (break up) for resource redistribution, see Figure 17.

*Figure 17: GA Controlled Cell Life Cycle*

Each cell initially consists of a ‘blank template’, which when switched on, starts to modify itself; based upon its GA instruction set, and various internal and external environmental parameter inputs.

(Confirmation checks enable cells to remember their subsequent configuration).
Different parts of the GA string are activated during start-up (or during reconfiguration) in response to environmental ‘signals’, some of which are derived from neighbouring cells; allowing it to determine its own functionality; the biological equivalent is hormone activation of a zygote.

Hence, through mutual interaction and in accordance with their ‘DNA’ instruction sets, neighbouring cells join together to provide local services, e.g. memory access, with status information used to set GA’s parameters.

**This approach mitigates the excessive rule based approach of Von Neumann’s Cellular Automata, and allows true ontogenetic behaviour to occur (Lala et al, 2005).**

By combining local activities to provide specific or dedicated functions, increasingly sophisticated (high-level) services can be achieved; creating a ‘service level’ equivalent to a ‘standard’ computer.

**By adapting to their immediate surroundings, cells can provide local and ultimately higher level services, consistent with the ongoing system’s need.**

As environment status includes resource availability, which can be encoded in a ‘standard’ GA data string; this information can modify system operation by changing fitness values (within defined limits), enabling it (the DNA string, and hence the cells functional operation) to evolve, Section 3.3. Similarly, the cells will continue to self-optimise during normal operations; and ultimately, the local configuration will stabilise when changes have no material impact upon operation. Hence, if the GA string, and therefore the cell, is ‘successful’, it will rapidly propagate throughout the local area. This creates different cell populations with local DNA variations, each providing similar functions throughout the system.
As the cell function is defined by the local environment, e.g. does it control a keyboard or memory; its evolutionary rate will be different, Figure 16.

Stable populations only change when environmental inputs or command objectives from higher fractals vary. Hence cell operational re-tasking, and the fractal function, may involve DNA strands which were either previously dormant or never activated.

Consequently, all cells require a copy of the complete operational ‘genome’ if it is to potentially perform multiple functions throughout its life.

From an operator’s perspective, providing that the appropriate service is available, how it is achieved is unimportant. Once operating, cells may be moved or replaced in the architecture without impacting the service, as capability is transferred to adjacent cells. Ultimately, when no longer needed, the cells and the associated fractal element ‘dies’; with constituent parts becoming resources for others (i.e. are re-tasked). Hence, as in biological systems, cells are ‘lifed’, and the DNA string should contain a ‘life clock’ (‘telomere’) to provide ‘vitality’ (Tharumarajah et al, 2003).

However, the importance of some functions means that their ‘clock’ is effectively switched off, and they become immortal.

4.4.4 Cell ‘Bio-Diversity’

In real life, diversity is the key to overall species survival, but not necessary an individual’s (Jones, 1999; Lala et al, 2005). Naturally, this has consequences, e.g. in agriculture, only a few hundred species of plants are used commercially. Hence, if a disease is introduced, then a plant’s population uniformity is a significant contributor in ensuring infection. This scenario can be seen in the way that computer viruses utilise existing operating system weaknesses.
In order to achieve robustness and therefore long-term survivability, a key feature is achieving diversity through adaptation.

As identified previously, different cell populations with local DNA variations can provide the same function. This effectively creates a barrier to (adverse) propagation, while acting as a potential DNA reservoir. The latter provides a source of alternative, and therefore potentially ‘fitter’ GAs, in the event of minor environmental changes; which can be used to ‘repopulate’ the now different system.

In most situations, if offspring are radically different from their parents (i.e. severely ‘mutated’), they normally do more harm than good.

Within GA’s, this is controlled by the Recombination Operator (Beyer, 2001). Radical offspring are only effective when they confer significant advantages, or when existing populations can’t cope with the current situation, i.e. a ‘massive’ environmental and or resource change occurs. Only then, do they have the opportunity and capability to replace the existing (ineffective) population. This may occur on a large scale, if the change impact is sufficiently systematic or systemic.

Alternatively, there maybe a unique localised situation i.e. a niche environment. Hence, if contact is subsequently re-established with the main population following a reconnection; the specialised offspring can fail to re-integrate, either dying-off or aggressively replacing them (Jones, 1999). Although taken from a biological standpoint, the above scenarios can be applied to computer generated GAs operating from within fractal (cell) based architectures.

Diversity, rapid propagation, and GA modification also forms a double loop learning process, which equates to a population based ‘genetic memory’.
This potentially eliminates a need for a knowledge base to inform decision making, as proposed by Ryu and Jung (2003), Section 4.2.2.

### 4.5 GA and Fractal Failure Modes

As with any natural or artificial system, there are a number of potential failure modes which typically manifest themselves as ‘performance issues’, for example:

- Although fractal architectures are inherently failure resistant, a potential scenario occurs, when two previously isolated sections are reunited. While the ‘who’s in command’ situation is avoided by the system’s structure, the new environment will produce changes in GA’s functionality, as they reconfigure. During re-optimisation and allocation, a consequential potential drop in performance may occur. If this is significant, ‘alternative solutions’ identified during normal operations may be recalled for use, (Shanthi and Parthasarathi, 2005).

- Memory ‘overhead’ for each cell’s operation versus that necessary for storing their ‘DNA’ may present a concern. Consequently, chip availability and their percentage utilisation, may quickly reach capacity in some areas of a fractal architecture, necessitating cell reconfiguration.

Furthermore, fractals potentially experiences the same issues associated with a distributed system, e.g. the need to share critical resources; and hence experience a number of transient faults, Section 3.3.1. This requires that it should “return to a legal configuration in a finite number of steps and remains in a legal state until another fault occurs”, i.e. it’s self-stabilising (Datta et al, 2004).

Other potential ‘failures’ occur through the GAs method of operation; i.e. they won’t necessarily find a solution to a particular problem in ‘real time’; or it’s not optimal (Beyer, 2001). Additionally, multiple GA runs are often necessary for performance
confirmation (Li et al, 2005). Although these are addressed by appropriate GA operator usage, from a practical perspective, it is often only necessary to find ‘a solution that works’ (in a timely manner), with optimisation occurring in ‘slow time’.

Extensive GA use to form an operating system base, also presents the possibility for more exotic types of failures:

- The fractals and their method of exchanging data effectively represent a ‘polygamous society’, giving rise to potential ‘social diseases’. These may originate internally through ‘bad DNA’ generation, e.g. excessive radical solutions; or arrive externally, e.g. a ‘true’ computer virus.

- ‘Bad DNA’ also gives rise to the concept of ‘cancer’, where a particular function grows in an uncontrolled manner ‘consuming’ all available resources, and is not checked by the rest of the system (Lala et al, 2005).

However, these may be addressed through sufficient biodiversity and an active ‘immune system’; Sections 4.3 and 4.4.4.

### 4.6 Summary of GA And Fractal Architecture Research

This Section has discussed possible alternative architectures and operating systems, Sections 4.1 and 4.3; which could potentially address the issues identified in Section 3. It has proposed a potential solution based upon a fractal architecture consisting of bionic cells, and a GA operating system, Sections 4.2 and 4.4. Key aspects have been explained in greater depth, by focusing on achieving ‘service equivalency’.

Each cell contains its own ‘DNA, which is modified through interaction with the environment, including other cells. Consequently, the Section has identified that a GA based fractal architecture is a complex adaptive system, capable of reconfiguration and re-tasking itself to continuously provide the required service.
5 Review of GA Resilience in Fractal Architectures

Section 3 identified the issues surrounding the effectiveness of current system designs in terms of resilience, while Section 4 proposed a potential solution. This Section examines the proposed approach by addressing those aspects which make GAs and fractal based architecture resilient; and therefore satisfies the issues raised by Section 3.

5.1 GA Resilience

A GA based operating system utilises a number of functions, which are not traditionally implemented; Figure 18. The objective is to provide a consistent level of service to the user irrespective of what happens ‘behind the scenes’. Furthermore, if service levels have to be reduced, those remaining are prioritised to the users needs. Hence, while the literature covers many of these aspects, they have not been placed in an integrated context.

Figure 18: Operating System Fractal Functions
5.1.1 Hardware & Software Partitioning

For many systems, functionality is determined and subsequently set during design; with hardware implementation being traditionally used for speed prioritisation, while software was used to obtain flexibility. Consequently, with the exception of a few special circumstances involving design facilities or military environments, it was neither practical nor appropriate to change functionality (i.e. re-designating hardware and software partitioning) during normal operations, Section 4.2.6.

This lack of re-configurability has consequences during failure, as it effectively excludes the remaining parts of the system from operating.

*Reconfiguration can occur in response to ‘local’ needs (Bower et al, and Lala, et al, 2005), e.g. damaged chips; or system optimisation requirements.*

This represents a major system resource loss, with potentially catastrophic consequences, Section 1.1. To address these issues effectively, means that the operating system needs to control hardware and software partitioning. This necessitates an architecture composed of reconfigurable components, i.e. FPGAs.

Managed re-configuration (and re-tasking) provides advantages for maintaining service levels and achieving graceful degradation, by allowing the sacrifice of non-essential areas to support critical functions, Section 4.4.3. Furthermore, it allows a system to have both speed and flexibility denied in a ‘pure’ solution (Zhang and Ng, 2000). For example, a GA was used by Harkin et al (2001), to partition (FPGA) hardware-software functionality and so obtain (near) optimal performance speed-up relative to conventional software implementations. While there is still a need for initial partitioning during design, as discussed by Finc and Zemva (2005); the ability to perform this function dynamically means that an FPGA can be partly reconfigured or
incrementally adapted, i.e. re-tasked during run-time execution (Fröhlich et al, 2001; and Harkin et al, 2001).

The ability of the operating system to change functionality (dynamically) can offer significant benefits, either to work around a damaged area or to re-optimise the remaining system; thus ensuring no (or minimum) perceptible loss of service and or significantly reduced down-time.

5.1.2 Software Rejuvenation

In a ‘normal’ system, software typically induces more failures than hardware, which slowly accumulate in a process known as ‘software aging’; resulting in performance degradation, (Xie et al, 2005) Section 3.2.2. Aging is further accelerated by communication failures within ‘clustered systems’, and is also more difficult to detect (‘heisenbugs’). While techniques such as recovery blocks are available, their general expense often precludes clustered system application; (Park and Kim, 2002), Section 3.4.2.

While a GA based operating system may accumulate aging errors, such as memory leakage; it operates within a fractal based architecture, i.e. an effective ‘patchwork quilt’ of interlocking ‘individual’ systems. These dynamic fractals only survive long enough to perform their intended function, before being broken up.

Preventative maintenance (‘rejuvenation’) involves taking the software temporarily off-line to fix the errors associated with aging; either partially after saving all necessary data, or as a full re-start (Bobbio et al, 2001)

Rejuvenation involves buffer flushing, file purging etc; and occurs when either a time or measurement ‘error threshold’ is reached. So although an overhead, rejuvenation
Improves system availability and reduces down time costs (Liu, et al, 2005). Hence, in a fractal system, rejuvenation occurs at a 'lower-level', and potentially more often. However, this reduces significant error build-up across the system, and improves overall resilience. The system-of-systems / distributed nature of fractal architecture means that rejuvenation is a key aspect for successful inter-cell management and communication, (i.e. cells will need to integrate their activities).

**If rejuvenation involved dynamic functional partitioning, then the operating system could isolate affected systems during repairs, with no noticeable impact at the user level, i.e. a partial restart.**

Consequently, a restart could trigger a (GA) configuration control mechanism to re-assign functionality to neighbouring units; and if the ‘damaged’ (fractal) system was repaired, then its re-introduction in the architecture would trigger another reconfiguration event.

### 5.1.3 Re-tasking And Changing Functionality

**A key capability of the proposed solution, is the ability to re-task, i.e. partially or completely re-assign functionality within a fractal in response to internal and external environmental changes, Section 4.4.3. Whiteson and Stone (2004) identify this as a major long-term cost saving.**

Re-tasking occurs when either an existing service requirement is ‘transferred’ to another fractal; or a fractal is changed in whole or in part, to enable to it provide a ‘new’ service; Figure 19. The ability to supply a given service and therefore the requirement to re-task is reliant upon the number of fractals already providing a similar service; relative importance; available resources; fractal efficiency etc. This
information is held in the GAs ‘double helix’, as operation requirements and their associated parameters; Section 4.4.2. Consequently, a (sub)fractal i.e. a cell or ‘cell cluster’, can be re-tasked and or have its functionality changed, to ensure that an overall service level is maintained; Figure 19.

**Figure 19: Fractal Cell Re-Tasking & Reconfiguration**

Functional modifications can occur either during start-up or in normal operations, where fractals need to (re)optimise to reflect environmental or resource changes. The GA ‘process’ places the fractal(s) in a “default configuration and then incrementally tune themselves to the needs of a particular enterprise based on observed usage patterns” which are in-turn provided by environmental stimuli (Whiteson and Stone, 2004). GAs effectively enable individual fractals to operate autonomously, and so adapt to changes by re-tasking and reconfiguring itself and others, Section 4.2. Hence, each fractal continuously modifies its environment and is
also changed by that same environment simultaneously; creating a complex heterogeneous system (Stacey, 2003).

Hence the ability to (continuously) re-task and reconfigure creates an inherently resilient system, and it is this interaction which provides the basis for the operating system; i.e. (emergent) fractal cell interaction.

An individual system (fractal) may not always be able to perform its entire assigned function, requiring it (the function) to be partitioned. As Sklyarov (2002) identifies, the ability to undertake this (re-tasking), requires “modularity of the behaviour specification… and… the capability for reloading (or re-switching) and changing the modules either statically or dynamically”; i.e. dynamic reconfiguration at both the FPGA and operating system level (Harkin et al, 2001; Ryu and Jung, 2003). However, this may result in system topological changes, and although dynamic reconfiguration can re-establish paths; deadlock and livelock problems may occur.

Hence, operating system dynamic reconfiguration needs to tolerate multiple simultaneous node and link failures on a potentially arbitrary architectural topology, (Aversky and Natchev, 2005). Also the operating system needs to select its own optimal architecture, given its required service provision (Oh et al, 2005).

5.1.4 Resource Allocation

The need to reconfigure and or re-task is driven by resource availability; for a fractal, this is characterised by ‘internal’ and ‘external’ environmental factors relative to its size, Figure 19. When an internal failure is perceived at the macro-level, a (sub)fractal is re-tasked with providing the service (element), Section 4.2.3. However, at the micro-level; the same failure occurs externally to other micro-fractals, with the
damaged fractal becoming a resource for its neighbours, (the fractal boundary effectively shifts through a combination of (re)tasking and reconfiguration). Shifting boundaries, means that resource allocation is a key activity.

The ability to use resources efficiently is a strong GA survivability characteristic, which governs fractal propagation and mortality.

Venkatasubramanian et al (2004), identified efficiency and effectiveness as key drivers, Section 4.2.4. As resources are finite, allocation requires careful consideration, especially in ‘distributed’ systems where multiple sites can produce the same service. ‘Normally’, resources are pre-allocated and are controlled by designated, albeit, competing functions. However, within a fractal architecture, resource allocation is controlled by cellar interaction, i.e. Darwinian principles driven by DNA requirements, Section 4.5.

In addition to physical resource, other resources also require management, Section 3.3.2. Both Gen et al (2005) and Li et al (2005) proposed a GA to address workflow, i.e. ‘service’ optimisation. An identical process is required in fractal architectures, which together with the ability to re-task, provides an alternative service resource. Re-tasking requires the GA to optimise the operating software, its versioning, and determine the execution sequence “such that system reliability….is maximised and total cost remains within budget”, (Levitin, 2005). As resource allocation (and control) occur at the cellular level, the overall system is more tolerant of faults; and is therefore capable of maintaining services for longer. This avoids an ‘all or nothing’ approach characterised by traditional systems.

5.1.5 Mutual Interaction

Each cell functions within a fractal, and that fractal within another; with its operation determined by ‘environmental inputs’ or objectives set by higher level fractals (i.e.
epigenetic behaviour), Section 4.2.3 and 4.4.3. Consequently, the overall system operation (and hence resilience), is dictated by intra and intercommunications between cells, (sub)fractals and fractals:

- Although every cell communicates, some will be dedicated to data exchange, and act as the fractal’s main interface. It is this many-to-many interaction between and at multiple levels, which creates an overall ‘operating system’;

- Fractals exhibit many of the properties (and issues) associated with a distributed system, especially if the role of the ‘operating system’ is considered; e.g. resource sharing. These can be continuous or special event driven, requiring compromises and prioritisation, all of which relies on communication;

- Changes in the GA based operating system involve mutual interaction through multi-generational solution development; especially if undertaken in parallel, i.e. (parts of) a solution are shared amongst multiple locations, utilising a potentially unknown architecture (Eklund, 2004; de Toro Negro et al, 2004);

  This is complicated by the operational context, as contributing inputs can change part way through the (evolutionary) optimisation process, modifying it.

- Sharing resource introduces dependencies and propagates potential failures. This was addressed by Younis et al (2004), for a multi-level architecture using general-purpose basic components (i.e. fractals), where strong temporal and spatial partitioning, ensured continued safe operation if failure occurred.

Consequently, mutual cellular interaction forms the basis for the operating system; however, as it is an emergent property, its exact operation is inherently unpredictable (Stacey, 2003).
However, as no one fractal is responsible for system control, it creates an inherently resilient system, capable of adapting to its surroundings.

### 5.2 Fractal Architecture Resilience

Fractals are a form of autonomic computing, which has traditionally been considered a ‘bottom up approach’ based upon optimising individual components rather than entire systems. As Whiteson and Stone (2004) identify, this “fails to address the effects of interactions between various components and does not capitalise on opportunities to optimise at a system-wide level.” Some of these system level issues are discussed below; see Figure 20 and Section 4.2.

**Figure 20: Fractal Architecture Considerations**

<table>
<thead>
<tr>
<th>Operating System (Genetic Algorithm)</th>
<th>Architecture (Fractal Cells)</th>
<th>Service Level Equivalency &amp; Prioritisation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FPGAs</strong></td>
<td><strong>Self-Repair</strong></td>
<td><strong>Cell Formation &amp; Interaction</strong></td>
</tr>
<tr>
<td><strong>Dynamic Reconfiguration</strong></td>
<td><strong>Changing Functionality</strong></td>
<td><strong>Layered Mutual Interaction</strong></td>
</tr>
<tr>
<td><strong>Service Prioritisation &amp; Equivalency</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.2.1 FPGAs

FPGAs changed the distinction between hardware and software with their ‘Run-Time Reconfigurability’ feature, Section 5.1.1. They can be reconfigured partially or completely during operations, changing the software / hardware functionality split, for
an unlimited number of times; (Fröhlich et al, 2001). FPGAs therefore create an effective architectural basis for (resilient) fractal formation, Section 4.2.6.

**FPGAs are autonomic, i.e. systems capable of self-repair, self-configuration, self-optimisation and self-protection (Marsh et al, 2004; Sterritt, 2004).**

This requires FPGAs to be ‘ontogenetic’, where the “structure of the array is homogeneous, i.e. cells are similar and only their functions distinguish them from their neighbours”, (Lala et al, 2005). However, as Zhang and Ng (2000) stated, dynamism significantly complicates (FPGA) reconfiguration as resource allocation changes continuously; making the exact execution sequence difficult to predict. Furthermore, ‘reconfiguration latency’ presents an additional complexity (Harkin et el, 2001); although as Fröhlich et al (2001) identify, reconfiguration time is now less than 1ms for an entire device.

### 5.2.2 Self-Repair

While an appropriate design can enable the remainder of the system to be available in the event of a failure; the ideal situation would occur if the affected system was capable of self-repair. This would enable the system to return to maximum effective operational capacity, while allowing it to respond if another failure occurred.

At the component level, the FPGA uses self-diagnosis and self-healing mechanisms to minimise failure impact, often creating a structural ‘work around’ (Glackin et al, 2004), Section 4.2.6.

**‘Self-repair’ uses spare cells to ‘route around’ a fault, while self-(re)configuration enables an FPGA to respond to failures occurring elsewhere. Together, they provide a twin tiered approach for maintaining critical services.**
‘True’ self-repair in the biological sense, requires fractal cells to undertake repairs at a molecular level (Lala et al, 2005). However, this may be simulated at the sub-cell level using “indirection to map out the faulty portions of the structure”, and spare rows in the FPGA array structure. Lala et al proposed an architecture, whereby a spare is dedicated to replacing one of a group of four cells, if it is found faulty; by cloning its functionality. This approach is especially suitable for soft errors in functional cells or on interconnection lines; as the spare is connected to the inputs of those that are physically adjacent, Section 3.2.

Dedicated check rows can also confirm hard faults; while error detection codes, using a pre-existing branch mis-prediction recovery mechanism, enable recovery from transient errors and errors due to hard faults, that have not yet been classified as hard, (Bower et al, 2005). Bower et al, also indicate that while both of these methods add some “performance overhead compared to an unprotected system, where the repair logic is on the critical path...if a hard fault occurs, then they outperform the existing fault tolerant approaches while avoiding large scale replication of microprocessor cores”; this represents a significant performance and cost advantage, Section 3.4.

Lala et al (2005) concluded that commercially available FPGAs can be used for digital systems with self-replication and self-healing features.

While these design aspects are capable of being applied at multiple levels; at a higher system level, an alternative mechanism to self-repair can also occur, whereby the remaining system synthesises the lost ‘aspect’. For example, it is important to be able to compensate for lost or damaged sensors in an array. This can be achieved through data synthesis of the remaining (close proximity) sensors to extrapolate the
lost data, although appropriate error margins may need to be revised and suitable advisory messages generated; (Marsh et al, 2004), Section 4.1.1.

**The ability to self-repair is a fundamental aspect of an effective resilient architecture, as it ensures that the maximum amount of resource is available.**

### 5.2.3 Cell Formation & Interaction

Using inputs from its neighbours, the environment and the ‘instructions’ contained in its genome, enable the cell to determine its function, (Glackin et al, 2004); Section 5.1.3. This interactive modification process facilitates which cells are associated with a given fractal and in what capacity.

Although redundancy of information is key to system level resilience, as complexity increases, a greater percentage of cells are involved with storing the genome, and hence impact resource availability (operational reconfiguration and parameters are also superimposed upon this basic genome). Furthermore, as far as a particular cell is concerned, the majority of the information is ‘redundant’. Several approaches can address this, each having a potentially different impact upon architecture resilience:

- Only ‘every other cell’ holds a copy of the genome;
- Cells contain the complete genome at start up, but this is subsequently broken down and deleted where not required;
- The overall complexity is reduced, enabling each cell to retain a complete genome (use multi-level GA’s);
- The genome itself is simplified to include a simple translator which may or may not be associated with the cell, and a ‘double helix’ parameter string;
- A hybrid approach of the above.
As cell formation is determined in part through mutual interaction, Section 5.1.5; this is complicated by signal transfer and hence interconnection length between various cell groups (and neighbouring FPGAs). Unlike biological systems which are physically three dimensional and where all communication occurs amongst neighbouring cells, this is a major consideration for silicon-cell self-repair and fractal recombination, (Lala et el, 2005). While interconnection length determines connection speed; the location and physical size of the links is also a consideration. Ultimately, this may require a re-examination of dye construction and chip-to-chip linking, e.g. fibre optic connections. The key consideration is that continuous interaction between cells, and hence their ability to adapt to new inputs, is fundamental for an effective resilient architecture.

5.2.4 Dynamic Reconfiguration

The value of dynamic reconfiguration stems from the ability to change during live operations, providing that performance costs are acceptable.

One method to improve dynamism uses a domain-specific approach, based upon a number of ‘templates’; where it is only necessary to configure the basic computational operations and the corresponding control algorithms i.e. GA parameters; (Skliarova and Ferrari, 2003). While this provides a basic model for dynamic reconfiguration in fractal architectures, including resource allocation (Section 5.1); typical characteristics include:

- Less idle operations, with associated applications using less resource, (Zhang and Ng, 2000);

- Reduced total ‘down time’, Avresky and Natchev (2005) proposed a revised dynamic reconfiguration algorithm which could tolerate multiple simultaneous node and link failures in high-speed networks with an arbitrary topology;
• Functionality can be split across multiple devices, Sklyarov (2002).

However, device design itself can often inadvertently hamper dynamic reconfiguration; e.g. Meribout and Motomura (2004) identified that many design techniques don’t take into account the effect of interconnections which impact overall performance. An effective design can reduce delay, by ensuring that only the ‘required part’ is reconfigured and not the whole device.

Furthermore, better design would also result in greater functional implementation instead of overhead (intra-device communication), i.e. reconfiguration effectiveness is affected by ‘granularity’ (Jóźwiak et al, 2003). Coarse grain architectures require less resource than fine grained architectures, and have more area of efficient interconnection routing switches. However, this increased performance and reduced resource utilisation, results in less flexibility; as coarse grained reconfiguration is unable to fully exploit bit level parallelism.

Consequently, the ability to change dynamically only those parts that need to be modified at the appropriate level, is essential to performance effectiveness in a fractal architecture.

5.3 GA Control in a Fractal Architecture

From the user’s perspective, it is (normally) immaterial where or how a service was generated; hence, ‘service equivalence’ needs to be considered from a number of aspects to ensure that the user remains ‘ignorant’, Figure 21 and Section 1.2.

5.3.1 Multiple Layers

Traditionally, a system is split into a number of architectural, operational and service hierarchical layers to aid understanding, ease of use, operation and maintenance. As the fractal architecture is a ‘complex adaptive system’ (Stacey, 2003); this is neither
necessary nor appropriate. Rather, the initial homogeneous design facilitates self-adaptation, and is subjected to a changing environment; creating a heterogenous architecture, and thus allowing emergent system behaviour to develop, Figure 22.

**Figure 21: SLA Equivalency**

![SLA Equivalency Diagram](image)

**Figure 22: Architecture Comparisons**

![Architecture Comparisons Diagram](image)

As cells are created, grow, divide and die to meet the overall system need; these form fractals, enabling them to grow, adapt and ‘die’ as required. Consequently, since fractals can occur inside each other, the concept of multiple layers, is no longer relevant. Rather the number of (concatenated) layers can be one, some or many, at
any given time or location within the system, i.e. the architecture is 'uncertain'. Although this fundamentally alters the way in which an operating system works, and is perceived by the designer, its now effectively a ‘distributed system’; with each cell trading requirements and resources with its neighbours above, below and adjacent. However, this immaterial to the user, as they only see the end service, Section 4.2.3.

5.3.2 Service Level Agreements and Prioritisation

The function of a system is to provide a ‘service’; hence, in considering a ‘service level agreement’, the focus is on providing a required resource or function to the right customer at the right time for the correct duration; the ‘how’ (i.e. the architectural arrangement) is no longer important. By focusing on resource allocation and tasking, Section 5.1 shows how GA’s operate through mutual interaction to provide a service, which is comparable to a ‘conventionally’ based approach. Prioritisation is another key design and operational driver, closely allied to service levels, i.e. who is served first. Although some parameters are set during design, the remainder are typically determined by operational needs, i.e. operator requests and environmental stimuli.

A fractal architecture has no dedicated system, and prioritisation is dictated by optimisation parameters during normal conditions, Section 4.4.2.

Consequently, when those conditions change, prioritisation results in preferential service trade-offs, with various functions being reallocated or moved around. Prioritisation therefore defines the preservation order; a key issue when sharing high and low integrity functions on the same system.

5.3.3 System Functional Equivalency

Service equivalency effectively implies ‘functional equivalency’, i.e. the ability to manipulate, store and process information to achieve the same desired result. A
fractal architecture achieves this by providing low level services through cell interaction, from which high level services emerge.

Section 4.4.1 proposed a neuro-fuzzy approach as a potential equivalent architecture; and although this was subsequently dismissed, it did identify that a major advantage was their ability to consider a ‘may-be’ option. However, this capability can also be developed in GA controlled reconfigurable systems, as they can respond dynamically to the environment. Consequently, the operational aspects of a GA can be used to simulate the neuro-fuzzy equivalent responses.

5.3.4 Topological Responsiveness

As a ‘distributed system’, many traditional failure modes are eliminated from a fractal based architecture; however, others associated with network loading, especially in degraded circumstances where the topology is irregular, remain; Section 4.5. While not immune, a fractal architectures’ capacity to undergo dynamic reconfiguration and self-repair significantly minimises their impact; Section 5.2.

5.4 Problem Space Summary

This Section has discussed how the properties of FPGAs and GA can create a resilient architecture; validating Section 4’s solution against Section 3. Hence, by considering dynamic reconfiguration and mutual interaction in the context of resource (re)allocation and specific autonomous capabilities such as self-repair; it is possible for a fractal system to mimic a conventional system through ‘service equivalence’ provision, Section 1.1.

Furthermore, this Section has demonstrated that ‘higher’ service levels are an emergent fractal architectural property; enabling a service level to be maintained in the presence of multiple failures, i.e. a robust resilient system.
6 Analysis

The previous Sections utilised a comprehensive literature review to identify inherent weaknesses in current computer architectures. They proposed a viable alternative design approach, and subsequently confirmed the solutions’ effectiveness in producing a resilient system. Furthermore, as this ‘narrative’ has unfolded, it has provided a progressive argument as to its own validity. However, it is also necessary to demonstrate the overall effectiveness and robustness of both the approach employed, and the architecture itself, for completeness:

- The use of a literature review is examined as a mechanism for undertaking research; and its capability to offer new insights. This provides assurance in the validity of the approach;

- By identifying the advantages and disadvantages of a fractal-based architecture, a qualitative assessment of the proposed solution is possible; i.e. whether it addresses the original research question;

- In developing a proposed system map, the feasibility of the architecture itself can be demonstrated.

6.1 Literature Review Effectiveness

Research methodologies, problem domain and associated circumstances have been discussed extensively in the literature e.g. OU (2005) and Sharp et al (2002). These sources ensured that the basis of the thesis was driven by several competing / complimentary criteria:

- Course requirements and suitability considerations;

- (Author) nomadic working and security restrictions at client sites;

- Available funding and probability of success;
Learning opportunities, relevance to work, and personnel enjoyment;

Limited employer facilities and personnel capabilities.

As a consequence, while approach selection was driven by circumstances, this does not distract from its appropriateness as a means of undertaking effective research, see below and Section 2.

6.1.1 Elementary Literature Review

At its most basic, a literature review simply provides an indication as to ‘who has done what before’. To be effective at this level, related topics have to be identified and researched, to ensure that different interpretations, phraseology etc, do not hide key issues, Section 2. A good review therefore presents sufficient and appropriate material to identify common key themes and words, which in turn provide the basis for an extended search of related topics. Furthermore, in developing an idea, the literature review provides an essential component in formulating a logical argument, where existing work is used to corroborate or repudiate different aspects.

Hence by splitting this literature review into three main sections, outlining how designers have traditionally responded to system failure; providing an alternative architecture; and showing how the alternative architecture could potentially address resilience issues; a logical argument has been progressively constructed which validates the dissertation’s main ideas; Sections 3, 4 and 5.

While the elementary review is an essential component of any research activity, a literature review of this nature is limited in scope, and does not make best use of previous work, i.e. ‘content’. However, this thesis relies on weaving several disparate core ideas together, using ‘compare and contrast’ to provide a novel architecture; while asking ‘what if’ questions to create original insight. This thesis has:
• Used fractal-based concepts, developed for manufacturing organisations, and applied them to computer system design to create a flexible architecture;

• Adapted GAs from their traditional problem solving role, and used them to control both system operation and data manipulation;

• Used GAs in combination with the capabilities of FPGAs, to create a physical system capable of automatic self-adaptation and emergent behaviour.

6.1.2 Originality

An additional function of a literature review is to demonstrate originality, i.e. no one has considered a particular idea or combination of ideas before. As in many theoretical engineering and applied science situations, original insight comes from (re)combining diverse existing ideas into new forms, or applying ‘old’ ideas to new situations, Section 6.1.1. Further, in performing this literature review, the shear diversity of information and its application, indirectly ensures originality.

6.1.3 Evidence Quality

As in any research activity, the ability to draw meaningful conclusions is highly dependent upon data quality and therefore the information derived from it. A literature review is no different, and consequentially, with the exception of a two commercial white papers, the source documentation has been derived from peer reviewed international journals, educational websites or technical books and standards who can demonstrate a similar provenance; see References. This approach has ensured that the material quality is of a sufficient factual and literary standard to meet the thesis’ needs.
6.2 Detailed Analysis

By considering the advantages and disadvantages of employing a GA based operating system on a fractal architecture, relative to a ‘traditional’ system, it is possible to assess the overall approach effectiveness in a resilience context.

6.2.1 Approach Advantages

Summarised below, the advantages represent new or additional capabilities over a traditional system:

- Exhibits distributed system behaviours in response to failure, e.g. ‘work arounds’ and no single control point (i.e. critical system), Section 5.3;
- A GA Operating System self-adapts to environmental and resource changes, allowing system optimisation and reconfiguration; Section 5.1;
- Mutually interacting FPGA cells form fractals capable of providing ‘multi-level services’ equivalent to a ‘normal’ computer, Section 5.3;
- Functionality can be rearranged and reassigned dynamically, Section 5.2.4;
- The fractal construction format consisting of multiple interactive cells results in a responsive ‘bio-diverse’ solution reservoir, with an ‘immune system’ capable of recognising and adapting to varying fault conditions; Sections 4.3 and 4.4.4;
- The ability to adapt can enable the system to cope with vague or potentially unknown responses, and hence mimic the capability of neuro-fuzzy systems, Section 5.3.3.

These advantages result in a system capable of achieving graceful degradation through service prioritisation, and unlike a traditional computer; the operating system through reconfiguration and division, can use every component for maximum service.
provision benefit. Correctly applied, a failure is therefore addressed in real-time, with no user perceived loss of service.

Also, while arguably not an ‘advantage’, the proposed approach embodies a major developmental opportunity over normal designs; as resilient, high availability networks represent a direct contributor to operational and infrastructure cost reductions and service improvements.

For a given size, a fractal based system should provide more functionality, with a greater level of resilience for less cost, over a traditional system.

6.2.2 Approach Disadvantages

The above represents a significant capability improvement; however, the proposed approach does introduce some novel situations which may potentially have an adverse impact, and would therefore require additional investigation. While some of these are unique to the proposed approach, others such as ‘overhead costs’, are consistent with normal design considerations:

- There is a potential overhead cost associated with a GAs physically storage, Section 4.5;

- A new GA failure category associated with genetic errors potentially occurs, e.g. ‘cancers’; although this may be controlled by cross comparison and an immune system, Section 4.5;

- In the event of catastrophic failure, the time taken to repopulate the fractal architecture with a viable operating system may become critical;

- Several issues impact GA optimisation, resulting in a scenario where ‘no’ optimal solution is obtained; however, often it only has to find ‘a solution that works’ (in a timely manner), Section 4.5;
The proposed approach may need to interface with existing traditional systems, and be capable of mimicking the transmission and reception of information. While the above focuses on technical concerns, the major problem may ultimately spring from ethical considerations; where computers make decisions (that impact human safety). For example, GAs use a set of rules to modify their behaviour in response to their environmental conditions, i.e. they evolve, Section 4.4.2. This may result in a set of conflicting instructions which the system will need to resolve. Furthermore, there may be an emotive backlash against such ‘logical’ decision making. Such issues and there consequences, are often explored in (Sci-Fi) literature. Classically this includes Arthur C Clarke’s ‘HAL’ in ‘2001, A Space Odyssey’; and Asimov’s short works regarding his ‘robotic laws’ made famous through the recent film ‘I, Robot’.

### 6.2.3 Approach Comparison

A traditional architecture contains little flexibility, i.e. a ‘CPU is always a CPU’, Section 1.1. Furthermore, it is also reliant upon the designer anticipating potential problems, rather than the system determining and adapting itself accordingly, Section 3.2. This has been argued as being a significant contributor to a lack of system resilience, and should be contrasted with three themes which have become readily apparent in researching this thesis:

- ‘Non-designated’ architecture eliminates single points of failure; Section 5.1;
- A dynamic and flexible architecture that rebuilds itself for maximum optimisation Section 5.2;
- A GA architecture exhibits emergent (service) behaviour, growing and adapting to its environment, Section 4.4.
These themes ensure that full system capability is always available, allowing graceful degradation to occur, and thus addresses the resilience issue originally proposed in the research question; Section 1.2. While this represents a major advantage, indirectly it also offers potential costs savings, Section 6.2.1. However, the approaches’ shear novelty, may result in the solution being rejected through inherent conservatism and fears associated with a ‘lack of control’, Section 6.2.2.

6.3 Proposed System Map

The previous sections have identified that the proposed architecture does not confirm to a standardised (stylised) hierarchical viewpoint.

Instead, it consists of a series of temporary, nested fractals formed from one or many cells; whose size, boundaries and inter-relationships with their peers and subordinates, are continuously changing and reforming as required, Figure 23, Section 4.4.3.

Although starting as a set of homogeneous standard cells, these adapt to both the environment and each other’s presence; where each cell will receive a slightly different set of inputs compared to its immediate neighbours. These variations progressively increase as the degree of separation increases. In addition, not every cell or every fractal will change exactly the same way. This is ensured by GA cross-over and mutation operators acting in both the operational code components and the associated data parameters, Section 4.4. As a result, no two systems, even starting from the same set of initial conditions will form exactly the same architecture.

Hence, the initial homogenous architecture becomes heterogeneous, as the system self-optimises to reflect the current resource and environmental requirements; i.e.
some cells / fractals will have a bigger demand upon resources, have greater
longevity, be more numerous etc.

**Figure 23: Proposed System Cellular Architecture Meta-Map**

As a consequence, cells are changing due to ‘social’ interaction with the system,
which is in turn, changing itself as a result of the same interaction (Stacey, 2003);
Section 5.2, Figure 24. Cell interaction is therefore non-linear (there is no ‘average’
interaction and the fractals are temporal in nature), creating bounded instability;
which gives rise to emergent behaviour at the ‘system level’, i.e. ‘services’.

Therefore, from a ‘systems perspective’, concepts such as feedback become
meaningless or inappropriate, as it is not possible to quantitatively predict the (long-
term) outcome of an interaction between cells, based upon the initial governing rules.
Although knowledge of the entire ‘system’ is required to understand the outcome; it is
possible to approximate the (long-term) output based upon prior understanding, i.e. the system is a strange attractor (Stacey, 2003).

**Figure 24: Proposed System fractal Architecture Meta-Map**

As there is no dedicated functionality, this can not be represented on a system map; rather it is more appropriate to consider information and service exchange between temporary peers and subordinates; Figure 23 and Figure 24. This therefore gives rise to the fractal concept, as each cell is integrated to form a fractal, which in turn is part of a larger fractal cell, and so on; all of whom are sharing similar information in similar ways, Section 4.1.
Consequently, in representing this architecture in the form of a system meta-map, it is better to consider it as a multidimensional temporal ‘honeycomb’. Where cells are joined together to provide a specific function, and then subsequently break-up when their purpose has been served to form the basis of others, Figure 24.

6.4 Analysis Summary

This section has assessed the literature survey presented by this thesis in terms of the effectiveness of the approach employed; as well as the ‘results’ in the form of the advantages and disadvantages of the proposed architecture, Section 6.1 and 6.2. The latter demonstrated how the fractal architecture in conjunction with the GA operating system created a highly resilient system, and so addressed the original research question, Section 1.2.

Using references back into the main body, the analysis highlighted how individual cells interacted to form a heterogeneous complex adaptive system, whereby the operating system (code components and data) were both created, and were simultaneously a result of emergent behaviour. Consequentially, there is no fixed architecture, rather cells joined together temporally providing specific functions for a period of time, before disassociating to form resource for other more effective / appropriate fractals. This has implications for representing the architecture; as by definition, it is situational / temporal dependent and it is not possible to predict its overall structure from its constituent parts, Section 6.3. Hence, two figures were presented for the system meta-map, showing the architecture for both a single, and a cluster of cells perspective.
7 Conclusions

This thesis is based upon an extensive literature review, and asked whether it was possible for a fractal based system to achieve graceful degradation; i.e., to continue to provide some service, reducing its commitments according to a previously agreed priority of functionality, (Section 1.2.2)?

While a literature synthesis of this nature is, by definition qualitative; the research outcomes indicate that it should be possible to design and build a fractal-based system capable of operating in high availability / reliability scenarios through dynamic reconfiguration of itself, and thus assure graceful degradation of service in the event of a failure.

Throughout the document, a number of key points have been highlighted, which demonstrate the proposed argument’s validity, or represent areas of further research. These points have been used to construct an architecture, which has been represented in two meta-system maps, Section 6.3.

7.1 Summary

Starting with a definition of resilience, the thesis has identified that the ability of any system to respond to a failure is governed by its architecture, Section 3. As computer architecture and hence design, have not changed essentially since they were initially developed, this has been identified as a major contributor to a lack of graceful degradation; Section 3.5.

By showing the feasibility of an alternative design, which used collaborating GAs in reconfigurable components in a fractal-based architectural layout; it was possible to demonstrate robustness with respect to resilience. This was achieved by identifying
different failure mechanisms, and how a system subsequently responded; Sections 3.2 and 3.4. The approach also focused on dynamic reconfiguration and the reassigning of functionality, as a potential recovery mechanism; Section 3.3.

A more in depth literature search proposed a potential solution, which addressed the limitations of traditional computer design, Section 4. Although several alternative architectures and operating systems were considered, Sections 4.1 and 4.3; the investigation developed a solution, weaved together several key concepts regarding fractals, GAs and reconfigurable components. This allowed an initial system configuration with partial ‘autopoiesic’ characteristics, to become a ‘heterogeneous complex adaptive system’ capable of self-repair, self-configuration, self-optimisation and self-protection in response to adverse (and beneficial) operational and environmental changes; Section 4.4. Furthermore, ‘new’ concepts such as system immunity and solution ‘bio-diversity’ were discussed, as a means of addressing potential faults and novel failure modes, which were introduced by the GA fractal-based approach; Section 4.5.

Section 5 discussed the properties of FPGAs and GA for creating a resilient fractal based architecture together with an adaptive operating system, Sections 5.1 and 5.2. This was subsequently considered at the physical level using such features as dynamic reconfiguration, mutual interaction, and re-tasking; where it was determined that a fractal system could mimic a conventional system, Section 5.3. These same properties enable a service level to be maintained in the presence of multiple failures, i.e. a resilient system.

Both the approach and the overall methodology were assessed in Section 6. The capability of a literature review as a research tool was question in Section 6.1, while Section 6.2 considered the advantages and disadvantages of the proposed approach; and its overall effectiveness against a conventional architecture. Finally,
'architecture maps' were produced which highlighted the heterogeneous complex adaptive nature of the proposed system, Section 6.3. It was established that the architecture is temporary in nature, continuously adapting to both its internal and external environment. This is highlighted by the service provision, which can not be linked to a specific architectural aspect; rather it is a product of the system’s emergent properties.

### 7.2 Additional Observations

Existing research typically focuses on a specific area, often reflecting grant availability, and specific development trends; this has been demonstrated by the numerous citations that have been used, and which cut across many areas in order to provide a holistic view. Although, using multiple citations facilitates originality, it has also allowed a number of direct observations; Section 6.1:

- The approach has major architecture implications, as (parts of) the design and its physical implementation will undergo regular minor and occasional major ‘extinction events’; to be replaced by self-determined functionality that addresses both immediate and long-term survivability requirements. As a consequence, functionality is now transient, reflecting a ‘fitness landscape’ determined by the operating system GA; Sections 4.2 and 4.4;

- The architecture exhibits emergent complex behaviour whereby it is changed by, and so in turn changes the environment. Hence, while its formation and operation can be determined through observation over the immediate short-term future, only approximate behaviour and structures can be anticipated over the longer term; Section 6.3;
• The operating system consists of code components and data parameters in the form of a ‘double helix’, which are continuously evolving in relation to both the environment and each other, Section 4.4;

• Increasing the overall availability of a given system will decrease the need for additional redundant components; this will have a significant impact on the design, and cost of a system. Hence this work will have direct consequences for the way designers look at a number of applications across several different industries. In particular, those involved in the design of safety critical applications such as nuclear controls and aircraft, as well as high availability commercial systems e.g. telecommunications.

Indirectly, this approach may potentially result in the reduction of the overall number (and diversity) of components used in a given design, which would be of a major interest to the engineering and manufacturing communities at large:

• Effectively simplifying the internal architecture will result in a significant overall (long-term) cost decrease through manufacturing gains, and by decreasing the number and variety of retained spares, i.e. logistical gains;

• Reduction and simplification will result in weight savings, a major design criteria in many industries such as space and aerospace;

• The general economic impacts on various manufacturing supply chains.

A further consequence of simplification, is that design standardisation coupled with increased formalisation, will further improve availability through an ongoing reduction in the overall number of design errors; while simultaneously, reducing the overall design time.

Longer term usage and benefits can also be gained, when the ‘service provider’ concept for an individual computer is considered in the context of a number of
standardised ‘chromosomes’ utilised in the GA based operating system. This in and of itself, will result in a re-examination of computer design and use.

7.3 Areas of Future Research

Previous sections have qualitatively demonstrated that a GA based operating system using a fractal architecture consisting of FPGAs, could provide a highly resilient system. Furthermore, a number of key observations were made regarding the design and specific system properties, Section 7.2. These observations, together with a number of aspects which have suggested themselves through the course of the thesis, should form the foundation of a future research programme.

The proposed research would centre around three main themes, which together, could demonstrate quantitatively through experimental data, the effectiveness of the approach as a means of providing a viable resilient architecture:

- **The formation of fractals and their ability to be reconfigured readily to provide (new) service offerings.** This would demonstrate the viability of the basic physical operational structure, and Tharumarajah’s (1996) original concepts. Also, whether those proposed by Ryu and Jung (2003) could be readily translated into a GA based operating system, Section 4.2;

- **The structure and ease of formation of a self-initialising ‘double helix’ GA based operating system.** This will require the formation of machine code level ‘base pairs’ / allele, using a variation of component based software, which could be tested and implemented on individual IC’s, Nordin et al (Accessed 09th October 2006), Section 4.4. This would assert the practicality of ‘allele’ software formation, as potentially many products of this process should not be carried forward into subsequent generations (the offspring are not viable or ‘dangerous’);
• **The emergence of high-level equivalent services from the interaction of discrete cells.** This would demonstrate the potential of heterogeneous complex adaptive systems, as identified by Stacey (2003); to provide a viable architectural construct, Section 6.3. As such, this provides the over-arching integration between the physical structure (i.e. fractals) and the operating system, demonstrating the effectiveness of emergent resilient equivalent services.

### 7.3.1 Suggested Tools and Devices

There are currently multiple FPGAs commercially available, e.g. Sklyarov (2002) referenced the XC6200 family; however as Actel (2002) identified, they have different capabilities in different scenarios. While a comparison would be required, and undoubtedly, some aspects of a particular specification will be more appropriate than others, e.g. ease of programming or repair; the research will initially only require a single device. Subsequently, additional devices and physical assets etc can be added as required, to simulate a true operational environment, in order to demonstrate potential emergent properties.

There are also several genetic programming packages on the market e.g. GP studio (BridgerTech, 2006) and Dicipulus Linear Genetic Programming Software (Francone, 2001).

Although Dicipulus’ high level interface enables machine code level modification; and this would allow the most opportunity to change the FPGA at the code level, (Nordin et al, Accessed 08th October 2006). While this may be appropriate for initial idea development, the thesis proposes manipulation of both the operating code to form new instruction sets, and the associated data (i.e. the ‘double helix’). Hence, either another product may need to be identified, or appropriate software developed.
7.3.2 Proposed Research Programme

Any proposed programme would need to address one or more (integrated) themes as identified above; however, the general outline is shown in Figure 25:

Figure 25: Proposed Future Research Programme

Either in addition or subsequent to the main programme, further work would be required to assess the effectiveness and efficiency of the approach in terms of response times to environmental changes; and its ability to 'mimic' the current functional service provided by existing systems. This could lead to the use of fractal based systems being employed where there are cost and weight issues associated with 'conventional' designs.
Similarly, as the self-adapting emergent properties of the system could have a significant impact upon future computer architectural development; this would also require further investigation; Section 6.3. Similarly, the thesis also identified a number of novel scenarios, e.g. ‘GA Cancer’, which would also require investigation, Section 6.2.

7.3.3 Additional Research Areas

In addition to the main theme, other potential research areas which could be incorporated include, but are not limited too:

- **Service Equivalency**: can the proposed architecture provide all the services of an existing computer or are there exceptions;

- **Efficiency**: does the architecture meet current response times and capabilities provided by existing designs;

- **Resilience**: to what extent can the system degrade before service loss occurs;

- **Security**: Does the architecture create changes in current security issues;

- **Failure Mechanisms**: Does the proposed architecture have a unique set of failure modes, and if so are these better or worse than current designs;

- **FPGA Design**: Is there an optimal FPGA design, or are several variations required due to implementation or design limitations;

- **GA Design**: What are the consequences of having fully autonomous GAs interacting within a system, and would it be considered acceptable;

- **Cell Formation and Interaction**: what is the most appropriate design for cell formation and are their limitations in terms of the number of interacting cells;

- **Design Limitations**: What are the limitations in the context of task implementation, and would a hybrid design be appropriate in some contexts?
References


McGraw Hill online encyclopaedia:

http://www.accessscience.com.libezproxy.open.ac.uk/server-java/Arknoid/science/AS/ Accessed 14/05/06


SAE (April 1996) *Certification Considerations for Highly-Integrated or Complex Aircraft Systems*, ARP 4754, Systems Integration Requirements Task Group As-1C, ASD, SAE


*Computers in Industry* Vol 51 pp 185-196, Elesvier.


Index

A
adaptive · 7, 24, 32, 41, 48, 49, 52, 56, 75, 88, 90, 91
allele · 54, 93
autonomic · See autonomy
Autonomic · See autonomy
autonomy · 40, 41
autopoietic · See autopoiesis
autopoiesis · 41
B
bionic · 15, 37, 49, 61
Bionic · 13, 104
C
chromosomes · 55, 93
D
deadlock · 33, 67
degradation · 6, 9, 10, 11, 12, 15, 17, 27, 36, 63, 64, 82, 89
Degradation · 1, See degradation
degrade · See degradation
DNA · 46, 50, 51, 55, 57, 58, 59, 60, 61
Dynamic · 13, 34, 74, 97, 98, 101
E
environmental · 7, 8, 12, 22, 23, 30, 35, 36, 38, 48, 53, 56, 57, 58, 59, 65, 66, 67, 68, 77, 82, 95
epigenetic · 36, 41, 46, 69
evolution · See evolutionary
everolutionary · 6, 11, 52, 58, 69
Evolutionary · See evolutionary
F
failure · See failures
Failure · See failure
failures · 6, 9, 12, 15, 20, 21, 22, 28, 29, 32, 33, 35, 38, 44, 48, 60, 61, 64, 67, 69, 71, 74, 78, 90
Failures · See failures
fault · See failures
Field Programmable Gate Arrays · See FPGA
FPGA · 11, 15, 34, 36, 49, 63, 67, 96, 99, 100
fractal · 11, 12, 15, 17, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 52, 55, 58, 59, 60, 61, 62, 64, 65, 66, 67, 68, 69, 70, 72, 73, 74, 75, 77, 78, 83, 89, 90, 95
fuzzy · 30, 38
G
GA · 11, 15, 17, 36, 46, 50, 51, 52, 53, 54, 55, 56, 57, 59, 60, 61, 62, 63, 64, 65, 66, 68, 69, 73, 74, 75, 77, 78, 82, 83, 90, 96
Genetic Algorithms · See GA
graceful · 6, 11, 12, 15, 17, 63, 82, 85, 89
H
heterogeneous · 7, 41, 56, 85, 88, 90, 91
holonic · 15, 37
homogeneous · 76, 85
I
instability · 86
J
M
mutation · 51, 55
O
offspring · 53, 54, 55, 59
ontogenetic · 36, 47, 57
operating system · 6, 15, 24, 36, 50, 51, 52, 58, 61, 62, 64, 67, 69, 77, 78, 82, 83, 90, 93
P
phylogenetic · 36, 52
phylogentic · 55
propagation · 59, 68
R
re-assign · 65
reconfiguration · 10, 12, 15, 32, 33, 34, 37, 41, 48, 63, 65, 67, 68, 71, 74, 75, 78, 82, 89, 90
Reconfiguration · See reconfiguration
rejuvenation · 64, 65
Resilience · 13, 17, 35, 62, 70, 96
resilient · See Resilience
resource · 12, 15, 24, 25, 26, 36, 43, 44, 45, 46, 48, 50, 57, 59, 63, 66, 68, 69, 71, 74, 75, 77, 78
re-task · 65, 68
re-tasking · 10, 12, 58, 67, 90

S
self-configuration · 6, 90
self-configure · 71
self-optimisation · 6, 49, 90
self-optimise · 71
self-protection · 6, 71, 90
self-reference · 40
self-repair · 6, 33, 48, 71, 72, 74, 78, 90

survivability · 51, 59, 68

T
tolerance · 22, 23, 29

V
variation · 52, 55
vitality · 46, 49, 58