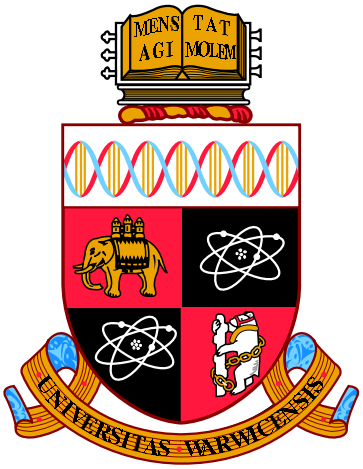


Error Management in ATLAS TDAQ: An Intelligent Systems approach

by

John Erik Sloper

Thesis

A dissertation submitted in partial fulfilment of the

requirements for the degree of

Doctor of Philosophy

School of Engineering

November 2010

THE UNIVERSITY OF
WARWICK

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

I would like to thank the people at the School of Engineering, University of Warwick, for their assistance during my period of research. I would specially like to thank my supervisor, Dr. Evor. L. Hines for his support and guidance. I would also like to thank Dr Jianhua Yang for our fruitful cooperation.

From CERN I would like to thank all my co-workers for the discussion and cooperation over the years. A special thanks goes to my supervisor at CERN Dr. Giovanna Lehmann-Miotto for giving me the freedom to complete my thesis even though some of the work did not have immediate impact for CERN.

I would also like to express my utmost appreciation of my brother and my parents for their unwavering support and understanding, regardless of any successes or failures in my research work.

Finally, special gratitude goes to my wife Itana for her continuous support and understanding despite being occupied with and completing her own PhD thesis.

Declarations

This thesis is presented in accordance with the regulations for the degree of doctor of philosophy. All work reported has been carried out by the author unless otherwise stated, including the production of this document. No part of this work has been previously submitted to the University of Warwick or any other academic institution for admission to a higher degree. All publications to date arising from this thesis are listed in the next section.

List of Authors publications

Journal Papers:

Kazarov, A.; Corso-Radu, A.; Miotto, G.L.; Sloper, J.E.; Ryabov, Y., A Rule-Based Verification and Control Framework in Atlas Trigger-DAQ, *IEEE Transactions on Nuclear Science* , vol.54, no.3, pp.604-608, June 2007

Sloper, J.E.; Miotto, G.L.; Hines, E., Dynamic Error Recovery in the ATLAS TDAQ System, *IEEE Transactions on Nuclear Science*, vol.55, no.1, pp.405-410, Feb. 2008

Sloper, J.E.; Leahu, M.; Dobson, M.; Lehmann, G., Access management in the ATLAS TDAQ, *IEEE Transactions on Nuclear Science*, vol.53, no.3, pp. 986-989, June 2006

Conference Papers:

Sloper, J.E; Hines, E.L., Detecting errors in the ATLAS TDAQ system: A neural networks and support vector machines approach, *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA),* 2009

Poy, Alex Barriuso; Sloper, John Erik; Khomutnikov, Viatcheslav; Miotto, Giovanna Lehmann, Operation of the ATLAS Experiment: Organization of the Detector Controls and the Data Acquisition System, *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics* , vol., no., pp.190-194, Nov. 2006

Almeida, J.; Dobson, M.; Kazarov, A.; Miotto, G.L.; Sloper, J.E.; Soloviev, I.; Torres, R., The ATLAS DAQ System Online Configurations Database Service Challenge, *2007 15th IEEE-NPSS Real-Time Conference* , vol., no., pp.1-8, April 29 2007-May 4 2007

Book chapters:

Sloper, J.E.; Miotto, G.L.; Hines, E., Dynamic Error Recovery in the ATLAS TDAQ System: Using Multi-Layer Perceptron Neural Networks E. Hines, M. Leeson, M. M.Ramn, M. Pardo, E. Llobet, D. Iliescu, and J. Yang, *Intelligent Systems: Techniques and Applications.* Shaker publishing, 2008, ch. 8, pp. 241268.

Abstract

This thesis is concerned with the use of intelligent system techniques (IST) within a large distributed software system, specifically the ATLAS TDAQ system which has been developed and is currently in use at the European Laboratory for Particle Physics(CERN). The overall aim is to investigate and evaluate a range of ITS techniques in order to improve the error management system (EMS) currently used within the TDAQ system via error detection and classification. The thesis work will provide a reference for future research and development of such methods in the TDAQ system.

The thesis begins by describing the TDAQ system and the existing EMS, with a focus on the underlying expert system approach, in order to identify areas where improvements can be made using IST techniques. It then discusses measures of evaluating error detection and classification techniques and the factors specific to the TDAQ system.

Error conditions are then simulated in a controlled manner using an experimental setup and datasets were gathered from two different sources. Analysis and processing of the datasets using statistical and ITS techniques shows that clusters exists in the data corresponding to the different simulated errors.

Different ITS techniques are applied to the gathered datasets in order to realise an error detection model. These techniques include Artificial Neural Networks (ANNs), Support Vector Machines (SVMs) and Cartesian Genetic Programming (CGP) and a comparison of the respective advantages and disadvantages is made.

The principle conclusions from this work are that IST can be successfully used to detect errors in the ATLAS TDAQ system and thus can provide a tool to improve the overall error management system. It is of particular importance that the IST can be used without having a detailed knowledge of the system, as the ATLAS TDAQ is too complex for a single person to have complete understanding of. The results of this research will benefit researchers developing and evaluating IST techniques in similar large scale distributed systems.

List of Abbreviations

AD	Antiproton Decelerator
ALICE	A Large Ion Collider Experiment
ANN	Artificial Neural Network
ATLAS	A Toroidal LHC ApparatuS
BMU	Best Matching Unit
BPA	Back Propagation Algorithm
CERN	European laboratory for Particle Physics (Conseil Europen de la Recherche Nuclaire)
CGP	Cartesian Genetic Programming
CMS	Compact Muon Solenoid experiment
CNGS	CERN Neutrinos to Gran Sasso
CPU	Central Processing Unit
CSC	Cathode Strip Chamber
DFM	Dataflow Manager
EB	Event Builder
EBN	Event Builder Network
EFD	Event Filter Dataflow
EFN	Event Filter Network
EM	Error Management
EMS	Error Management System
ERS	Error Reporting System
ES	Expert System

FCM	Fuzzy C-Means
FN	False Negative
FP	False Positive
FSM	Finite State Machine
GA	Genetic Algorithm
GNMM	Genetic Neural Mathematical Method
HFC	Hierarchical Fair Competition
IPC	InterProcess Communication
IS	Information Service
ISOLDE	Isotope Separator on Line
IST	Intelligent System Techniques
KB	Knowledge Base
L2N	Level 2 Network
L2PU	Level 2 Processing Unit
L2SV	Level 2 Supervisor
LAr	Liquid Argon
LEIR	Low Energy Ion Ring
LHC	Large Hadron Collider
LHCb	The Large Hadron Collider beauty experiment
LINAC	Linear Accelerator
MDT	Monitored Drift Tube
ML	Machine Learning
MLP	Multi Layer Perceptron
MRS	Message Reporting System
n-TOF	neutron Time Of Flight
NFLT	No Free Lunch Theorem
NPV	Negative Predictive Value
OAA	One-Against-All

OAO	One-Against-One
OSH	Optimal Separating Hyperplane
PCA	Principal Component Analysis
PMG	Process ManaGer
PNN	Probabilistic Neural Network
PPV	Positive Predictive Value
PS	Proton Synchrotron
PT	Processing Task
QP	Quadratic Programming
RBES	Rule Based Expert System
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
ROD	Readout Driver
ROIB	Region Of Interest Builder
ROS	ReadOut System
RPC	Resistive Plate Chamber
SCT	Semiconductor Tracking detector
SFI	Sub-Farm Input
SFO	Sub-Farm output
SGP	Standard Genetic Programming
SNR	Signal to Noise Ratio
SOM	Self Organised Map
SPS	Super Proton Synchrotron
SSE	Sum-Squared Error
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TDAQ	Trigger and DataAcQusition
TDNN	Time delayed Neural Network

TGC	Thin Gap Chamber
TN	True Negative
TP	True Positive
TRT	Transition Radiation Tracker

Chapter 1

Introduction

This thesis is concerned with investigating how Intelligent System Techniques (IST) may be applied to improve the overall Error Management (EM) capabilities within a large distributed computing system, specifically the ATLAS TDAQ system currently in use at the European Laboratory for Particle Physics (CERN). A number of different techniques were used in order to create error detection systems based on data available from different sources in the TDAQ system. The results of these error detection systems are then analysed and different ways in which to utilise those results are investigated. This could potentially produce a better understanding of the system which again enables better EM systems to be built.

Section 1.1 provides a short background to the ATLAS TDAQ system and the context of the thesis work. Section 1.2 then provides an introduction to the problem before the thesis objectives are described in Section 1.3. Section 1.5 describes the contributions to knowledge made throughout the thesis.

1.1 Background

This section provides a very brief introduction to CERN, the LHC and the ATLAS TDAQ system.

1.1.1 CERN (Conseil Europen pour la Recherche Nuclaire)

CERN was originally established as a provisional body in 1952. After the actual foundation of CERN in 1954 by 12 European states, the provisional council was

dissolved, but the original acronym was kept. In its early days the research was concentrated on nuclear physics, but has since then focused increasingly more on particle physics and is now commonly referred to as the "European Laboratory for Particle Physics". Today 20 European member states collaborate to run CERN, although contributions are made from countries around the world, including USA, Russia, Japan and China among others, making it a truly global collaboration. In addition to the approximately 2500 staff permanently at CERN, more than 8000 scientists visit and perform part of their work at CERN.

Currently, the main effort at CERN is to operate and maintain a new particle accelerator, namely the Large Hadron Collider (LHC). The LHC is currently in operation and has already broken previous records by similar experiments.

1.1.2 CERN and the Large Hadron Collider (LHC)

CERN hosts a number of different experiments, both theoretical and experimental although the main efforts are concerned with the construction and operation of particle accelerators. CERN contains a number of accelerators where particles are successively accelerated to higher and higher energies as shown in Figure 1.1 where the last stage of this acceleration is the Large Hadron Collider (Evans, 2007).

The LHC accelerates protons and lead ions to higher energies than any previous experiment, and will eventually collide particles every 25ns with a centre-of-mass energy of 14 TeV. This makes it possible to study physics phenomena that have previously never been observed in a controlled experimental environment.

Four main experiments are connected to the LHC; ATLAS, CMS, LHCb and ALICE. An overview of the LHC and the location of the 4 experiments is shown in Figure 1.2 (please refer to Table 1.1 for the complete list of abbreviations). Each of the 4 experiments are designed to fulfill specific goals. The Alice experiment is concerned with studying lead-ion interactions while the LHCb experiment is concerned with studying matter and anti-matter. The ATLAS and CMS are both general purpose detectors designed to cover the widest possible range of physics phenomena. While the two experiments have the same goal in mind, they are built using different

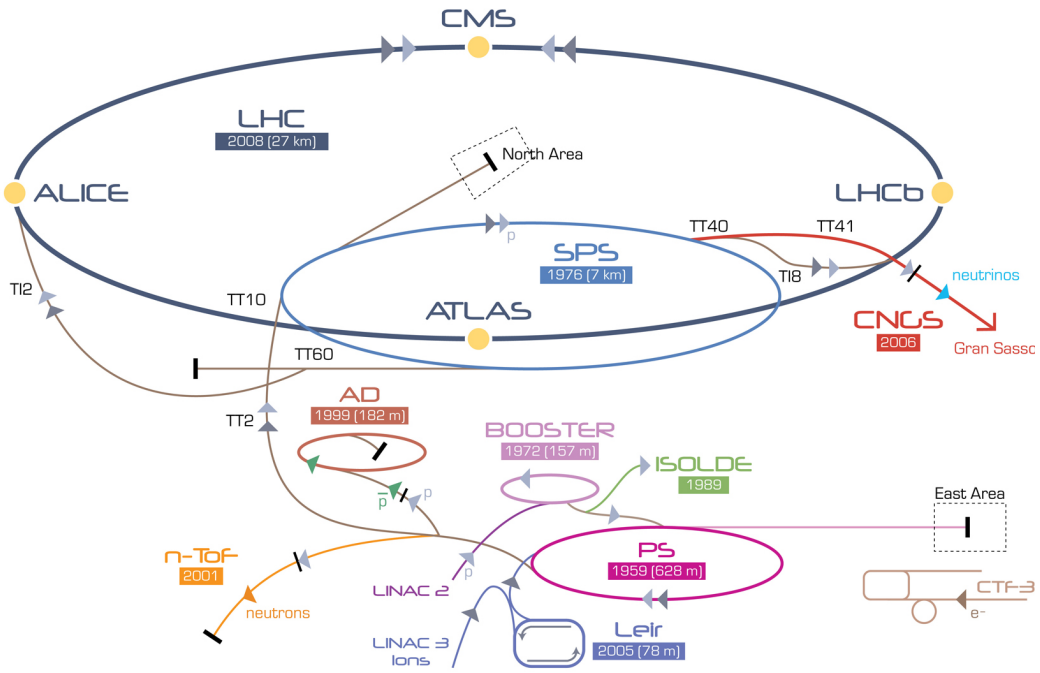


Figure 1.1: Schematic overview of the main accelerators at CERN.

technical solutions and design.

The LHC itself is located in a 27km tunnel 100m underground on the border of Switzerland and France (Figure 1.2). It was first switched on the 10th of September 2008 and the initial operation was considered a huge success. Unfortunately, after just a short time in operation complications arose and it was shut down for repairs. The main repairs were finished during the summer 2009 and the LHC is at the time of writing (April 2010) operational. Initially, it will be operating at a relatively 'low' energy to ensure the safe operation of the LHC and the collision energy will only later be increased to its full potential of 14 TeV.

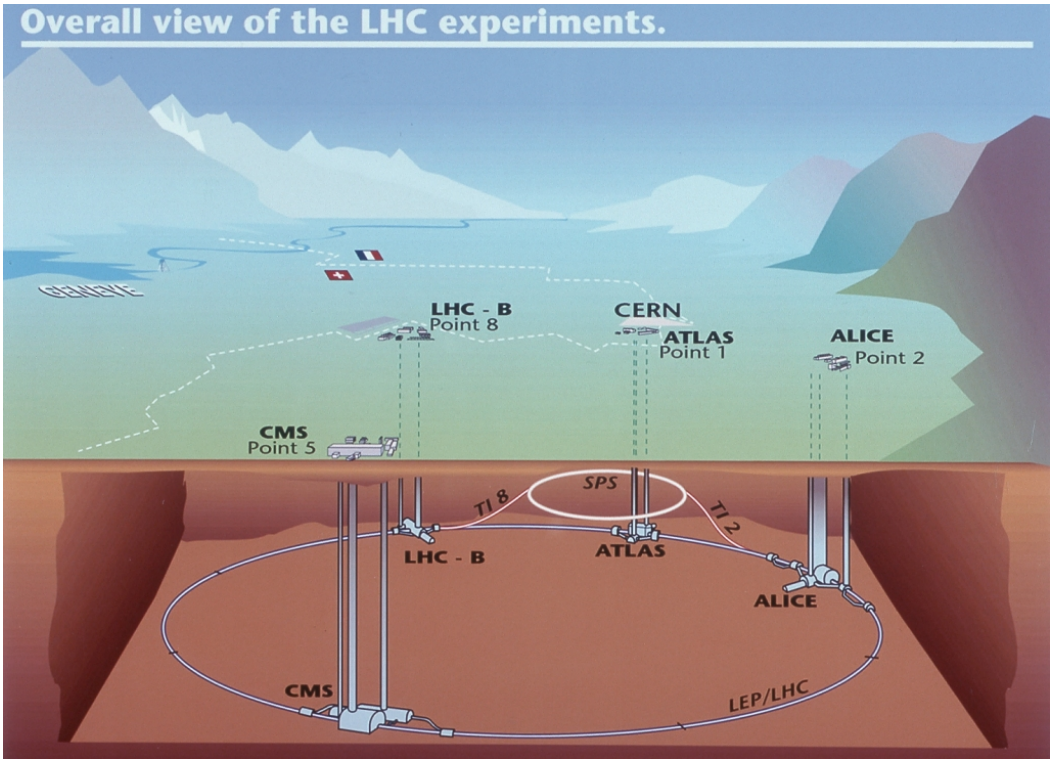


Figure 1.2: Overview of the main LHC experiments. Please refer to Table 1.1 for the list of abbreviations.

Table 1.1: Abbreviations of the LHC related experiments.

Abbreviation	**Full name**
ATLAS	A Toroidal LHC ApparatuS
ISOLDE	Isotope Separator on Line
AD	Antiproton Decelerator
PS	Proton Synchrotron
LEIR	Low Energy Ion Ring
LINAC	LINear ACcelerator
n-TOF	neutron Time Of Flight
CNGS	CERN Neutrinos to Gran Sasso
SPS	Super Proton Synchrotron
ALICE	A Large Ion Collider Experiment
CMS	Compact Muon Solenoid experiment
LHCb	The Large Hadron Collider beauty experiment

1.1.2.1 A Toroidal LHC Apparatus (ATLAS)

ATLAS is the largest particle detector ever built and its scope is to determine which particles are produced during collisions of the high energy protons accelerated in the LHC. ATLAS is at the very forefront of particle physics research and incorporates a large number of custom hardware and software modules. It consists of a large cylinder (43m length x 25m diameter) of detecting devices, placed around one of the beam collision points of the LHC. The ATLAS is built and funded by the ATLAS collaboration, which consists of approximately 2500 scientists originating from 37 different countries and 169 universities.

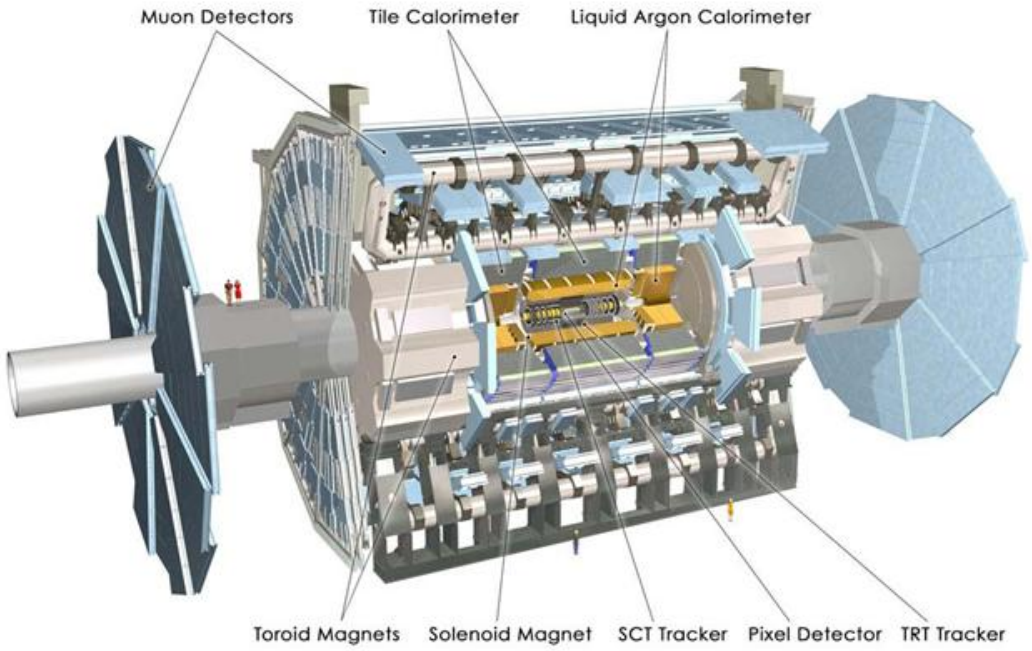


Figure 1.3: A schematic view of the ATLAS detector.

The work for this thesis has been conducted within the context of the Trigger and DataAcQusition (TDAQ) system of the ATLAS experiment. The TDAQ system is described in more detail in the following section.

1.1.2.2 The ATLAS TDAQ system

The ATLAS TDAQ system is a large heterogeneous system consisting of a wide variety of software and hardware components. Its overall goal is the gathering of 'interesting' physics data from the ATLAS detector. The TDAQ system is based on three levels of on-line triggering/filtering of the data; Level-1 trigger, Level-2 trigger and Event filter. Each level will further refine the results of the previous one only keeping those parts of the data which may be of interest for further analysis. In total the data is filtered from 40 MHz at the detector level to 300Hz at the output of the system. The amount of data is reduced from 10s of TB/s to 100s of Mb/s (the exact rate depends on a variety of factors such as the operational energy of the LHC and the configuration of the TDAQ system). The whole system comprises approximately 3000 computers ranging from 'off-the-shelf' 8-core processing nodes to computers containing custom made hardware modules linked directly to the ATLAS detector. The majority of the nodes take part in the filtering of the data. A schematic overview of the different parts of the TDAQ system is shown in Figure 1.4.

A detailed description of the design and implementation of the TDAQ system can be found in (Collaboration, 2003) and (Vermeulen and et al., June 2006).

1.2 Introduction to the problem

While considerable time and effort have been devoted to the design and development of the ATLAS TDAQ system, there has not been any significant effort devoted to the area of error detection, handling and prevention; collectively referred to as error management. As the system has moved increasingly into an operational phase the need for a global error management system has become evident and automatic error detection and handling have become increasingly important and is therefore receiving more and more attention by the system designers and developers. Due to the very high costs of operation of the TDAQ system, both economically and in terms of manpower, reducing the amount of 'downtime', here meaning the time when the system is not performing its main tasks at its full potential, is very important.

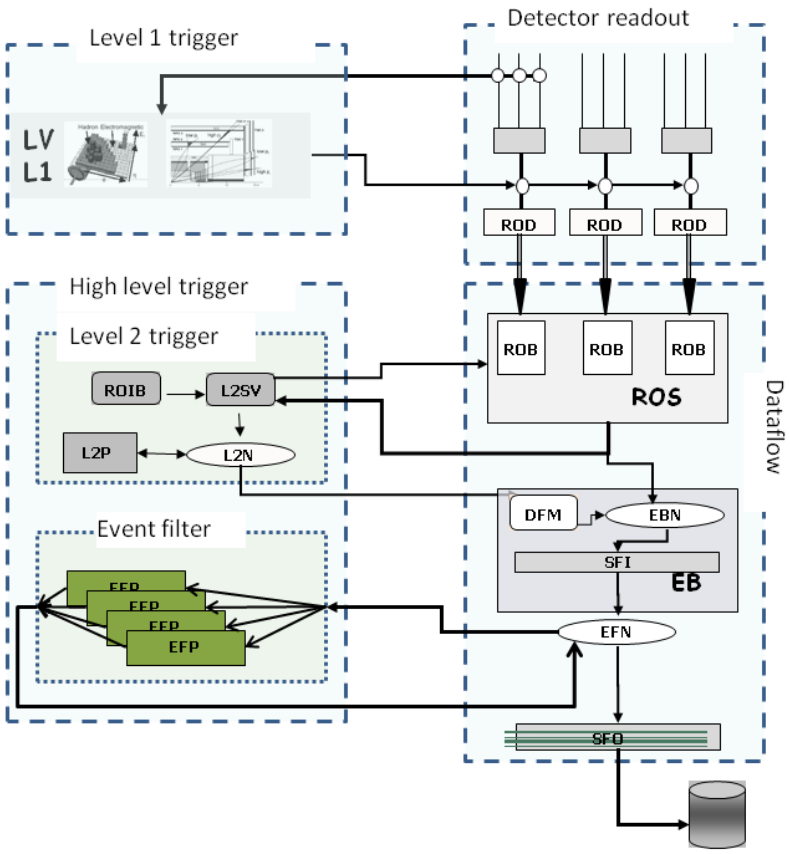


Figure 1.4: Overview of the ATLAS TDAQ system architecture.

The total downtime consists of two periods of time:

Time-to-detection: The time from when an error occurred in the system until the error is detected. This clearly depends on the effectiveness of the error *detection* system available, the skill of the human operator involved or a combination of the two.

Time-to-recovery: The time from when the error is detected until appropriate actions have been performed and the system has been restored to a functional state. This also depends on the available recovery system and/or on the skill of the human operator.

The overall goal is to reduce the downtime as much as possible. This can be achieved in three main ways:

(i) Early detection of errors: By detecting the errors as soon as possible the time from when the error occurred until it is detected can be reduced. This enables the system to perform the recovery measures sooner and thus reduces the overall downtime.

(ii) Fast and efficient handling of and recovery from errors: By reacting as swiftly as possible to errors after they have been detected the down-time will be reduced. This can also reduce the likelihood of further errors occurring as a result of the initial one(s).

(iii) Preemptive measures to try to reduce the number of errors occurring in the system: If a thorough understanding of the causes of an error can be determined, it might be possible to prevent the error from happening. This can include not only changes to applications and the system configuration, but also early warning systems indicating situations where errors could occur (for example where network or other infrastructure is severely strained).

1.3 Thesis objectives

The overall objective of the thesis is to investigate ways of improving the top-level EM system currently used in the TDAQ system. The main focus is on error detection as this forms the basis for both error recovery and preemptive measures. Error detection in a system such as the ATLAS TDAQ is very difficult due to a number of factors including:

(i) The very complex nature of the system means that no one person can gain a full understanding of all the sub-systems let alone all the interconnections and dependencies between the different parts.

(ii) The constant ongoing development of the system means that detection techniques must constantly be adapted in order to meet new requirements and/or the changing behavior of the system.

(iii) Naturally, a large number of data sources exist in such a system, but it is not clear which parts are the most useful or how to best process this data. Identifying and extracting useful parts of the data that may contain valuable information about the system is therefore also of key importance for both error detection and handling, and to realise a better understanding of the system as a whole.

The approaches considered in this thesis may be divided into three parts and focus mainly on researching and developing new ways of detecting and understanding the errors that may occur in the TDAQ system. The strategies which will be applied to achieve the objectives are to:

(i) Develop techniques to help understand and visualise the data available from the system to both be able to predict its behavior and to better understand error situations that have arisen or may arise. The first step is to identify which parts of the available data sources in the system are of interest in the context of error detection.

(ii) Develop methods for error detection. A number of different IST techniques will be investigated and compared to each other in order to determine which are the most suitable to be utilised in the TDAQ system. These methods should ideally be easily repeatable and adaptable to any changes that are made to the system.

(iii) Extract knowledge from the developed error detection techniques. This could potentially be in the form of rules to be incorporated in the existing expert system as described in Chapter 3. Extracting knowledge is fundamental to both developing error handling and recovery techniques (one must understand *why* the problem happens to be able to fix it) and to be able implement preemptive measures.

1.4 Thesis outline

Chapter 2

Provides an overview of the ATLAS TDAQ system in order to give the reader background information to better understand the choices made in subsequent chapters. It describes the different components of the TDAQ that are of relevance to EM systems.

Chapter 3

The existing expert system approach used in the TDAQ system is presented and its overall design and implementation is described. The limitations of the system are discussed and the areas best suited for improvements are identified together with ways of extending and improving the system. The expert system is at the heart of the TDAQ EM and is likely to be operational for years to come. It is therefore important that the reader has an adequate understanding of the system as any techniques investigated in this thesis must be integrated to it.

Chapter 4

Provides an overview of the main factors concerning error detection in the TDAQ system and presents different ways to measure effectiveness of such error detection systems. These measures are used in order to evaluate detection methods in subsequent chapters.

Chapter 5

Here, the experimental setup which was used for the work in this thesis is described. It includes a description of the applications taking part in the test setup and its topology. It also describes how errors were simulated in the system and how data was gathered and stored for subsequent analysis. The datasets are then visualised and analysed in this chapter. The analysis is performed in order to evaluate whether error detection techniques can be applied based on the available datasets. The visualisation can also be used to gain insight into the system and thus contributes to

improving the overall error management process.

Chapter 6

In this chapter a number of different artificial neural networks (ANNs) are described and are then applied to detect errors based on the previously gathered datasets. The results of the different network types are discussed and compared. An ensemble of the ANNs is then designed and implemented using a genetic algorithm (GA) approach. The results are then compared to the results achieved using single ANNs. Ultimately, the use of a new technique for variable selection, namely the genetic neural mathematical method (GNMM), is applied in order to reduce the complexity of the datasets and thereby making it easier for the ANNs to process the data.

Chapter 7

This chapter introduces support vector machine (SVM) theory and applies SVMs using the same datasets as in the previous chapter. Different kernels are tested and the results are compared to the ANN results. The advantages and disadvantages of SVM are then discussed and compared to the ANN approach.

Chapter 8

Here, a new evolutionary programming approach is developed and investigated. Cartesian genetic programming (CGP) is presented together with a comparison to standard genetic programming (SGP). It also investigates the usefulness of the Hierarchical Fair Competition (HFC) evolutionary model used together with CGP. A new form of crossover is also presented before CGP is then applied in ordered to evolve a program to perform error detection. The program is analysed and compared to the previously achieved results and the advantages and disadvantages of this approach are discussed.

Chapter 9

In this final chapter the results are summarised, discussed and the final conclusions

are made. Possible future work is also discussed.

1.5 Contributions to knowledge

This thesis aims to provide significant contributions to knowledge by showing that IST can be used to successfully detect and classify errors within the ATLAS TDAQ system. Such techniques have not previously been utilised within the TDAQ system nor previous system of similar design. In particular, contributions have been made in the following ways:

- Clustering and visualisation of data gathered from several sources in the ATLAS TDAQ system provides further insight into the system in the context of error detection and management. The techniques used in this thesis have not previously been applied within the context of the TDAQ system and the datasets analyzed have previously not been utilised within the context of error management.

- The expert system approach described in Chapter 3 provides a novel approach to error handling and error management in the ATLAS TDAQ and similar systems and the results have lead to publications in peer reviewed journals(Poy et al., Nov. 2006; Sloper et al., 2008; Kazarov et al., June 2007).

- A number of different ISTs have been investigate in order to develop error detection systems based on data available in the ATLAS TDAQ system. The thesis shows that these techniques can be successfully applied and that they can produce prediction and classification of errors with a high level of accuracy. These techniques therefore provide further 'tools' for developers aiming to improve the error detection capabilities of systems such as the ATLAS TDAQ. A paper describing the application of a subset of these techniques was published in (Sloper and Hines, 2009).

- Also, the use of HFC together with CGP has not previously been investigated prior to this thesis. In addition, a new form of crossover has been developed

and is shown to provide better results for a number of problems compared to other crossover techniques.

It is the opinion of the author that the work reported here represents a 'significant and original contribution to knowledge'.

1.6 Conclusions

This chapter has provided a background to the workplace where the thesis work was conducted in order to put it into the proper context. The problem that the thesis investigates has been introduced and the thesis methodology and objectives have been presented. The outline of the thesis which was then presented should give the reader an overview of the thesis as a whole.

References

Atlas Collaboration. Atlas high level data acquisition and controls - technical design report. Technical report, CERN, Geneva,, 2003.

Lyndon Evans. The large hadron collider. *New Journal of Physics*, 9(9):335, 2007.

A. Kazarov, A. Corso-Radu, G.L. Miotto, J.E. Sloper, and Y. Ryabov. A rule-based verification and control framework in atlas trigger-daq. *Nuclear Science, IEEE Transactions on*, 54(3):604–608, June 2007. ISSN 0018-9499. doi: 10.1109/TNS.2007.897825.

Alex Barriuso Poy, John Erik Sloper, Viatcheslav Khomutnikov, and Giovanna Lehmann Miotto. Operation of the atlas experiment: Organization of the detector controls and the data acquisition system. *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on*, pages 190–194, Nov. 2006. ISSN 1553-572X. doi: 10.1109/IECON.2006.347298.

J.E. Sloper and E. Hines. Detecting errors in the atlas tdaq system: A neural networks and support vector machines approach. In *Proc. IEEE International Con-*

ference on Computational Intelligence for Measurement Systems and Applications CIMSA '09, pages 252–257, 11–13 May 2009. doi: 10.1109/CIMSA.2009.5069960.

J.E. Sloper, J.E. Sloper, G.L. Miotto, and E. Hines. Dynamic error recovery in the atlas tdaq system. 55(1):405–410, 2008. ISSN 0018-9499. doi: 10.1109/TNS.2007.913472.

J. Vermeulen and et al. Atlas dataflow: the read-out subsystem, results from trigger and data-acquisition system testbed studies and from modeling. *Nuclear Science, IEEE Transactions on*, 53(3):912–917, June 2006. ISSN 0018-9499. doi: 10.1109/TNS.2006.873311.

Chapter 2

Overview of the ATLAS TDAQ system

This chapter provides a description of the key components in the TDAQ system with a particular focus on those that are related to error handling and detection and provides the reader with an introduction to the TDAQ system and how it is organised. Naturally, the organisation and composition of the TDAQ system dictates the choices that can be made when developing an error management (ER) system and it is therefore important that the reader has an overview of the system in order to understand the choices made in subsequent chapters.

2.1 The TDAQ system

As outlined in Section 1.1.2.2 the TDAQ system is a vast heterogenous system consisting of a large number of both software and hardware components. The system is connected to the ATLAS detector and its main purpose is to read the data from the detector, filter out the subset of data which may be of interest for further analysis and ultimately storing the data. This data is then later used for the so called 'off-line' analysis. The system gathers the data as it is produced by the detector and is therefore subject to strict efficiency requirements. The LHC produces collisions every 25ns (i.e. at a rate of 40MHz) the system is therefore massively parallel in order to be able to perform both the gathering and filtering of the data at the

required rate. At each stage of the filtering the data rate is reduced and more thorough filtering can be done. The filtering is divided into three stages; Level-1 filtering which reduces the rate to a few hundred kHz, Level-2 filtering where the rate is reduced to around 2kHz and finally, the event filtering reduces it to a few hundred hertz. At each stage of the filtering a more and more complete set of the detector data is considered, hence making the filtering more complex. An overview of the TDAQ system and the rate of data at the various stages is shown in Figure 2.1.

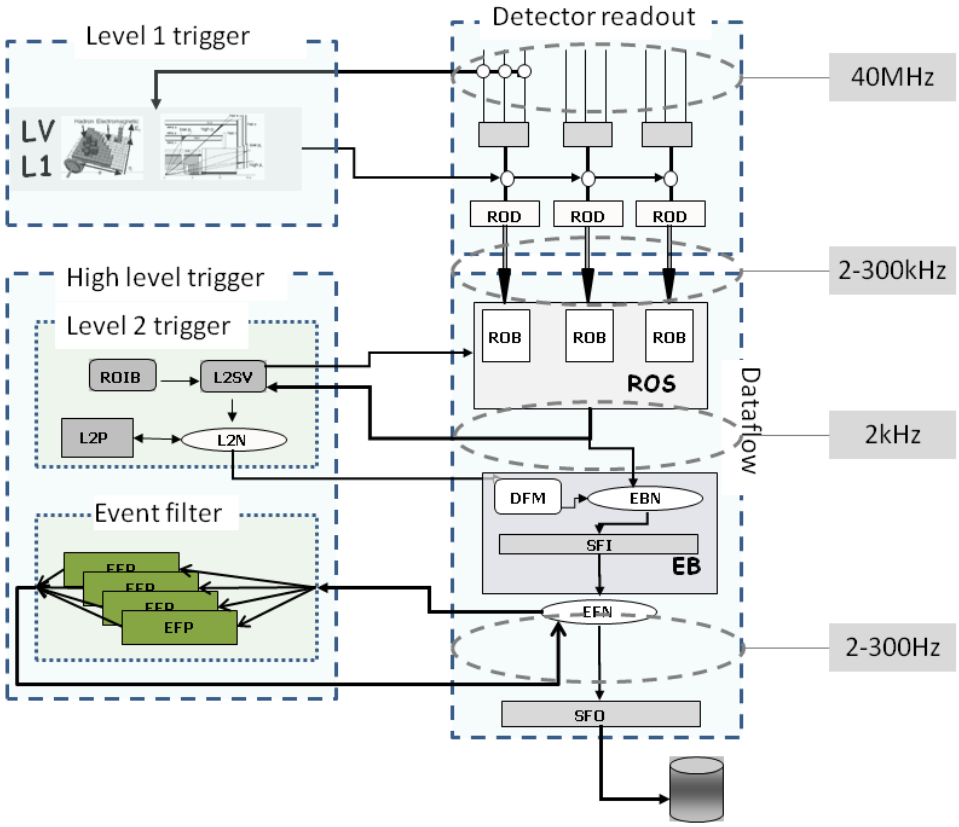


Figure 2.1: An overview of the TDAQ system and the data rates at each stage of the filtering.

The system comprises approximately 3000 computers of different types depending on the type of operation the processes running on it performs. The network connecting these computers is divided into two separate networks, one for distributing the data and a separate network for controlling the various processes. In total

there will be of the order of 50000+ processes running in the TDAQ system. At such a size errors must be expected and they do indeed frequently occur in the system today. It is therefore of great importance that the system is able to deal with and recover from these errors, should they occur.

In addition to the processes directly taking part in the flow of data there are also a number of services available providing functionality in order to monitor the system, store application errors and warnings, provide means of communication, enabling data quality monitoring, etc. There also exists a control framework providing control, command distribution to and synchronisation of all the processes in the system.

2.1.1 Key components

This section focuses on the subset of components that are of particular relevance in the context of the EM system.

2.1.1.1 Configuration Database

The configuration of the TDAQ system is based on an object oriented database containing a description of the TDAQ system. These descriptions cover the control-oriented configuration of all ATLAS applications which can be running during data taking. It includes all the information needed to configure the system, such as:

- which parts of the ATLAS systems and which detectors are participating in a given data taking session.
- where processes shall be started and when. It also contains information about which run-time environment is to be created for each of the processes.
- how to check the status of running processes and to recover run-time errors.
- when and in what order to shut down running processes.

In addition, the configuration database provides configuration parameters for many applications such as the overall DAQ data-flow configuration, online monitoring configuration and connectivity, configuration parameters for various DAQ

systems and for detector modules and channels. The configuration database is available to all applications in the system through a database service hierarchy and is accessible by all applications that request it.

The configuration database will contain one or more *partition* objects which also includes the complete description of the system from a control point of view. This description contains all information needed to configure the system for a data taking session. Such partitions may for example contain the configuration information needed to test and calibrate a specific part of the ATLAS detector, or it may contain information for a complete data taking session including all parts of the detector.

A partition object contains a set of *segment* objects, typically representing a subsystem or a collection of applications with similar functionality, e.g. a set of Readout modules. Each segment contains a set of applications and resources, and may also contain other (sub-)segments. Each of the segments is associated to a controller application (see Section 2.1.1.2) which is responsible for all applications contained in that segment. In this manner a hierarchical structure is used to build the system as shown in Figure 2.2.

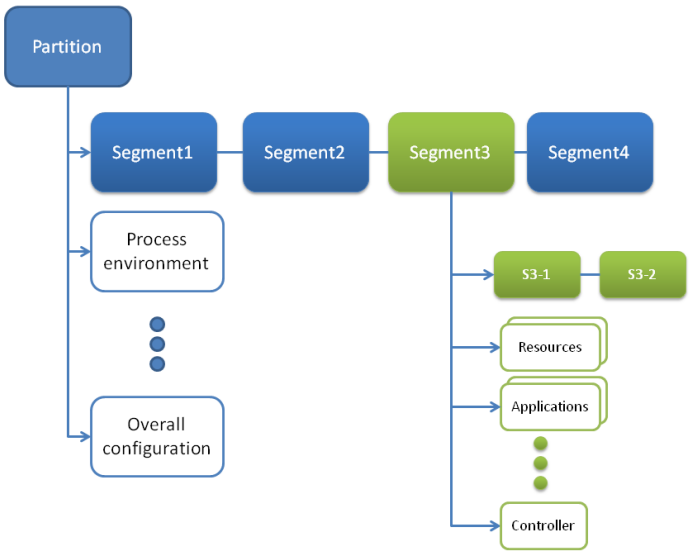


Figure 2.2: A schematic view of the control part of the configuration database.

The author has contributed to the work on the configuration database and a paper was published describing further details and challenges of the configuration database (Almeida et al., April 29 2007-May 4 2007).

2.1.1.2 TDAQ RunControl

The TDAQ RunControl system is responsible for distributing commands from the operator(s) throughout the system. It starts, stops and monitors all applications in the TDAQ system and ensures that the system is in a coherent state. The system comprises of a number of 'controller' applications organised in a tree structure.

In addition, there are a number of services based on the client-server model. The most important services will be discussed in the following sections. In order to synchronise operations throughout the system, Finite State Machine (FSM) principles are used. Figure 2.3 shows the FSM used for the TDAQ system. In addition to the states shown in the diagram it can also go into an error state indicating that the application cannot continue its function. The RunControl is constructed using the configuration database with controllers arranged in a tree structure in which each controller is responsible for a segment. Normally, commands are only sent to the topmost controller and are then propagated throughout the control tree. Interaction with the RunControl is performed through a graphical interface which among other things displays the RunControl tree, including the current state and any errors. Figure 2.4 shows the logical layout of the RunControl.

2.1.1.3 Inter Process Communication (IPC)

Communication between the processes in the TDAQ system is realised using the dedicated IPC package. The package is built on top of third party solutions, OmniOrb(Grisby) (C++) and JacOrb (Brose, 1997)(JAVA), which are implementations of the CORBA (Group) standard, a widely used standard for interprocess communication. Very briefly, the IPC allows applications to make themselves available as *named* services in the system and makes it possible for processes to communicate without taking into account 'low-level' issues such as machine names, protocols,

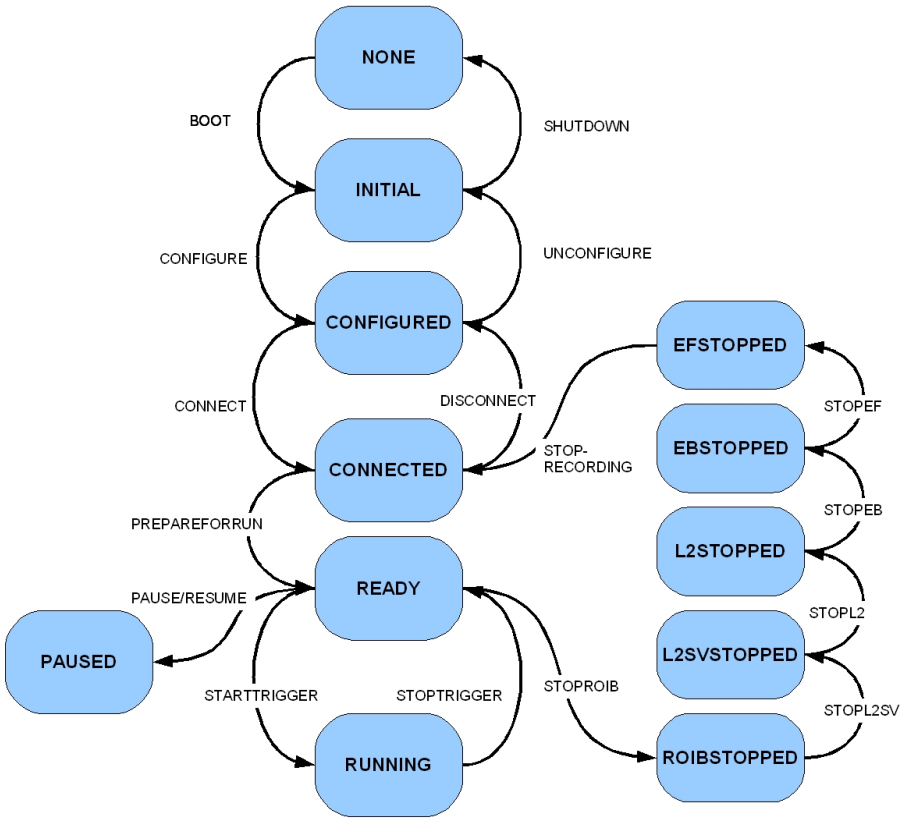


Figure 2.3: The finite state machine used by the RunControl system.

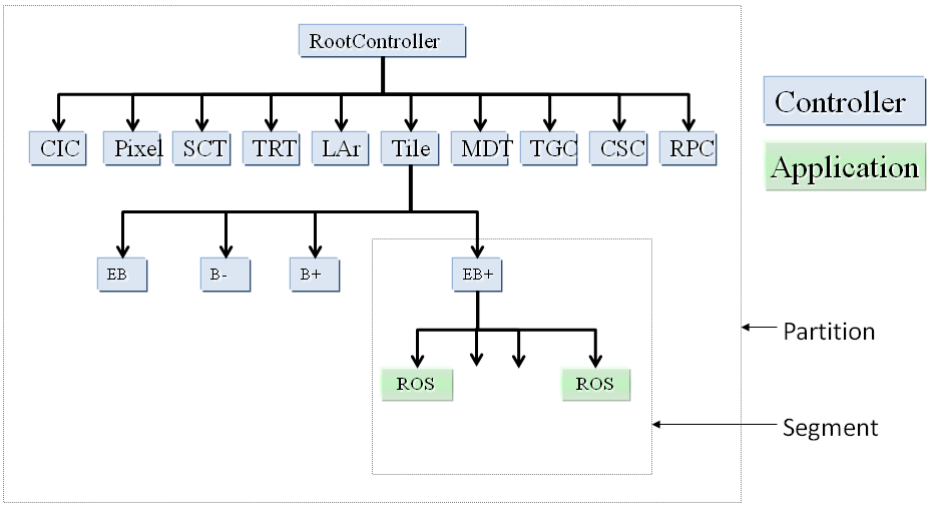


Figure 2.4: The logical layout of the RunControl.

port numbers, sockets, and so on. Hence, when communicating with another process one simply has to know the name that process is published with and one can then communicate with it using the IPC API.

2.1.1.4 Message Reporting System (MRS)

The MRS is a service for passing messages between different applications using a *subscription-notification* model. It provides a means of passing messages between the applications in the system and is designed to be scalable in order to sustain any needed message rate. Tests have been conducted showing that the system is capable of sustaining message passing between more than 50000 processes. The MRS is the general way of passing messages in the system outside the 'point-to-point' communication provided by the IPC. The difference between the IPC and the MRS communication model is illustrated in Figure 2.5.

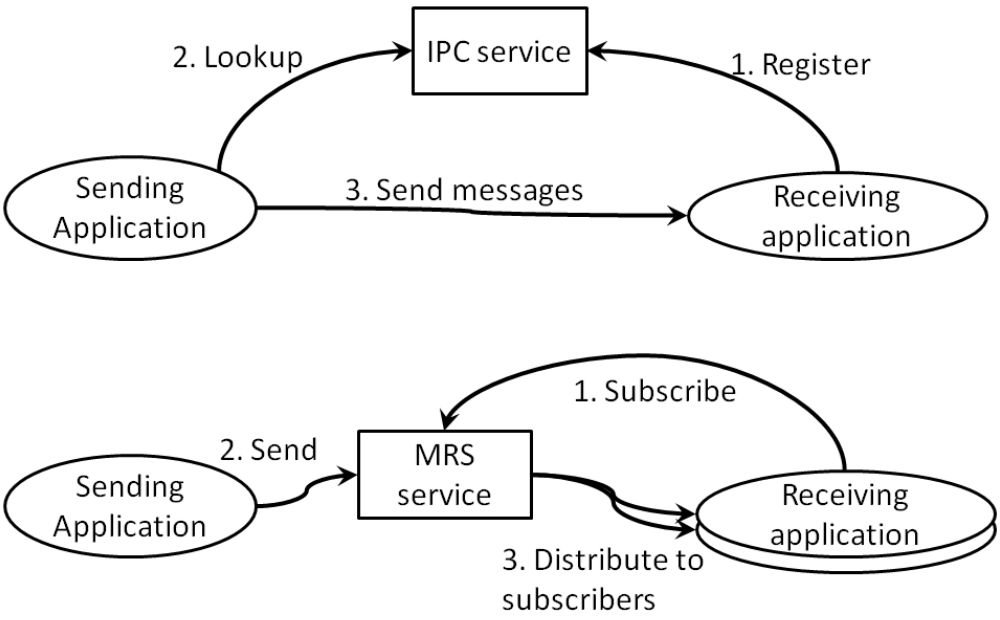


Figure 2.5: Illustrates the difference in communication model using IPC versus MRS.

2.1.1.5 Information Service (IS)

The IS is a general way of sharing information in the system without using any one-to-one communication. The IS allows applications, referred to as *providers* in the context of the IS, to publish information which will be stored on a server. Other applications, called *receivers*, can then actively retrieve the data they are interested in, or subscribe to changes of a particular information set. The information made available (i.e. published) by a provider can be updated or deleted and receivers can retrieve the latest copy or be notified that it no longer exists. The IS provides a number of basic types which dictate which types of information can be stored (integers, floats, strings, etc) and can also store an array of any of this. It also allows the users to create new data types which can be stored in the server. The IS also supports meta-information which can be used to describe the information that is published in the service.

Many of the applications publish statistics about their own performance using this service, and it is therefore of interest when doing error detection. For example, a reduction in the processing rate for a number of applications may be an indication that something has gone wrong in the system.

2.1.1.6 Error Reporting Service (ERS)

The ERS provides several services, including a common format and a fixed range of severity levels for all errors reported in the TDAQ system. It is possible to configure global settings that define the behavior of error reporting, such as where errors are sent, amount of information for each error, etc. This common framework also makes it possible to collect errors into classes/groups to be reused by other applications. For example, in the case of a 'file not found' or a 'connection not possible to establish', different applications developed by different developers can use the same 'error class' (or sub-classes thereof) to report the issue. In order to pass ERS messages between applications the ERS relies on the MRS package in order to pass error messages between different applications and also to allow for subscriptions to different messages.

2.1.1.7 LogService

The TDAQ system also includes a 'log service' which gathers all errors reported through ERS by all applications in the TDAQ system and stores them in a common database. This is relatively easy to do as all messages are transported over MRS and it is therefore a matter of subscribing to all messages. As all the messages are stored in a single database this makes it easy to browse them and retrieve parts of it based on any combination of parameters such as the level of severity, application type, application name, time, host and message contents. The operation of the LogService is illustrated in Figure 2.6.

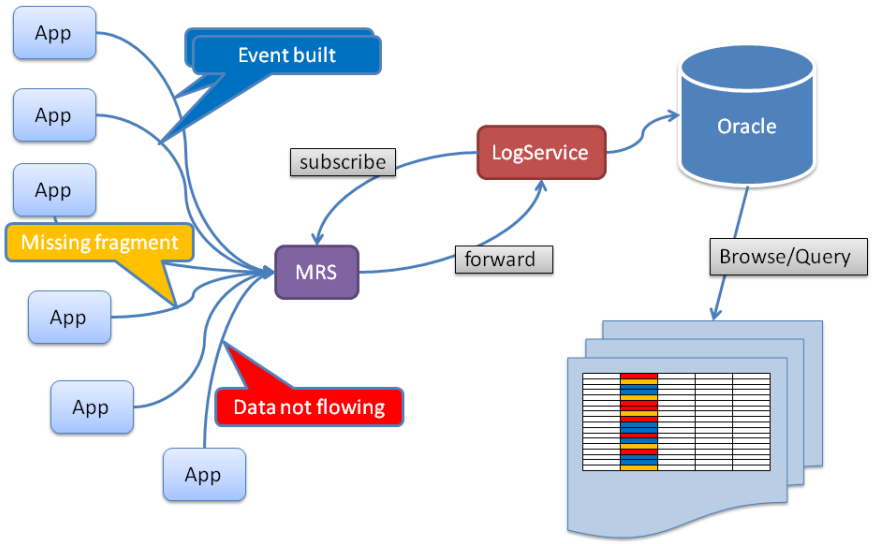


Figure 2.6: The LogService stores all messages in a database which can subsequently be queried.

A detailed description of the LogService is available in (Murillo-Garcia and Miotto, 2007).

2.1.1.8 Process ManaGer (PMG)

This package handles all communication with the operating system related to starting processes, monitoring them, signalling them and so on. It allows applications to start processes on any machine without dealing with low-level calls or operating system interfaces. The PMG also provide call-back mechanisms for notifying applications of any changes in other processes. For example, a controller is notified whenever a child application is terminated and also what was the cause of the termination. Applications are identified using a unique handle constructed based on the configuration information. This handle is therefore identical between different executions of the same application, as opposed to for example an operating system process id. Having a fixed handle for each application makes it possible for a controlling application to re-connect to any 'child' processes if it for some reason has lost contact with it. This extra level of indirection makes the system much more fault tolerant and makes a number of recovery actions possible which otherwise would not have been possible.

As an example consider the case where a controller application was restarted. It is now possible for the controller to find out which of its child processes are still running and regain complete control of them. It can then start applications that are missing or otherwise bring its part of the system back to the state in which it should be. Also, using the PMG service, it allows for other applications (e.g. some part of the EM system) to query the process manager about the status of any application in the system. This can be used in order to obtain further information about the actual status of an application. An example is the case where an application is reported as not responding to requests. It is then possible to query the PMG service in order to find out whether the application is indeed running and not responding or whether it is actually not running anymore.

2.2 Conclusions

This chapter has provided an overview of the ATLAS TDAQ system and its main component and provides the reader with an introduction to how the system is organised and how the main components work and interact, and thus provides some context for understanding choices made in subsequent chapters. In particular, this provides a context via which to evaluate many of the design and implementation choices made when developing the current error management system as described in Chapter 3.

The focus has been on the components that are directly relevant to a EM system while less relevant components in this context have been left out. It is meant purely as an overview of the system and for a more complete description of the system please refer to (Collaboration, 2003).

References

J. Almeida, M. Dobson, A. Kazarov, G. Lehmann Miotto, J.E. Sloper, I. Soloviev, and R. Torres. The atlas daq system online configurations database service challenge. *Real-Time Conference, 2007 15th IEEE-NPSS*, pages 1–8, April 29 2007-May 4 2007. doi: 10.1109/RTC.2007.4382770.

Gerald Brose. Jacorb: Implementation and design of a java orb. In *International Conference on Distributed Applications and Interoperable Systems*, pages 143–154. Chapman & Hall, 1997.

Atlas Collaboration. Atlas high level data acquisition and controls - technical design report. Technical report, CERN, Geneva,, 2003.

Duncan Grisby. The omniorb version 4.1 users guide. Web site: `http://omniorb.sourceforge.net/omni41/omniORB.pdf` [Last accesed: 20th August 2010].

Object Management Group. Common object request broker architecture (corba/iiop). Web site: `http://www.omg.org/spec/CORBA/3.1` [Last accessed: 10th August 2010].

Raul Murillo-Garcia and Giovanna Lehmann Miotto. A log service package for the atlas tdaq/dcs group. *IEEE Transaction on Nuclear Science*, 54:202–207, 2007.

Chapter 3

Error Management in ATLAS TDAQ: The Expert system approach

This chapter describes in detail the existing expert system (ES) approach used to build an efficient error management system (EMS) that has been implemented and is currently in use in the ATLAS TDAQ system. First, the concept of ESs in general is introduced before the requirements of the EMS for TDAQ are presented together with a description of how it has been designed and implemented. The EMS with the ES at its core forms a basis for any further development of, and extensions to, error detection and recovery in ATLAS TDAQ and it is therefore very important to have a complete understanding of it. Finally, the known limitations of the system are discussed together with how it performs for different types of error situations in order to find areas that are suitable for improvement using ISTs.

3.1 Introduction to expert systems

Expert systems (ESs) are a subfield of Artificial Intelligence (AI) which deals with complex problems within a well defined specialised field or domain. An ES is usually realised by encoding the knowledge of an expert in the field/domain in question (hence the name expert system) in such a way that this knowledge can be reproduced

by an automated system. Some of the typical applications of an ES are:

(i) to offer advice to a non-expert user. The system can act in the place of an expert and make the expert knowledge widely available to non-expert users.

(ii) automatically try to solve given problems using the encoded knowledge. For example, an ES can be used to control the traffic lights at a junction (or over a larger area) in order to optimise the flow of traffic. This is then done without any user interaction, but automatically by the system using the encoded knowledge.

(iii) be used as a tool for the expert themselves for either verifying their work or possibly increase their efficiency. For example, a car technician can use a diagnostic ES in order to verify his/her own diagnostic, or alternatively run the diagnostic program in the first place in order to save time.

ESs are used within a wide range of fields including fields such as flow and water quality modeling (Chau et al., 2002), finance (Nedovic and Devedzic, 2002), engineering (Starek et al., 2002) and others and is still an evolving field with numerous practical applications (Liao, 2005).

3.1.1 Rule based expert systems

The most widely used form of ESs are the so called rule based expert systems (RBES). In an RBES the encoded expert knowledge is usually referred to as the knowledge base (KB) and consists of *rules* in an IF-THEN form. Rules consist of two parts, namely:

Antecedents: One or more conditions that must be fulfilled for a proposition/rule to be true. Hence, in a rule in the IF-THEN form it is the part that follows the IF statement and precedes the THEN statement.

Consequents: is the second half of a proposition (i.e. the rule). In a rule in the IF-THEN form, the consequent is that part that follows the THEN statement.

An example of a rule with two antecedents and one consequents would therefore be:

"IF A is black AND A flies THEN A is a raven".

In this statement the antecedents are *"A is black"* and *"A flies"*, while the consequent is *"A is a raven"*. While the logic is unsound it demonstrates the difference. In addition, it contains a short-term memory containing factual information (e.g. current pressure of a valve is 20 bar) usually referred to as *facts*. These facts usually represent the antecedents of a rule. The RBES then relies on an *inference* engine in order to drive the system and automatically activate/fire rules that are relevant to the current situation. That is, it will match the rules from the knowledge-base to the facts in its current working memory. Inference engines usually follow one of two approaches:

Forward chaining: This is an approach where the working memory is matched with the available rules so that if all the antecedents of a rule are fulfilled the consequent part is added to the working memory. An example of a system using forward chaining is CLIPS (see Section 3.2.3.1 for further detail).

Backward chaining: As opposed to forward chaining this approach starts with a consequent and attempts to find antecedents that fulfills that rule. An example of a system that uses backward chaining is the logic programming language of Prolog(Shapiro and Fuchi, 1988).

Forward chaining is typically referred to as a data-driven method as it continuously updates its working memory as new rules are fired. It is often employed in expert systems where the working memory is often changing. Backward chaining in contrast is a goal-driven approach.

An explanation facility may also be of importance (and sometimes crucial) depending on the application of the ES. For example, an ES providing a medical diagnosis must be able to provide an explanation for its results, while an ES controlling the temperature of a bakery oven does not necessarily have such requirements.

A system using backward chaining makes it easier to create an explanation facility as the full chain of reasoning exists for each goal. Using forward chaining on the other hand one can not necessarily deduce how a particular goal was achieved.

RBESs also typically include a user interface in order to present the results to the user. Such an interface can also be used to input new information into the system dynamically. Alternatively, or in addition, the system can gather information itself as a result of its execution. For example, a rule could be executed which actively gathers more information (e.g. runs an external test program). The information gathered in that way may then trigger new rules and so on. The different components of a typical RBES and their interactions are shown in Figure 3.1.

The data structures typically available in a RBES are shown in Figure 3.2 and can be summarised as follows:

Rule: expert knowledge written in an IF-THEN form containing *antecedents* and *consequents*.

Fact: A single factual statement representing some information, for example that a valve is open or closed, or that the current speed limit is 60mph.

Object: representation of some entity usually containing one or more facts describing that entity. For example, an object representing a car may contain facts such as the cars make, its speed, its weight, the current amount of fuel left, etc. The concept of objects should be well known to anyone with experience from object oriented programming languages (Stroustrup, 1991).

Facts and objects are typically a part of the working memory and change as further input is given to the ES (or gathered by it), or as rules are fired that change the state of the objects or the facts.

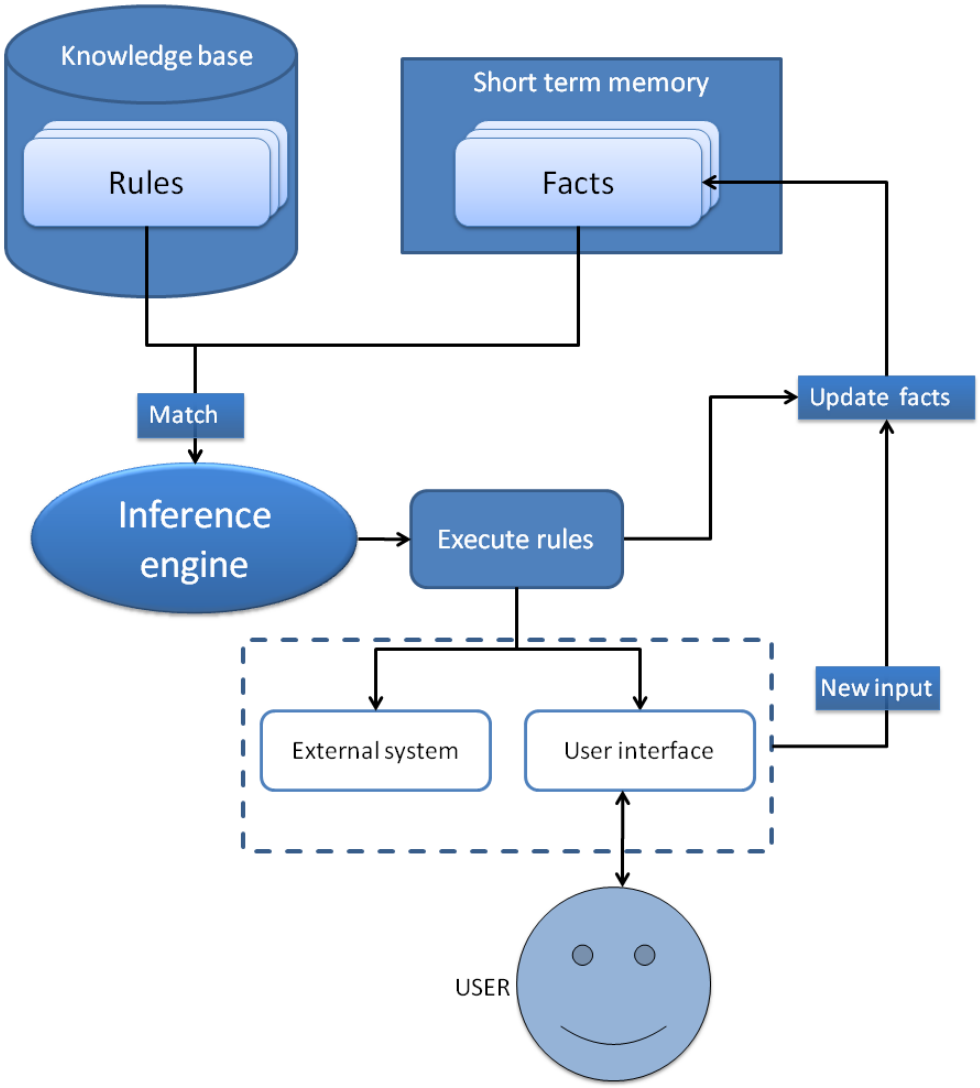


Figure 3.1: Overview of a typical RBES.

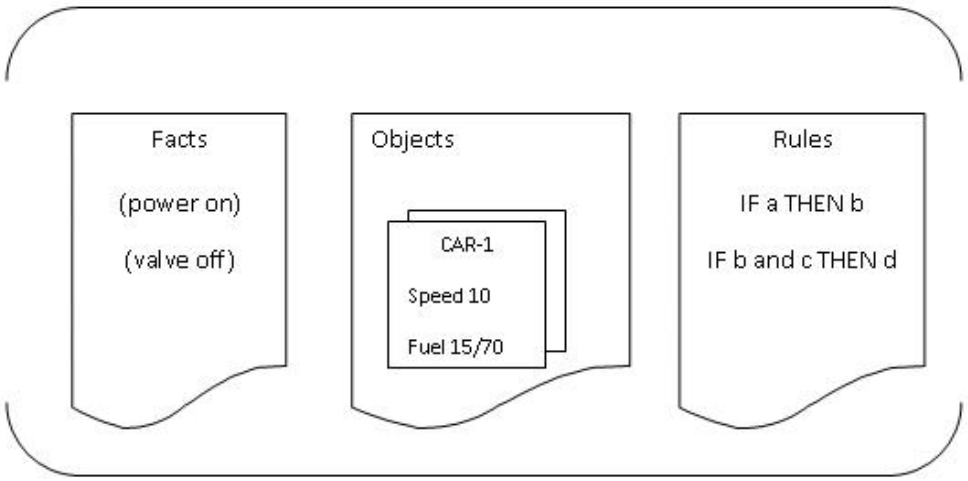


Figure 3.2: The different types of data structures used in a typical ES

3.2 Error Management in ATLAS TDAQ using an ES

Given the size and complexity (see Section 2.1) of the ATLAS TDAQ system, errors are bound to happen and must be dealt with. Due to the modular and distributed nature of the system it is normal practise to create a dedicated EMS to deal with this. An alternative would be to create separate error management systems for the different components and sub-systems. However, this would mean that error handling efforts would be duplicated in each sub-system. It would also complicate efforts to deal with cross-system errors as each sub-system may have been based on different technologies and designs making communication between the different EMSs difficult. Such an approach would also make understanding and maintainability of the system as a whole more difficult.

An ES approach to creating a dedicated EMS was first attempted more than 6 years ago (Liko et al, 2004) and is still being developed to deal with the changing requirements of the TDAQ system. The initial ES encompassed a much broader requirement set including the top-level control (see RunControl in Section 2.1.1.2) of the entire system. Error management was therefore only a sub-set of the functionality of the ES. The size and complexity of the ES became very hard to maintain and

the functionality of the RunControl was separated out and the ES was redesigned to deal exclusively with error management related functionality. The new requirements and design choices of the new ES are described in the following sections.

3.2.1 Requirements

There are a number of main requirements that have underpinned the general design of the EMS. The following is a brief overview of the most important ones.

3.2.1.1 Error handling

The EMS should be able to:

(i) react to errors reported in the system. This entails error detection and is the basis for all further action.

(ii) analyse the errors. The EMS must be able to recognise different types of errors and find (as far as possible) the root cause of them.

(iii) take appropriate actions to ensure that the system returns to or stays in an operational state. This entails automatic error *recovery* operations which could include a number of operations such as restarting applications, reverting previous actions, switching to fail-over mechanisms or simply ignoring the error. If automatic recovery is not possible the system must support interaction with a human operator.

The errors and situations detected and dealt with by the EMS should be well defined and known to the operators of the system. This is important as one does not want the EMS to take recovery actions at the same time as another component or a human user intervenes. Such 'double' actions could thus lead to confusion and potentially cause problems for the system, thus leading to situations where valuable operational time could be lost.

3.2.1.2 Customizability

It must be possible to ensure that different behavior will take place in response to similar, or indeed identical, problems arising in different types of sub-systems. For example, if an application which is a part of one of the filtering sub-systems (see Section 2.1) stops working this might have no overall effect on the system other than a slight reduction in computing power as there are potentially thousands of other applications performing the same task. However, if an application in the ReadOut System (ROS) dies, this will have considerable impact on several other sub-systems, and must therefore be distinguishable and be dealt with differently.

3.2.1.3 Configurability

It must be possible to easily change and configure the behavior of the recovery system. There are three main reasons for this:

(i) As error management was largely overlooked in early stages of the system development all the requirements are not yet identified. Sub-systems are still requesting new functionality which must be taken into account.

(ii) The TDAQ system is still under development and it must therefore be able to accommodate for both new components being added and changes being made to existing ones. Such additions and changes to the system might lead to new error scenarios that need to be accommodated for by the ER system.

(ii) In the final system there will be a relatively large number of different configurations of the TDAQ system. This depends on which detectors are currently taking part in the data collection process, machine availability, current goals for the experiment, etc. It must therefore be possible to configure the ER system to accommodate the different sets of needs in terms of functionality.

All these factors mean that the EMS must be as flexible as possible and it must be possible to add, change and remove functionality within a relatively short timescale (weeks or even days).

3.2.1.4 Abstractability

The EMS should, as far as possible, be built in a modular way such that any parts can be replaced without the need to change any other components. This would make it possible to change one part of the system, for example a module detecting a certain error, without the need to change the recovery part of the system. It should also offer an interface to the reporting applications without exposing any details or mechanisms of the EMS. For example, applications should be able to report errors through a fixed interface without knowing anything about details such as where the EMS is located, or whether it exists at all! This will also help to minimise the need for change by developers of other components in the system. This is very important because the EMS is being introduced relatively late in the development of the TDAQ system and there might be limited possibility for changing application code. Such a process can be tedious and error prone and it is difficult to ensure that all developers have implemented the correct changes.

3.2.1.5 Maintainability

As the TDAQ system is a very large and complex system with a large time span (decades), code maintenance is a serious issue that must be taken into account. The EMS, or at least the core part of it, must be implemented in such a way and using techniques and technologies which facilitates future maintenance of the system. This means that while new techniques may be introduced, the core of the system must be implemented using relatively well known technologies which future developers of the system are likely to, or can be expected to, know. This includes using commonly known programming languages.

3.2.1.6 Performance

The EMS should be able to analyse the errors and reach a decision within a reasonable time span, usually in the order of seconds or even faster, depending on the type of error. The longer these decisions take, the longer the TDAQ system will remain in an error state causing a loss of data taking capability. Also errors in one

part of the system can lead to errors in other parts and so on creating a 'cascade' of errors in the system. Fast recovery from the early errors will help to prevent such situations from arising. This is of importance as such situations can be very difficult to deal with for the EMS and can greatly increase the time the system is not operational.

3.2.2 Design

In the following section we will consider in more detail how the EMS was designed taking into account the requirements presented in the previous section.

The design of the EMS is divided into *two* main parts; a local unit and a global unit. Each of these units have a distinct set of responsibilities and the collection of all units compose the overall EMS. Considering the RunControl as shown in Figure 2.4 in Section 2.1.1.2 the local unit exists at the level of each controller, while there exists a single global unit independent of the control tree. An overview of this is shown in Figure 3.3.

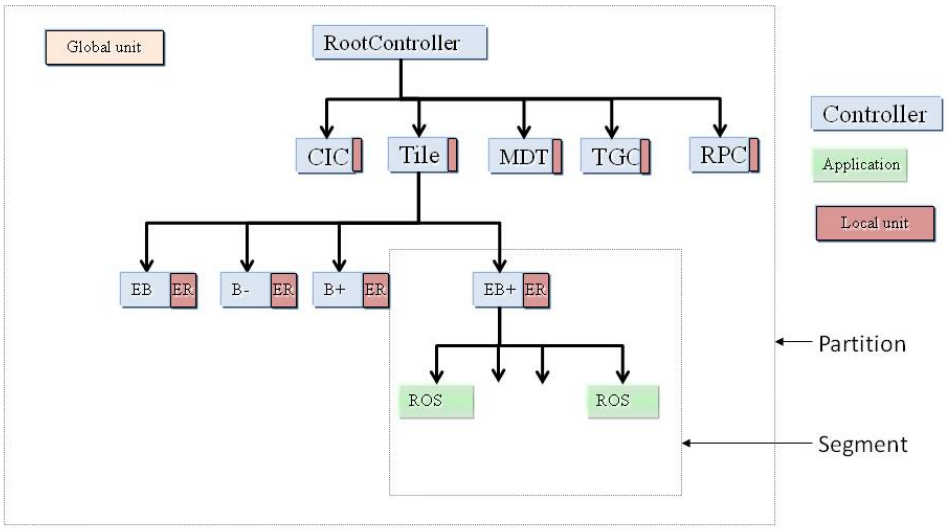


Figure 3.3: The local and global unit in relation to the RunControl tree.

Both the global and local units have access to the configuration database

and uses it to build maps of the connections in the system, infer which applications are dependent on each other, read default recovery actions and determine which tests are associated with the different components. In the following section there is a more detailed consideration of the specifics of the local and the global units in turn:

Local unit: The main goal of the local recovery unit is to handle errors that can be dealt with at a segment level, that is, errors that do not have an immediate impact on the rest of the system. It is integrated with each controller in the control tree and has access to information about all the applications within the associated controller's segment. Any changes in the applications status are immediately reported to the EMS by the controller. In this way the EMS can react as soon as possible to any problems within that particular segment. It will then analyse any problems taking into account information such as the current configuration, the overall state of the system, the state of the controller and any other errors already present.

It can set the error state of the controller if needed, but is also able to perform more advanced actions such as restarting applications or notifying other applications if necessary. In addition to receiving errors directly from the controller it can also receive errors directly from applications through the ERS/MRS and it subscribes to relevant information in the IS which may be needed for the recovery procedure. All errors gathered by the local unit will be reported to the global unit, including information such as whether the action has been taken and if the problem has been solved by the local unit or if further action is needed.

In general the local unit should be able to deal with the most common errors in the TDAQ system. Each local unit may also be specialised to perform specific actions depending on which sub-system it is connected with (see Figure 3.3). Hence, this helps fulfill the requirement of customizability (see 3.2.1.2), at least for errors which can be dealt with at the level of that sub-system.

As an example consider an application, e.g. an L2PU application, within a given segment which silently stops working. This is detected by its supervising

application, and L2SV application, which then reports the problem using the ERS. The local unit is notified through its ERS subscription. Depending on its configuration it can then take a decision whether the problem can be handled locally, i.e. without affecting other parts of the system. If it can be dealt with locally the appropriate actions are taken by the local unit, e.g. restarting the application and notifying its supervisor if and when it has successfully reached its running state. If it cannot be handled locally the global unit is notified.

Global unit: The global unit handles errors that have a system-wide impact. For such errors there will be applications in different segments that are affected by the recovery actions and they can therefore not be dealt with by the local unit. As the global unit keeps track of all errors in the system, including the ones reported by the local units, this allows for more general decisions to be made based on factors such as frequency of errors within a segment and/or the system as a whole.

Let us consider an example of recovering from faulty Readout drivers (RODs). The RODs are both part of the trigger hierarchy, but are also connected to Readout subsystems (ROSs) which are located in a different part of the control tree. The local unit does not have any knowledge of other parts of the system and can therefore not deal with this problem. Instead there are some applications monitoring the RODs which, whenever a 'faulty' ROD is detected, send a predefined ERS issue. The global unit receives the issue through the ERS framework (using the ERS extension). The issue is then parsed and the global unit can take action. Depending on the overall state of the TDAQ system and its current configuration the global unit will then take appropriate actions informing all affected applications and storing the information in a conditions database so that any later analysis of the ATLAS data have full knowledge of the system at any given point in time.

3.2.2.1 Inter-operation with external systems

In addition to handling errors from the TDAQ system, the EMS must also be able to handle errors from systems outside of TDAQ such as the Detector Control System (DCS), networking, farm monitoring tools and so on. This is realised by integrating a

proxy application representing the external system into the control tree and passing error messages through the proxy. A detailed description of this interaction in the case of the DCS can be found in (Poy et al., Nov. 2006). The information gained in this way allows the ES both to deal with a wider range of errors, and also to correlate or add information to existing problems.

3.2.2.2 Active testing

The EMS is designed to interface with relevant components available in the TDAQ system such as the Diagnostics and Verification System (DVS) which is described in detail in (Kazarov et al., June 2007). The DVS allows the ER to actively test components in the system, which is especially useful in cases where the actual fault is not immediately apparent or when further evidence should be gathered. For example, if an application is reported to be 'not responding' the recovery system can test the network connections or the host of that application to see if the error is in fact a hardware problem. The DVS has a number of well defined tests for different hardware objects and applications which can be used by the recovery system to properly identify different types of problems.

Asynchronous recovery: Due to the distributed nature of the system it is, in most cases, not practical to perform a synchronous recovery of errors. Even though the system does support synchronous communication between applications (through the IPC package), the recovery system is designed to perform the recovery in an asynchronous manner. There are several reasons for this:

(i) First of all to ensure the abstraction requirement is not violated (see Section 3.2.1.4) , it is better not to have a direct connection between applications which are reporting errors and the EMS. The system is designed so that applications can report errors without needing any extra logic to deal with or wait for response from the EMS, or indeed have any knowledge about it at all.

(ii) Also due to time constraints it might not be practical for some applications

to wait for an answer from the ER system before continuing its operation. For example, an application might report that another application is not responding to its requests, but it may still be able to continue its own operation. Hence, the problem is a minor one for the reporting application and using an asynchronous recovery model the application does not need to wait for any response to its report, but may simply continue working.

3.2.3 Implementation

This section describes how the EMS was implemented based on the design presented in the previous section. As the core parts of the local and global units are the same, they will be described first. The specifics of each unit will then be described in more detail.

3.2.3.1 Expert system implementation

An expert system is used at the core of the EMS and is implemented using the CLIPS framework which is a free open source ES framework developed by NASA (Riley et al., 1991). Some of the main features of the CLIPS framework are:

- An inference engine supporting *forward* chaining.
- Supports both procedural and object oriented programming in addition to the declarative rule programming
- Represents expert knowledge in an IF-THEN form which is human readable
- It is easy to extend using the C++ programming language

CLIPS uses the Rete (Forgy, 1982) algorithm for driving the inference engine. The Rete algorithm is best used in situations with many rules/many objects and is therefore well suited for representing the TDAQ system. The forward chaining approach also suits the EMS well as one is not usually trying to diagnose, but rather detect errors. Also, the system is changing very often meaning that new information must constantly be updated in the ES memory. Hence, the data-driven approach

is easier to use for this particular system. CLIPS has been used successfully for a number of different projects ranging from robot control (Spelt et al., 1989) to on-line diagnostic systems (Lauriente et al., 1992). CLIPS has also been used for a number of years at CERN and in the TDAQ group in particular and was therefore a natural choice as an implementation framework.

3.2.3.2 Organisation of the knowledge base

At the core of the ES is naturally the knowledge base containing the necessary rules to effect the EMS. As CLIPS supports an object oriented paradigm, this has been extensively used in the implementation. Information about the different applications, computers and other hardware is represented in the ES using 'proxy' objects. These objects correspond to the actual applications and hardware in the system and are updated as soon as any changes affecting their state happens in the system. Figure 3.4 shows a UML class diagram of all classes defined within the ES. In addition to the classes, a number of facts describing either global variables or other simple information are used. This can include information such as global time-outs, or information that is not persistent or is only used for a set period of time. For example, certain facts are used to keep track of the recovery procedure of an application and are removed after the procedure is completed. Hence, the knowledge-base consists of both the rules and the facts and class definitions.

Whenever the ES is started it is populated with relevant information such as class instances representing all the applications in the controllers segment (in the case of the local unit). This information can then trigger rules in the expert system. The matching or facts and objects to the rule base is performed by the inference engine.

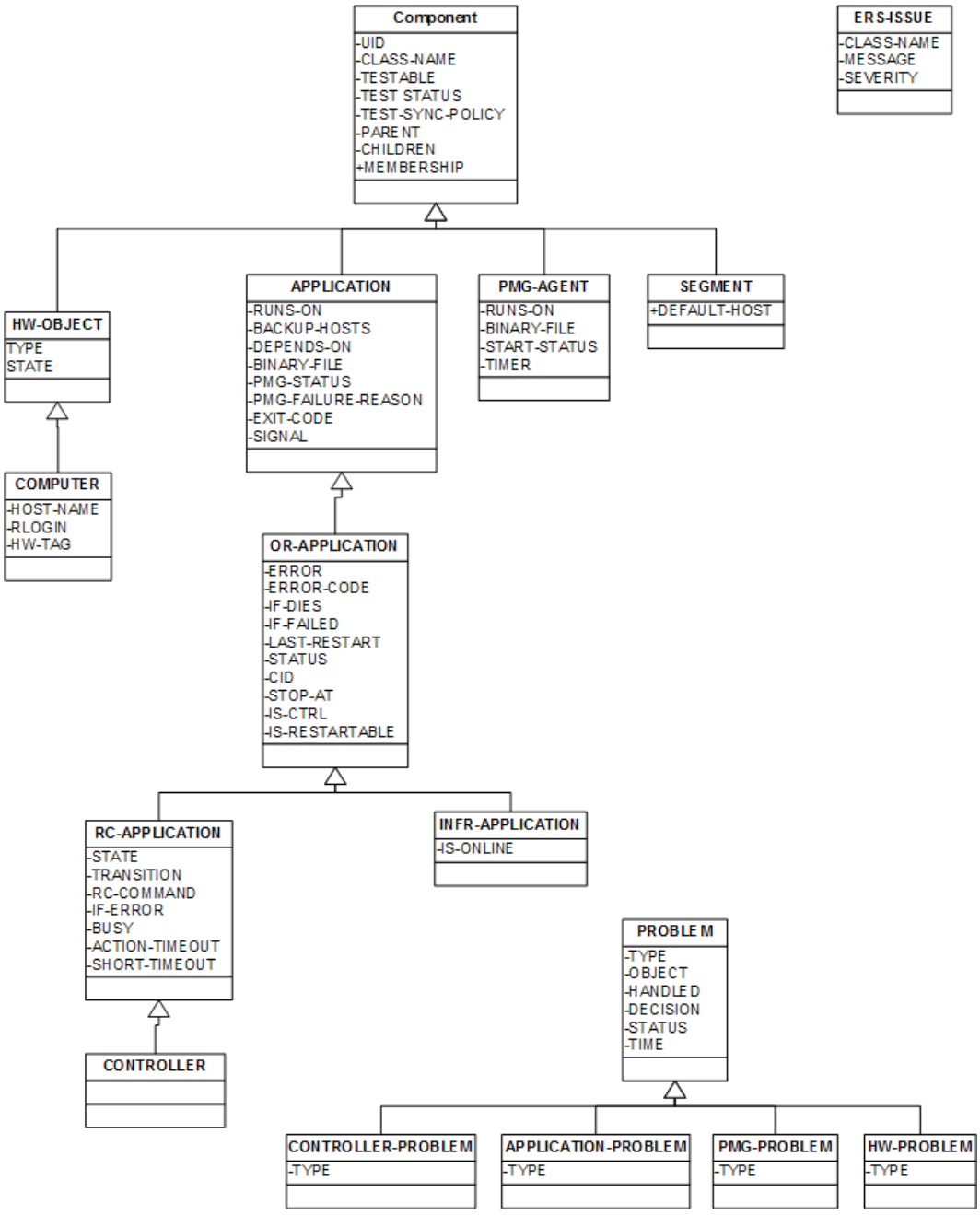


Figure 3.4: A UML diagram of the classes used in the CLIPS implementation.

3.2.3.3 Extensions

The ES can be extended using C++ modules, thereby adding functionality to the ES framework. This allows for an easy and flexible way of extending the functionality of the CLIPS framework and to provide integration with the rest of the TDAQ system. Figure 3.5 shows a schematic overview of the key components of the implementation. As an example of an extension let us consider the ERS extension. This extension makes it possible to use ERS functionality from within the CLIPS environment. This includes both subscriptions to errors coming from the ERS and the reporting of errors through it, both of which are used extensively. For example, subscriptions to particular error types are created using functionality provided by the extension. Then whenever such an error is received a corresponding object (an instance of the ERS-issue class) is created in the ES memory. This might then trigger rules within the system which may cause actions to be taken by the EMS.

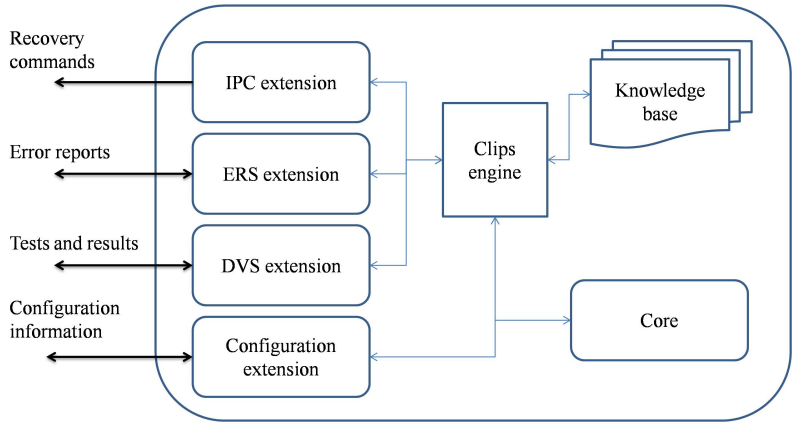


Figure 3.5: The key components common to both the local and global unit.

Another example of an extension is the IPC extension which enables communication with the rest of the system through the IPC services (see Section 2.1.1.3). This allows the EMS to send information or commands directly to any application running in the system based on a handle constructed from configuration information.

The local and global unit contain much of the same core components and mainly differs in the knowledge bases (expert rules and facts). The global unit also includes some additional extensions in order to fulfil its extra responsibilities, such as dealing with system wide errors. In the following sections the details of both the local and the global unit are described.

Local unit: In accordance with the design there should be a local unit connected to each controller. This is realised by the means of a *plug-in* library loaded by the controller (see Figure 3.6). The controller passes on configuration parameters to the plug-in when it is loaded. This facilitates further configurability of the EMS and this configuration information could for example be extra rules to be loaded by this plug-in in particular. This is the main method with which custom rules are loaded for specific sub-systems, i.e. the controller associated with a particular sub-system tells the plug-in to load the rules specific to that sub-system.

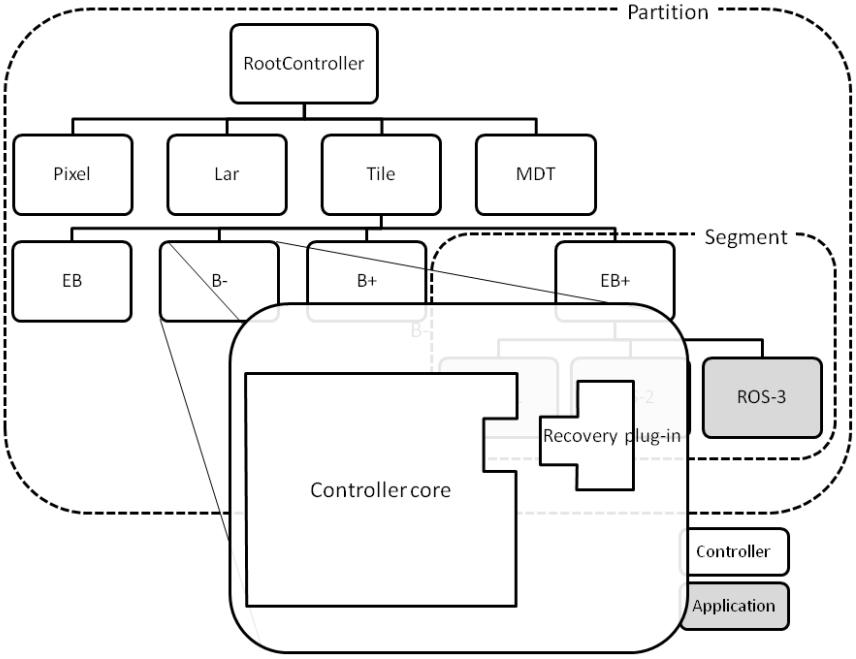


Figure 3.6: The local unit is implemented as a plug-in to the controller

Global unit: The global unit is implemented as a service using the IPC package to expose its interface to the rest of the system. It has both an interface for receiving messages from the local units, and also the possibility of interacting directly with a user/expert. This direct interaction is used in order to configure the behavior of the global unit while it is running and also to retrieve a variety of statistics from it such as number of errors per sub-system, the cause and resolution of each error etc.

The global unit mainly deals with errors whose impact extends beyond the scope of a single segment and thus cannot easily be dealt with by the local unit. It receives information directly from the local unit and also through the ERS. This is achieved by subscribing to certain predefined issues which are sent by sub-systems whenever certain conditions are detected or specific types of errors occur. The global unit is then able to communicate with the sub-systems and issues commands to the sub-system controllers or directly to the affected applications. This way the global unit is able to notify applications across the sub-system boundaries and thus effectively deal with errors outside the local units' scope. Figure 3.7 shows a schematic view of the recovery server's interaction with the rest of the system.

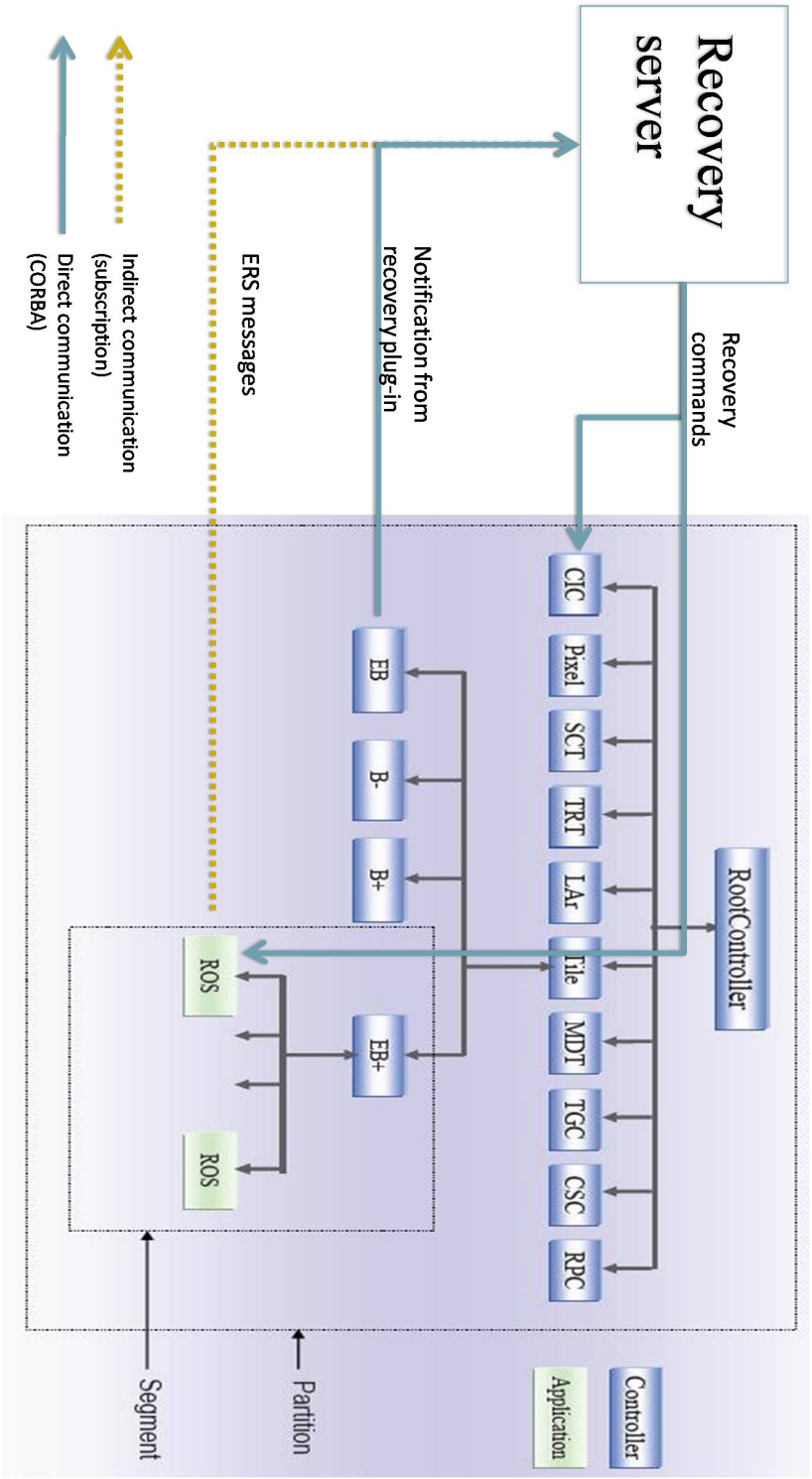


Figure 3.7: Communication with the recovery server is achieved either directly using IPC/CORBA or by sending ERS messages to which the server is subscribed.

3.2.4 Known limitations and possible extensions

As the ES is likely to stay at the core of the ER system for the foreseeable future (especially due to maintainability reasons) it is natural to look at ways of extending it, rather than replacing it. This has been taken into account when exploring new techniques for error detection and handling.

The current implementation does allow extensions to the CLIPS system using C++ extensions and this provides the best route for adding new techniques which can then interact with or be interrogated from the ES. By connecting new techniques with the existing ES this will yield a high degree of flexibility where for example some new techniques are only activated whenever certain conditions are present in the system. It does however mean that one should mainly be looking at dealing with problems which are currently not handled by the EMS, or situations where the current performance is limited or inaccurate.

The ES has some notable limitations that must be taken into account when developing techniques to improve it. These include:

Error detection: The system is built upon a discreet framework (either something is in error or not). It relies heavily on notifications from external components, such as the controllers, the process management system and specialised applications in order to detect errors in the system. While this is sufficient for the basic operation of the TDAQ system it does leave room for significant improvement, particularly in the area of actively detecting and dealing with error situations. This area will be investigated in this thesis. Specifically, the data is analysed and visualised in Chapter 5 and various ISTs are investigated in Chapters 6, 7 and 8 in order to detect errors.

Generalisation: The current system is designed to deal with a number of specific error scenarios. This means that the generalisation ability of the system is very limited. While it can react to and deal with a number of errors very well, it is not able to detect or handle slightly different or similar errors. Currently, such errors must be dealt with by a human operator. This is a major weakness

in the current system and is an important factor when evaluating new ISTs in Chapters 6, 7 and 8.

Adaptation: Due to the fact that the system relies on a rule based ES approach this means that all knowledge must be coded into the ES knowledge base. In order to do this experts of a particular problem must be consulted and information gathered from them. There is no possibility to train or adapt the system to new problems or error scenarios without adding or modifying the existing rules. This means that the system needs to be updated by an expert developer and subsequently re-tested which is both time consuming and error prone. Overcoming this problem is another aspect that must be taken into account when evaluating the new techniques that are developed.

3.3 Discussion and conclusions

The ES approach has been used in the TDAQ system for more than 6 years and provides a solid base for an error detection and handling system. The implementation has been based on CLIPS which was chosen mainly due to the knowledge and experience of the developers in the TDAQ group with this particular technology. The TDAQ project has a large timescale (at least until 2020), and as no re-design or implementation is currently planned this means that the current solution will be continuously updated for the foreseeable future of the project. Maintainability is therefore an very important factor and it is unlikely that any large changes will be made in the underlying implementation.

Also for reasons of maintainability and consistency it is very important that new functionality can be added while maintaining the current functionality and without substantial changes to the existing implementation. The current design and implementation using the CLIPS framework is well suited for realising this.

The EMS is an ongoing work and will continue to be used in the ATLAS TDAQ system for the foreseeable future. However, adding to it and/or incorporating new techniques, such as those investigated in subsequent chapters, is of great

interest and provides a promising way forward in order to increase the usefulness and effectiveness of the system. There are a number of areas which can be improved and where functionality can be extended as discussed in Section 3.2.4. This thesis provides the first step in this direction and provides an insight into a number of approaches that can deal with the limitations of the current system. As new methods and functionality are developed they must be integrated with the ER system and tested on a large scale. As experience with this process is gained there may be new opportunities for improvement and extension of the system which may not be apparent at this time.

References

Kwok-wing Chau, Chuntian Cheng, and C. W. Li. Knowledge management system on flow and water quality modeling. *Expert systems with applications*, 22(4): 321–330, May 2002.

Charles Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982. ISSN 0004-3702.

A. Kazarov, A. Corso-Radu, G.L. Miotto, J.E. Sloper, and Y. Ryabov. A rule-based verification and control framework in atlas trigger-daq. *Nuclear Science, IEEE Transactions on*, 54(3):604–608, June 2007. ISSN 0018-9499. doi: 10.1109/TNS.2007.897825.

M Lauriente, M Rolincik, Koons HC, and D Gorney. An on-line expert system for diagnosing environmentally induced spacecraft anomalies using clips. In *The Sixth Annual Workshop on Space Operations Applications and Research (SOAR*, pages 1992–329, 1992.

Shu-Hsien Liao. Expert system methodologies and applications–a decade review from 1995 to 2004. *Expert Systems with Applications*, 28(1):93 – 103, 2005. ISSN 0957-4174. doi: DOI:10.1016/j.eswa.2004.08.003.

D. Liko et al. Control in the ATLAS TDAQ system. In *Computing in High Energy Physics and Nuclear Physics*, page 159, Interlaken, Switzerland, sept-oct 2004.

Ljubica Nedovic and Vladan Devedzic. Expert systems in finance–a cross-section of the field. *Expert Systems with Applications*, 23(1):49 – 66, 2002. ISSN 0957-4174. doi: DOI:10.1016/S0957-4174(02)00027-1.

Alex Barriuso Poy, John Erik Sloper, Viatcheslav Khomutnikov, and Giovanna Lehmann Miotto. Operation of the atlas experiment: Organization of the detector controls and the data acquisition system. *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on*, pages 190–194, Nov. 2006. ISSN 1553-572X. doi: 10.1109/IECON.2006.347298.

G. Riley, C. Culbert, and F. Lopez. C language integrated production system (clips). *NASA Tech Briefs*, 1991.

E. Shapiro and K. Fuchi, editors. *Concurrent Prolog.* MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-19255-1.

P.F. Spelt, G. De Saussure, E. Lyness, F.G. Pin, and C.R. Weisbin. Learning by an autonomous robot at a process control panel. *IEEE Expert*, 4(4):8–16, Winter 1989. ISSN 0885-9000. doi: 10.1109/64.43281.

Michael Starek, Mukesh Tomer, Krishna Bhaskar, and Mario Garcia. An expert system for mineral identification. *J. Comput. Small Coll.*, 17(5):193–197, 2002. ISSN 1937-4771.

Bjarne Stroustrup. What is "object-oriented programming"? In *In European Conf. on Object Oriented Programming*, pages 27–6. Springer-Verlag, 1991.

Chapter 4

Error detection in ATLAS TDAQ - A theoretical overview and analysis of governing factors

4.1 Introduction

This chapter presents background information about the theoretical consideration of error detection in the context of machine learning and looks at different ways of evaluating the effectiveness of an error detection system. It also discusses some different error detection models before considering the specifics of the ATLAS TDAQ system and which factors are most relevant when implementing error detection in that system.

4.2 Error detection and measures of effectiveness

This section provides a brief overview of error detection from a machine learning (ML) perspective and discusses the general approaches to developing and adapting different error detection models. It also defines different ways with which to evaluate the effectiveness of various error detection models.

4.2.1 Error detection from a machine learning perspective

The process of detecting errors can, in a simple form, be compared with pattern classification where the goal is to classify a pattern as being of a given class or not. An error detection system is therefore a 'machine' which is able to classify input data into errors and non-errors. More formally we can define it as follows. Given a vector x of k observations where each observation has an associated correct classification of y_i being either -1 or 1.

$$x_i \in R^n, i = 1, \ldots, k \tag{4.1}$$

If there exists an unknown probability density $P(x, y)$ from where our x observations are drawn from; assuming there exists a machine whose task it is to learn the mapping $x_i \rightarrow y_i$. Then if the machine is defined by a set of possible mappings $x \rightarrow f(x, \alpha)$ where α are the configurable parameters of the machine then a choice of α will be what one refers to as a 'trained machine'. One can then define the expected error of the trained machine on any test set as

$$R(x) = \int \frac{1}{2}|y - f(x, \alpha)|dP(x, y) \tag{4.2}$$

However, one will often not know the distribution $P(x, y)$ and hence we will normally operate with an *empirical* risk:

$$R_{empirical}(x) = \frac{1}{2k}\sum_{i=1}^{k}|y - f(x, \alpha)| \tag{4.3}$$

As can be seen from (4.3) this is merely the mean of the sum of the machine output minus the expected output.

4.2.2 Training

This section provides a general overview of different approaches to training and common problems that can occur independently of the underlying model that is being trained. It also discusses how to best test and choose between the different models that are being developed.

4.2.2.1 Supervised learning

Supervised learning is a type of ML where known target values are used to control the learning process. Typically each set of input data is associated with a set of output data, so that for a given number of inputs $x_i,\ i = 1, \ldots, n$ there exists a corresponding known output value $y_i,\ i = 1, \ldots, n$. This output is then used in order to drive the learning process. A typical approach is to use the output value in order to calculate the empirical risk and then use this calculated risk to drive the learning process. This approach is used to train the different ANNs in Chapter 6, to find the optimal SVMs in Chapter 7 and in the fitness function used in the CGP approach in Chapter 8.

4.2.2.2 Unsupervised learning

Unsupervised learning, in contrast to supervised learning, has no associated target for the input values. Unsupervised learning is normally used to perform some kind of compression (e.g. dimensionality reduction) and/or clustering of the data. An example where unsupervised learning is used is Self Organised Maps (SOMs)(Kohonen, 1982); a type of ANNs often used for classification and data visualisation (see Section 5.2.3).

4.2.2.3 Semi-supervised learning

A third method exist where a mixture of supervised and un-supervised learning is used. This method is not used in this thesis, however an overview of the approach can be found in (Zhu, 2005).

4.2.2.4 Over-fitting

One problem that can often occur when training different types of learning machines is that the developed model fits too closely to the training data. This can lead to poor generalisation performance as the model is biased towards the training samples. An example of over-fitting is shown in Figure 4.1 where a model has been trained using the available samples. The samples have been created using the function

$y = x^2 + x$ and some white noise has been added. One can see that the trained model interpolates all the sample points, but does not represent the underlying model very well. For an in-depth look at over-fitting the reader is referred to (Hawkins, 2004).

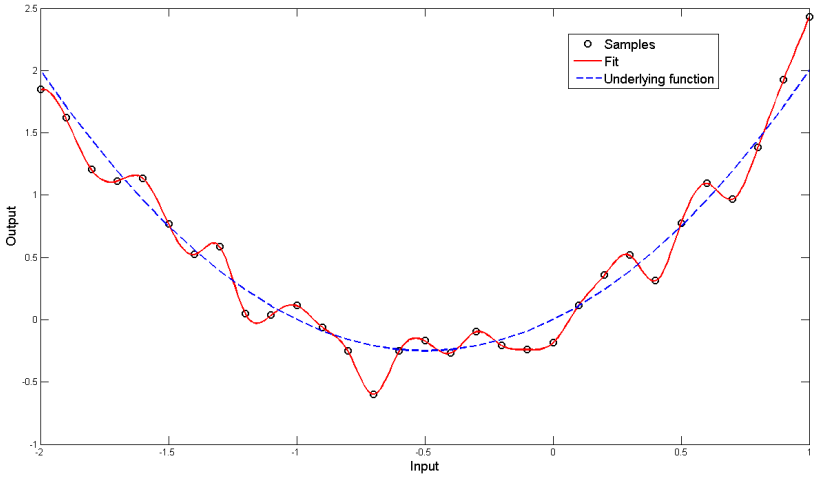


Figure 4.1: A model suffering from over-fitting.

4.2.2.5 Model selection

The ultimate goal of our models is to predict errors based on future unseen input data. In other terms we wish to maximise the *generalisation* performance of our dataset. In order to estimate the generalisation performance of the developed models one must test the model using some input data. Training and testing with the same dataset will yield a poor estimation of the true generalisation performance and will likely yield a heavily biased results being overly optimistic of the performance(Bishop, 2007). Instead, it is common to separate the dataset into a *training* set and a *test* set. The model can then be trained using the training set. The generalisation performance can then be estimated using the test set.

Often a model may contain different parameters that can be adjusted and one would like to choose the optimal one for a given application. Furthermore, one might wish to compare different models in order to evaluate which is better for the particular application. One approach would be to train the range of models using

the training set and then choose the model which achieves best results using the test set. However, this may lead to some over-fitting to the test set, especially if the model design is repeated a number of times. In such cases it may be necessary to introduce a third set often referred to as the *validation* set. The models are then chosen based on their performance on the validation set instead. The generalisation performance of the final model is then evaluated using the test set.

Note that in the literature the meaning of the terms validation set and test set are often exchanged. In this thesis the validation set is used for intermediate tuning of the model performance while the test set always refers to the set which is used to estimate the final generalisation performance.

4.2.3 Classification measures

This section presents different measures which can be used to evaluate and compare different classification models.

4.2.3.1 Classification accuracy

Classification accuracy is the basic measure of effectiveness and is simply the percentage of correctly classified samples:

$$ClassficationAccuracy = \frac{Correct}{(Correct + False)} \tag{4.4}$$

Alternatively one could use classification error:

$$Classficationerror = \frac{False}{(Correct + False)} \tag{4.5}$$

However, merely using the classification accuracy or error is not sufficient in most cases.

4.2.3.2 Alternative measures

Given that one operates with binary outputs (either something is class A or not), one can introduce some common measures of effectiveness for different classification methods. Each classification result can be labeled in one of four ways:

True positive (TP): a positive classification output that is correct.

True negative (TN): a negative classification output that is correct.

False negative (FN): a negative classification output that is incorrect.

False positive (FP): a positive classification output that is incorrect.

Based on these labels we can now define some more advanced measures of effectiveness:

Sensitivity: Sensitivity gives a measure of how well positive cases are correctly classified. Sensitivity is defined as:

$$Sensitivity = \frac{TP}{(TP + FN)} \tag{4.6}$$

A sensitivity of 100% indicates that the all 'positive' cases are correctly classified.

Specificity: Specificity gives a measure of how well our test correctly classifies negative cases. Specificity is defined as:

$$Specificity = \frac{TN}{(TN + FP)} \tag{4.7}$$

A specificity of 100% indicates that all 'negative' cases are correctly classified.

However, sensitivity and specificity are in themselves not enough to provide an adequate measure of the performance of a classifier. Let us consider a hypothetical situation where we have 100000 samples where 99000 belongs to class A (negative) and 1000 belongs to class B (positive). We have trained a classifier which correctly classifies 990 samples as positives, but misclassifies 900 true negative samples as positive as shown in Table 4.1.

Table 4.1: Example result of a classifier

	Positive	Negative
Positive result	990	900
Negative result	10	98100

From Table 4.1 we have that sensitivity = 990/(990+10) = 99% and a specificity of 98100/(98100+900) = 99.1%. Looking only at specificity and sensitivity one could assume that this is an excellent classifier. However, looking closer it is apparent that just over 50% of the positive classifications are correct. In order to better understand such cases we introduce the positive predictive value and the negative predictive value as further measures of a classifier's effectiveness.

Positive predictive value (PPV)

$$PPV = \frac{TP}{(TP + FP)} \tag{4.8}$$

A high positive predictive value indicates that a positive result from the network is likely to be correct.

Negative predictive value (NPV)

$$NPV = \frac{TN}{(TN + FN)} \tag{4.9}$$

A high negative predictive value indicates that a negative result from the network is likely to be correct.

For the example discussed earlier we can see that PPV is just 52.38% while the NPV is 99.98%. So we see that a positive classification result is not much more than 50% likely to be correct. Clearly, one should strive to maximise the PPV and the NPV as well as sensitivity and specificity.

In general, when performing error detection in the TDAQ system it is more important to have a high NPV than a high PPV. This is due to the fact that a positive result is usually followed by further tests which can eliminate the false positive results.

4.3 Error detection in distributed systems

A lot of research has been done related to dealing with errors in distributed systems and the following is a brief overview of some different error models that are

commonly used in computer systems.

4.3.1 The Byzantine model

The Byzantine error model originates from the Byzantine generals problem (Lamport et al., 1982). The general problem describes a situation where n generals must agree on a specific action. Messages can only be passed between the generals by a messenger. However, some of the generals may be traitors attempting to sabotage the other generals. In a computer system this would mean that a component in the system can exhibit arbitrary and/or malicious behavior, i.e. a component may actively attempt to sabotage the overall system. However, the Byzantine model is quite general and many systems may not encounter such situations in reality. We will therefore look closer at two more error models in the following sections.

4.3.2 Fail-stop model

Another popular model is the fail-stop model (Schneider, 1990). In this model applications are modeled using a state machine, and will upon failure enter an error state that permits other components to detect this error. The ATLAS TDAQ system is to a large extent modeled using this approach, where the local units are responsible for detecting the errors of the applications in its segment (see Section 3.2.2 for more details). However, the approach has some general limitations. For example, if a failure is manifested, not as a wrong result or a halt of operation, but as a degradation of performance the application will not necessarily move to an error state and hence the error cannot be detected. The system will therefore not be able to deal with such problems or will only deal with them when they become very severe (e.g. when the performance degrades completely). Such situations are indeed very possible in the TDAQ system where the throughput of different components may be affected by a number of factors. Depending on the state of the component and the system, such degradations should be considered an error and it is therefore clear that the Fail-stop model is not enough to deal with systems such as the ATLAS TDAQ.

4.3.3 Fail-stutter model

In order to deal with performance degradation situations Arpaci-Dusseau (Arpaci-Dusseau and Arpaci-Dusseau, 2001) introduced the 'fail-stutter' model which takes into account these kinds of factors. He argues that performance faults should be separated from 'correctness' faults. A correctness error is here referred to the situation where a component is no longer operating according to its specification. A performance error on the other hand occurs when a component has not absolutely failed, but is no longer performing at a level according to its performance specification. This naturally entails specifying the acceptable performance levels for each component something that is typically an ongoing process and the specifications may change as the system evolves. This is also true for the TDAQ case, which has the added complexity that what is considered acceptable performance changes with a number of configuration parameters. Still it is important that the error model used in the system has the possibility of dealing with such errors.

4.4 Error detection in the ATLAS TDAQ system

Performing error detection in a system such as the ATLAS TDAQ is a potentially complex and difficult tasks. As the TDAQ system consists of a variety of distinct parts and sub-systems, each behaving in specialized ways, it is not likely that a single general way of performing error detection exists. One must first define the scope of the error detection, both in terms of parts of the system to consider and which kind of errors one should attempt to detect.

As mentioned in Section 4.3.2 the use of the fail-stop model is at the basis of the current error detection and handling in the TDAQ system. However, this is no longer adequate and in order to move to a fail-stutter model new error detection techniques must be developed in order to detect performance degradation of components.

4.4.1 Types of errors

There are many different types of errors that can occur in a large distributed system such as the ATLAS TDAQ. Following is a classification of the most common types of errors:

Crashing software: This is by far the most common problem in the system. Crashing is for these purposes defined as an unexpected stop of the process in question. Software may stop working for a number of reasons including memory violations (usually leading to a segmentation fault), uncaught exceptions, etc. These situations are almost without exception caught by the PMG and its controller is notified. It therefore fits well in the fail-stop model and can usually be dealt with quite easily.

Non-responding processes: Some processes stop responding to requests arriving over IPC. Experience with the system shows that there is no easy way of determining whether the process actually stopped working or if there is a problem with the communication. For example, the process could be still performing its operation, but the part handling communication might have problems. Also there could be a problem with the network, either permanent, or simply a transient problem. Hence, the non-response does not necessarily mean that the process is not working any more, but may be due to another problem which is not immediately obvious.

Hardware failures: Hardware failures are not unlikely to happen in the system, especially considering the amount of custom hardware used in a physics experiment such as the TDAQ. Such errors may be detected immediately by the software, but could also be much more subtle and manifest themselves only as slight corruption of data without the controlling software ever detecting it. This could ultimately lead to problems further 'downstream' or when analysing the data at a later time.

Network problems: A number of network problems may arise. This could lead to situations where one or more applications are not reachable. Also, as the

TDAQ system has a separate data-flow and control network, a process may be reachable on one network and not the other and vice versa, making the situation more complex.

Operating system problem: A number of operating system related problems have been observed. Limitations such as number of file descriptors, TCP limitations, applications being terminated due to excessive memory consumption, and so on. These problems are sometimes easy to detect, but other times they may lead to 'obscure' situations where the problem appears to be caused by other factors or seems to reside in a different place than what is actually the case.

4.4.2 Available data sources

The current way of detecting errors in the system is based on information received from the PMG and/or by specific error messages sent by applications. The current system is well equipped to deal with such errors, and does so with very good results. However, the system is not capable of detecting the more 'subtle' types of errors such as applications not meeting their performance requirements or otherwise stopping working without any immediate indication that something has gone wrong. In order to deal with such errors we need to look at new sources of information for performing the error detection. There are two particular sources of information available in the system; namely error messages and IS data published by or about the different applications or sub-systems. They are currently vastly unused for the purpose of errors detection and it is therefore of interest to 'tap into' this reservoir of potentially useful information. Let us consider each data source in turn:

Error messages : There are two main services in the TDAQ system facilitating the gathering of error messages from the applications; ERS and the LogService. These services are described in Section 2.1.1. All messages reported through ERS use a common format which greatly simplifies the analysis and processing of these messages. The LogService subscribes to all ERS messages produced

in the system and stores them in an Oracle database. The LogService stores the ERS messages in a specific format keeping the information for each of the attributes shown in Table 4.2. The fact that the messages are kept in a database means that they can be accessed independently of the TDAQ system which allows for 'post-mortem' analysis of errors.

IS data : Data is published in a central information service. This can be, for example, data about the performance of the different components such as CPU rates, buffer levels, throughput, the amount of data processed by the different sub-systems, etc. However, one should note that there are no limits to the type of information that can be published in the IS. The data published in the IS system are available while the system is running and can relatively easily be gathered and subsequently analysed.

Table 4.2: The available attributes of an ERS message.

Attribute	**Description**
Message id	This is a unique name assigned to a certain error type.
Machine name	The name of the machine the application sending the error is located on.
Application name	Name/identifier of the application sending the error (as set in the configuration database).
Severity	Severity of the error message ranging from 'for information' to fatal.
Message	A text description of the problem. Usually for human readability, but they are in some cases parsed by the system automatically.
Time	The date and time the error message was created.
Parameters	A number of other parameters such as file name and line number where the issue was raised, the user running the process, etc.

Some examples of ERS messages gathered during tests are listed in Table 4.3. Note that the ERS parameters field has been left out for brevity.

Table 4.3: Example ERS messages.

Message Id	Machine name	Application name	Severity	Message	Time
L2Process::checkTimeout	lnxatd61.cern.ch	L2PU-8190	Warning	L2PU-8190: Incomplete RoI received for LVL1ID=63818: 0 of 1 ROS replies received	1211210124
ers::Message	pcatd133	monitoring-conductor	Warning	Event Sampler ReadoutApplication:ROS-1 not responding. Removing its subscription.	1211210144
onasic_oks2cool2::Info	lnxatd43.cern.ch	onasic_oks2cool	Information	onasic_oks2cool2::started onasic archiver to cool...	1211210013

4.4.3 Uncertainty

As in the case of most datasets there are a number of factors that can lead to uncertainty. Let us consider ERS and IS data in turn:

4.4.3.1 ERS data

- First of all, it is not obvious how to choose which features of the data is of real importance. In our example there are a number of features for each error message, such as the machine and the application it originated from, as well as time, severity, etc. To be able to analyse the data as efficiently as possible some of these fields might be discarded, modified or converted when the data is pre-processed. For example, fields containing text, such as the application name, might have to be converted into numerical values depending on the method of analysis used.

- Another problem is identifying which sub-set(s) of the data is of relevance to a particular analysis. In such a large system there will be a lot of 'noise' which may not be relevant to the particular problem one is considering. As an example, for a system containing 50000 applications there will be a significant number of messages (potentially thousands) from applications which are not related to a particular problem being investigated. Indeed, even for the smaller systems currently being used the number of messages gathered over a few days

is in the order of millions.

- Due to the distributed nature of the system another form of uncertainty arises. Applications will send error messages based on individual conditions such as lost connections, missing events, etc. There might be a number of factors determining the time when these conditions arises, such as current bandwidth, CPU scheduling, etc. Identical error conditions can therefore produce different sets of error messages even if all configuration options available to be set by the user are identical.

4.4.3.2 IS data

- As for the ERS data it is not trivial to determine which part of the data is relevant. This is even more difficult for the IS data as the type of information available varies from application to application. Each type of application in the system may publish a completely different set of information about itself or information related to its operation. This makes it more difficult both to choose the information that may give indications of errors in the system and also the processing of the data must take into account the different structure for each application type.

- Determining the correct sub-set of information is also a problem when considering the IS data. The final system will comprise more than 50000 applications. The information published by or about some of these applications will not be relevant in terms of performing error detection. However since the system is highly interconnected it might not be obvious how to best filter the IS data. For example, the information published by a seemingly unrelated application may indeed contain information that indicates the error (through interactions that are not necessarily obvious). Similarly, an application directly related to an error might not reflect this in the information it publishes in the IS. Thus, identifying the 'correct' subset leads to some uncertainty about the final dataset.

4.4.4 Incompleteness

There are two main aspects of incompleteness to be considered. Let us again consider the ERS and IS data in turn:

4.4.4.1 ERS data

- Since we are analysing messages sent by applications the available data clearly depends on the developers of each application. We are assuming that there exists a pattern of messages indicating a certain error (for example, warning of full buffers, delayed responses, etc). However, there are no guarantees that an application raises the correct cause of the malfunction (or any at all) for a specific problem. It is up to the developers of each application to put in place appropriate mechanisms for handling errors in a consistent way and raise the correct 'alarm' in order to represent the error in the most appropriate way. As there are tens of developers working on the ATLAS TDAQ system this suggest that mistakes on the developers parts are likely to occur. Clearly, the data may therefore be incomplete and this must be taken into account.

- The influence the different components have on each other is not necessarily fully understood. This is made even more difficult due to the distributed nature of the system, and also the possible effects of having a large number of developers and a relatively frequent release cycle (about two releases each year). In addition, since all the components have a highly specialised behavior it is very difficult for any single person to have complete knowledge of all the parts of the system and how they interact. Thus, the implications of some errors on the rest of the system are near-impossible to deduce without actually running the system and gathering empirical data which can then be analysed.

4.4.4.2 IS data

- The available IS data depends almost solely on the developers of the individual applications. Relevant information in terms of detecting errors may or may

not be published in the IS. The relatively large number of developers in the system makes it relatively likely that some useful information is not published.

4.4.5 General factors

In addition to these main concerns there are some other general factors which impact both datasets that should be considered. These are:

4.4.5.1 Continuous Development

Although the operation of the ATLAS experiment has started there is still ongoing development of, and changes made to, the software of the TDAQ system. This directly affects any effort to do error detection and recovery in several ways:

Information changes: The available data gathered may change or be removed in subsequent versions of the system. Even for the exact same situations and overall configuration of the system, the information produced may differ. Also, new information might be added which was not previously available and which may be useful for error detection.

Application behavior: The behavior of the different components and applications might also change as the development of the system proceeds. This is especially important when considering error handling methods, but also for error detection as the different applications might react differently to error situations and thus influence the detection procedure.

Recovery procedure: The 'correct' recovery actions might be different as the applications or components change or are added/removed from the system. It is therefore important that the error detection and handling techniques developed are flexible, adaptable or otherwise easily reproduced so that they can be applied to future versions of the TDAQ system.

4.4.5.2 Maintainability and available expertise

Due to the relatively long lifespan of the TDAQ system, in the order of tens of years, it is very important that any error detection and handling techniques or technologies are available and maintainable throughout its lifetime. This is especially the case in terms of the 'core' parts of the system which should be as 'simple' as possible and use techniques and technologies that are commonly used. This will make it more likely that future users and more importantly experts working with the system will have some knowledge of and experience with the techniques used to implement the system.

4.5 Conclusions

This chapter has provided a brief overview of machine learning theory and has presented ways of evaluating the effectiveness of error detection systems. The available data in the TDAQ system has then been discussed and possible factors that must be taken into account have been described. These considerations can then be taken into account when analysing the data in Chapter 5.

References

R.H. Arpaci-Dusseau and A.C. Arpaci-Dusseau. Fail-stutter fault tolerance. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 33–38, May 2001.

Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, New York, 2007.

Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, January 2004. ISSN 0095-2338.

T Kohonen. Self-organised formation of topologically correct feature map. *Biological Cybernetics*, 43:56–69, 1982.

Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. ISSN 0164-0925. doi: http://doi.acm.org/10.1145/357172.357176.

Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/98163.98167.

Xiaojin Zhu. Semi-supervised learning literature survey. Web Site: `http://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf` [Last accessed: 11th September 2010], 2005.

Chapter 5

Data gathering, preprocessing and preliminary analysis and visualisation

This chapter first describes the experimental setups used to gather data from the TDAQ system and includes a description of the topology of the test system and an overview of the different components and applications that are involved. A number of preprocessing and preliminary analysis and visualisation techniques are then described in detail before these techniques are applied to the datasets gathered. The results are then discussed together with the possible usefulness of the datasets in the context of developing error detection techniques using the datasets as inputs.

5.1 Experimental setup

Due to availability problems it was not possible to perform the tests on the real ATLAS system at the site of the experiment. However, the TDAQ software is designed to be able to run independently of the real detector and can therefore be run on appropriate test systems. Two different setups have been used in order to gather data from the TDAQ system; a single-host setup where all applications are running on a single host and a multi-host setup running on a dedicated computer cluster utilising 36 computers and including 57 different applications. The single-host setup

was used for initial tests and to verify the proof-of-principle. Subsequently, the multi-host setup was used to verify the results in a setup more similar to the real TDAQ system. Let us consider the two different setups in turn:

5.1.1 Single-host-setup

For the first setup data are gathered from a simple configuration of the TDAQ system. This configuration includes all parts of the data gathering chain; starting with detector readout through filtering and finally storing data on disks. The total number of applications/processes in this particular configuration is approximately 30, though 7 of them are not directly involved in the 'data chain'. A simple description of each component can be found in Table 5.1 and a schematic view of the experimental system is shown in Figure 5.1. All the processes are, as the name implies, running on a single computer.

This is naturally not comparable to the actual system which runs on 3000+ computers, but it is representative of the key components and will give some initial indications of how the system behaves in different scenarios. There are several reasons for starting with a simple configuration:

(i) First of all, it is easier to control the behavior and environment surrounding the setup. This is important as one must be able to induce different types of errors in the system at specific times. This can then form a basis for developing error detection techniques and/or to measure the effectiveness of clustering and analysis methods.

(ii) Secondly, it allows for an easier assessment of the overall status of the system, ranging from 100% functional through partially functional to 100% non-functional.

(iii) Thirdly, it provides a base reference for further tests. By comparing the results gained using this setup with those using a larger system one can compare how the system reacts to the same scenarios. This may then give a better

understanding of how different errors affect the system taking into account the scale and topology.

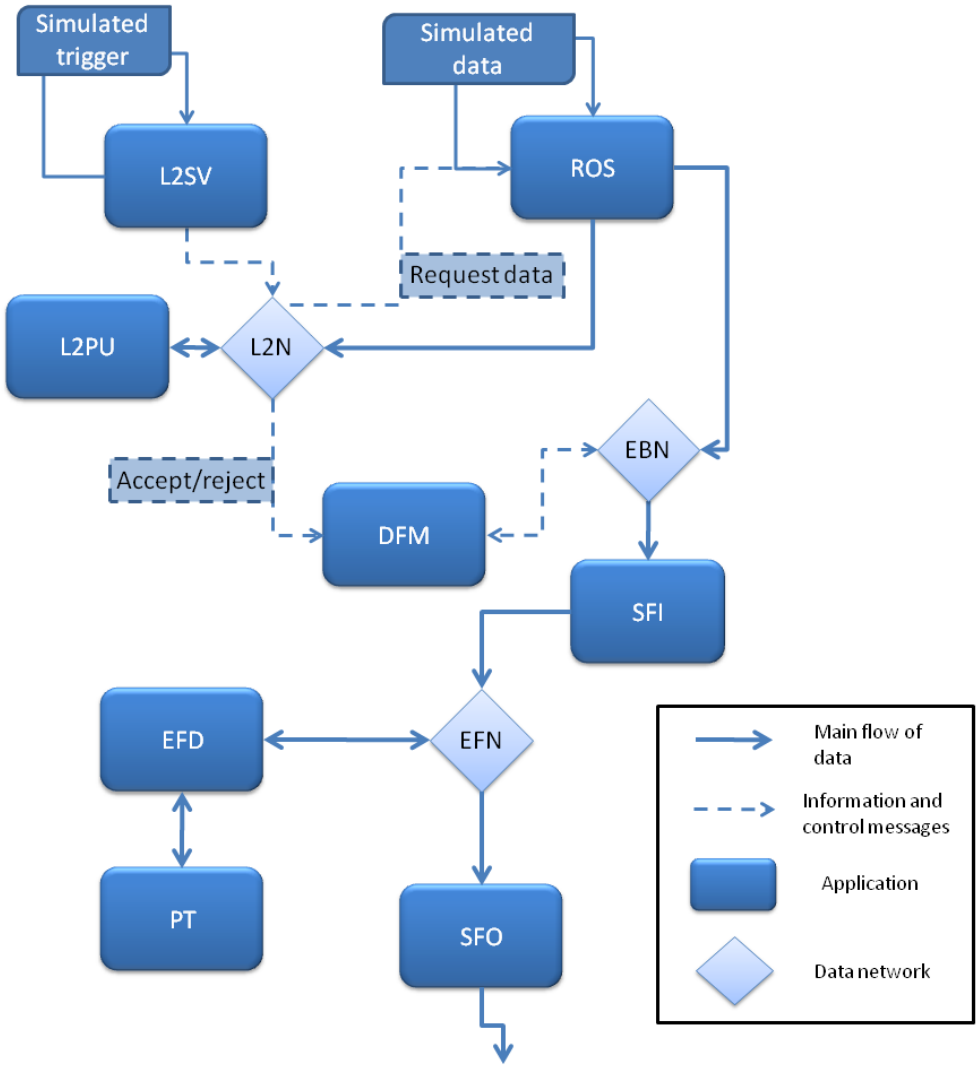


Figure 5.1: The flow of information (dashed arrows) and data (bold arrows) in the experimental setup. Please refer to Table 5.1 for a description of the different applications.

Table 5.1: Description of the components in the experimental setup

Abbreviation	Name	Description
ROS	Read out System	A part of the system responsible for receiving data from the detector electronics.
L2SV	Level 2 supervisor	Responsible for distributing data within the level 2 system and communicating accept/reject decisions to the DFM.
L2PU	Level 2 processing unit	L2PUs are assigned tasks by the L2SV. Data is then requested from a ROS, processed and the result is communicated back to the L2SV.
L2N	Level 2 network	Data network passing information and data between level 2 system applications. Information from the L2SV is also passed on to the DFM (through the EBN network).
DFM	Dataflow manager	Receives information from the L2SV and orchestrates the correct flow of data between ROSs and SFIs.
SFI	Sub-farm input	The part of the event building system where data from the entire detector is collected into single units called events.
EFB	Event builder network	Data network passing all information in the event builder sub-system.
EFD	Event filter dataflow	Responsible for dataflow within the event filter where further filtering of the data is conducted.
EFN	Event filter network	Data network passing all information in the event filter sub-system.
PT	Processing Task	Part of the event filter. Responsible for making an accept/reject decision for a single event.
SFO	Sub-farm output	Receives the final events and forwards them to mass storage.

5.1.2 Multi-host-setup

In the second setup the TDAQ system was configured to utilise one of the available computing clusters at CERN which are specially designed for testing the TDAQ system. One can therefore ensure that no other processes are influencing the tests. The process types involved in this setup were the same as in Section 5.1.1, but the scale of the system is larger with a total of 36 computers involved and the number of applications were 57. Given that the real TDAQ system will ultimately contain 3000 computers and tens of thousands of applications this may seem insignificant or rather unrepresentative. However, there are several characteristics of the real system and the multi-host-setup that justifies the use of the multi-host-setup for testing. These are:

(i) Most of the applications in the TDAQ system are identical and are involved in the different stages of filtering. For example, 1500 of the 3000 computers are all dedicated to the Event Filter sub-system and are all running identical processes. Similarly, a large number of the computers and applications are involved in the other stages of the filtering process. Hence, the biggest difference between the test setup and the real system is in the rate of which data can flow through the system and not the actual behavior as such.

(ii) The multi-host-setup has a well defined network topology. This means that there are dedicated hosts for running different sub-systems. This is important both in terms of the behavior of the system as a high load in one sub-system will not affect another part. It also means that any information gathered from the system containing the computer/host name will implicitly carry some information about which sub-system it is relevant to.

(iii) The multi-host-setup has separated control and dataflow networks as in the real TDAQ system. This ensures that activity and/or failures in one of the networks will not affect the other.

However, some situations are not easily reproducible. Some of these are:

(i) Information from external systems. In the real TDAQ system information is received from external systems such as the detector control system (controlling voltage, gas temperatures, etc), the accelerator (beam status, configuration parameters) and the networking system (operational status of switches and network).

(ii) The size of the real ATLAS system is of such a scale the some errors that are normally very rare, e.g. disk failures, will occur frequently.

Overall the system is still sufficiently similar in behavior, and the data resulting from the use of the multi-host setup should be sufficiently similar to the real ATLAS system so that different techniques may be tested using data from this setup.

Ultimately, the methods developed must naturally be incorporated and tested in the real ATLAS system in due course.

The data gathered using the single-host-setup formed the basis for the work in (Sloper et al., 2008). Since then the work has progressed to the use of the multi-host setup and this forms the basis for the work presented in this thesis.

5.1.3 Error scenarios

There are naturally a vast number of possible error scenarios one could investigate, however in this thesis focus will be dedicated to situations where one or more applications are not producing data that is expected. Note that as mentioned in Section 4.4.1, if an application actually terminates this will be detected immediately by its controller and can be dealt with appropriately. This type of error was chosen due to the fact that such situations are relatively common in the real TDAQ system and are frequently observed by operators. There are a number of underlying reasons for this kind of error and they include:

(i) Internal failure in the process. This could lead to the process not producing data or producing corrupted data or simply operating at a lower rate.

(ii) Network problems could hinder the transmission of data between the processes. While this is not a problem for the process itself, the effect is in some cases the same as if a process stopped producing data.

(iii) Problems related to the operating system could stop a process from executing as expected. This could be anything from lack of memory or disk space to problems with process scheduling and priority.

In order to simulate this type of error POSIX signals were utilised. POSIX is an IEEE standard (ISO/IEC 9945) for user and software application interface to the operating system and is supported by the UNIX systems we use for our experiments. Among others, it defines an interface that allows us to send signals/commands to any running process. There are a number of signals available, however the ones most

useful in terms of simulating errors are SIGSTOP and SIGCONT. SIGSTOP will temporarily stop the execution of the process while SIGCONT will resume it. Using these two signals it allows us to simulate the situation where a process is no longer working properly (or rather not at all) without actually terminating the process. We can then resume its operation whenever we choose, thus simulating a situation where they are not producing any data for a given period of time or with reduced efficiency.

This process was repeated for three different *types* of applications, namely the ROS applications, L2PU applications and SFO applications (choosing a different application each time). These applications all take part in the dataflow and are therefore good candidates for determining whether error detection based on ERS and IS data is viable. In total 23 different applications were temporarily stopped and restarted. Hence, there are 23 time periods in which the system is in error, each period representing one of three possible error types. For simplicity these errors will from now on be referred to as TypeI, TypeII and TypeIII error respectively.

5.1.4 Datasets

As discussed in Section 4.4.2 there are two main sources of information, the ERS messages stemming from the applications in the system and the information published[1] in the IS. Throughout the error simulations all ERS messages were recorded by the LogService and stored in a database from where they are subsequently retrieved. All information published to the IS by the applications shown in Figure 5.1 were also recorded. As the different application publish information at different intervals this naturally puts a limit to the frequency of which the IS data can be sampled/recorded. The IS data was therefore recorded at intervals of 2 seconds as this accounted for the fastest rate of updates from any single application and therefore ensured that no information was missed. As the information is recorded at specific time intervals this means that the IS dataset naturally take the form of

[1] *Publishing* here refers to the action of sending information to the IS service were it can be accessed by other application (see Section 2.1.1.5 for further details of the IS system)

time series.

The ERS and IS datasets cover the same time frame and error scenarios. The ERS dataset consisted of 4258 messages stemming from a variety of applications in the system. The IS dataset consisted of 978 measurements of 107 different structures/datasets published by differnet applications. Concatenating all the IS information this meant 978x1588 though many of the variables are repeated in different datasets. The IS and ERS datasets are presented in further detail, then preprocessed and analysed in Section 5.3.1 and Section 5.3.2 respectively.

In the following section we will look more closely at some methods for analysing and visualising these datasets.

5.2 Analysis methods

This section gives an introduction to the background theory of the preprocessing, analysis and visualisation methods used in subsequent sections.

5.2.1 Principal component analysis

Principal component analysis (PCA) is a mathematical method that reduces the number of dimensions in a dataset while retaining as much of the variance as possible. The method has a wide range of uses including utility in areas such as data mining, dimensionality reduction and de-correlation, pattern recognition and (lossy) data compression (Liu et al., 2006; Tien et al., 2-6 Nov. 2004). It will in this chapter be used to help identify the key components of the dataset, and allow for the elimination of redundant information and to filter out noise.

PCA is realised by projecting the original M dimensions of data into a D dimensional space where $D <= M$. This D dimensional space consists of orthogonal vectors called principal components. The principal components are chosen in such a way that a maximum of variance is achieved and they are ordered by the amount of variance explained by each one. In this way the first principal component explains more variance than the second and so on. By choosing $D < M$ one can therefore

project the data into a lower dimensional space while retaining as much variance as possible. PCA is purely based on the variance in the data and is therefore an unsupervised method.

There exist a number of variants of PCA (Wei-min and Chein-I, 23-28 July 2007) (Tien et al., 2-6 Nov. 2004), however the initial tests were performed using the standard form as explained in this section. As can be seen through the analysis Section 5.3 the results gained through PCA correspond well to that of the other approaches. There were therefore no obvious need to test other variants of PCA. Unless otherwise stated the MATLAB® implementation of PCA was used throughout this thesis.

5.2.1.1 PCA visualisation

There are a number of ways to visualise the results of the PCA. In this chapter two main approaches are used:

1. After the PCA has been performed the data are plotted in the principal component space. This can be done in two or three dimensions as shown in Figure 5.2 where the ERS data is plotted. The first three subplots show the data projected onto respectively the 1st and 2nd, the 1st and 3rd and the 2nd and 3rd principal components. The 4th plot shows a 3-dimensional plot of the data using the first three principal components. Such plots facilitates the inspection of the data and can be used to identify any clusters in the data.

2. The data are plotted by using the 1st and 2nd principal component as for the approach described above. However, the original variables are also projected onto the same dimensions. This is a good way of visualising how the different original variables 'take part' in the final results. An example of this plot is shown in Figure 5.3 where again the ERS data has been plotted.

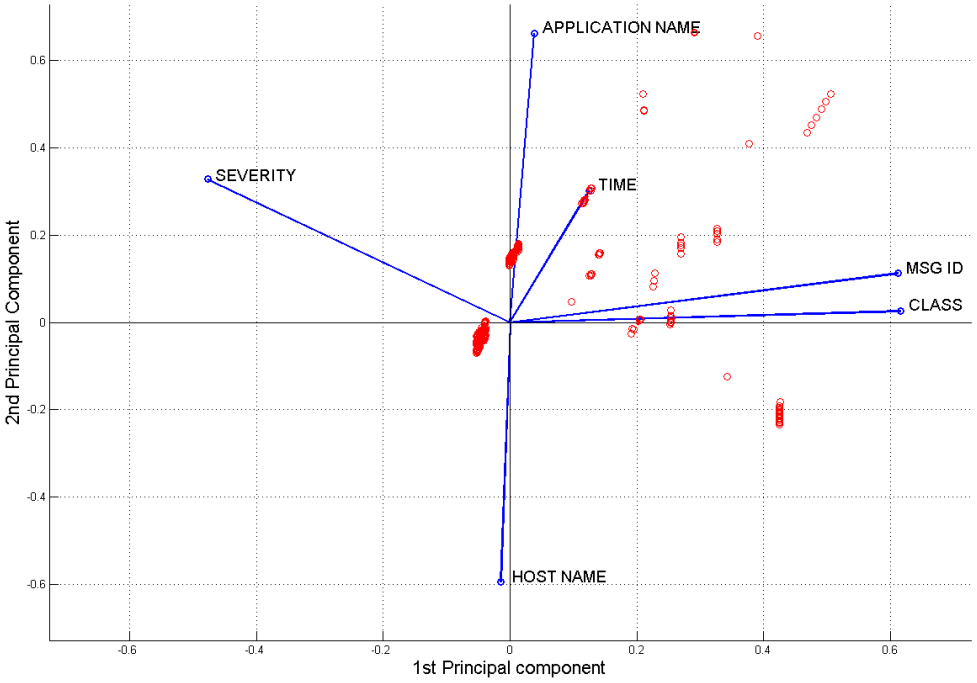


Figure 5.2: Principal component plot of example data.

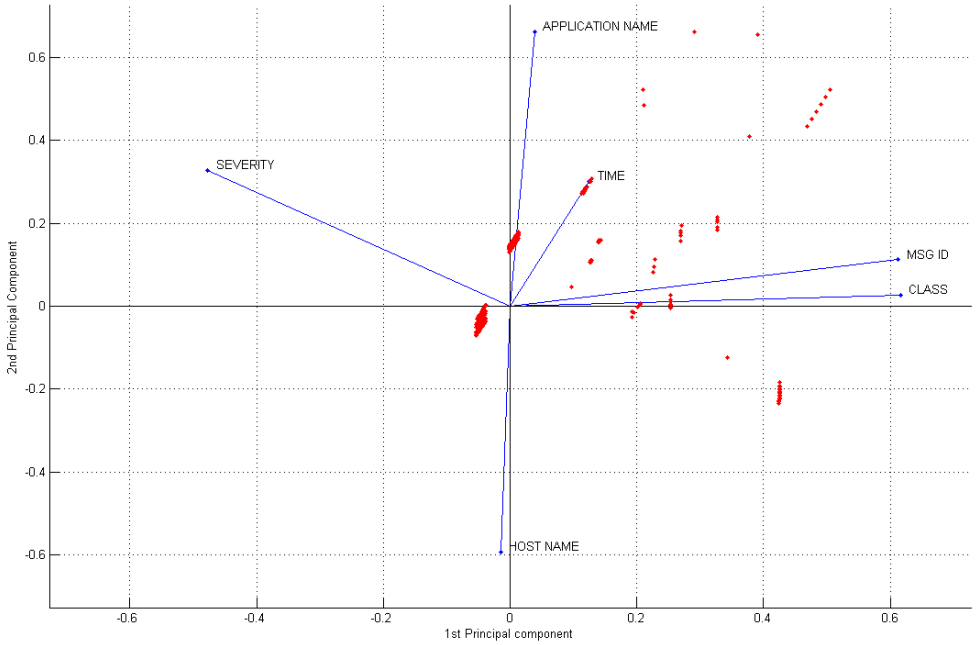


Figure 5.3: Principal component plot of data with the original dimensions projected onto the principal component plane.

5.2.2 Fuzzy C-Means clustering

Clustering, in general, is a method for dividing a set of data into two or more groups and is accomplished by identifying similarities between the individual data points and categorising groups where data points in that group exhibit some similarities. FCM clustering was first proposed in 1973 by J.C. Dunn (Dunn, 1973) and was further developed by Bezdek (Bezdek, 1981). It is a method where a data point is allowed to belong to two or more clusters according to a fuzzy membership function for each cluster.

More formally we can define it as follows: Given a set of points x_i, $i = 1, \ldots, n$ then for each point x_i, there exists a degree of membership to a cluster j represented as U_{ij}. Each cluster centre C_j is calculated using (5.1). Membership of a value in a cluster is computed using (5.2).

$$C_j = \frac{\sum_i^n U_{ij}^m}{\sum_i^n U_{ij}} \tag{5.1}$$

$$U_{ij} = \frac{1}{\sum_{k=1}^{l} \left(\frac{||x_i - C_j||}{||x_i - C_k||} \right)^{\frac{2}{m-1}}} \tag{5.2}$$

5.2.2.1 FCM algorithm

Algorithm 5.1 describes the general FCM algorithm. By adjusting the parameter m one can change the extent to which degree the distance from a centre affects the degree of membership. If m is closer to 1 the function behaves more like k-means clustering (MacQueen, 1967) (i.e. membership in one cluster only).

5.2.3 The Self Organised Map (SOM)

The utilisation of SOMs is another technique which has widespread use in data mining, classification and data visualisation. The SOM was first introduced by Kohonen (Kohonen, 1982) and is thus sometimes referred to as a Kohonen network. SOM is based on unsupervised learning, meaning that it does not use any a priori knowledge about the data to be clustered. It is capable of finding classes/clusters

Algorithm 5.1 FCM algorithm

1: Choose a number of clusters l, a convergence threshold ϵ and the factor $1 < m < \infty$.
2: Assign randomly the membership of each point to the different clusters.
3: **while** $||U^{t+1} - U^t|| > \epsilon$ **do**
4: Compute cluster centers using (5.1).
5: Update the membership vector using (5.2).
6: **end while**

inherent in the data, though depending on the dataset this might not always be true.

SOM is a type of ANN, consisting of an input layer and a map layer. All nodes in the map layer are connected, using weighted connections, to each of the nodes in the input layer. The SOM algorithm works by iteratively finding the map nodes that are the most 'similar' to a given input and updating its weights so that it fits the input better. Similarity is measured using the Euclidean distance between the data points.

5.2.3.1 SOM topology

The number of nodes in the input layer corresponds to the number of dimensions in the dataset to be clustered. The number of nodes in the map layer can be defined to adjust the granularity of the clustering. A number of possible SOM topologies exist, however the one used in this chapter is one where the map units are organised using a hexagonal lattice structure as can be seen in Figure 5.4. In this topology each adjacent unit is considered a neighbour giving an intuitive neighbourhood model which is used during the training of the network (see Section 5.2.3.2).

Before training the map units are organised in a two-dimensional grid as shown in Figure 5.5a. However, after training the map is capable of retaining the topology of higher dimensional input data. An example of a trained grid is shown in Figure 5.5b. The SOM was in this case trained using some example data and the grid and input data have been projected into a three dimensional space using PCA.

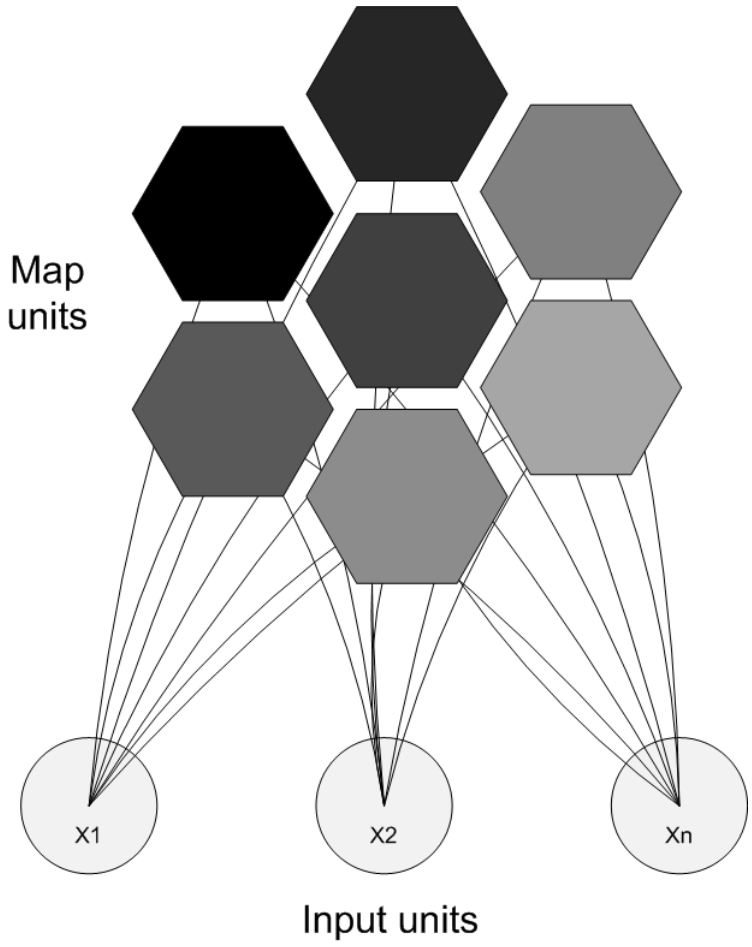
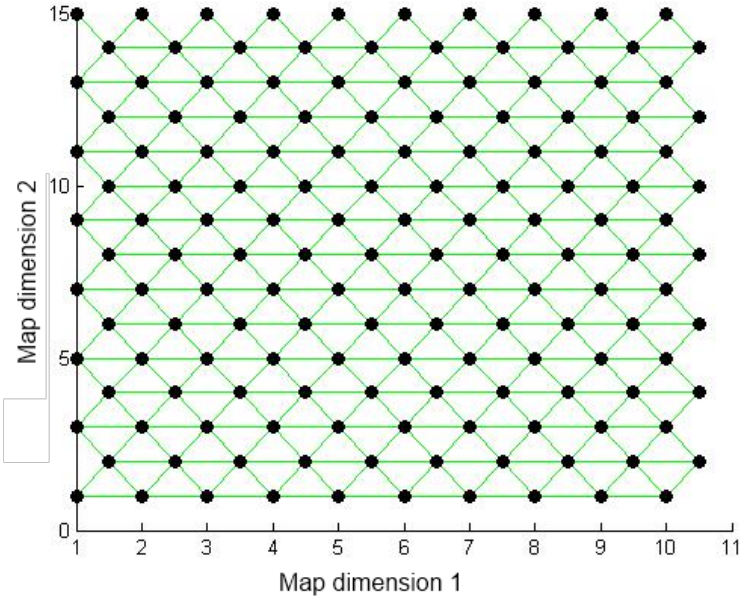
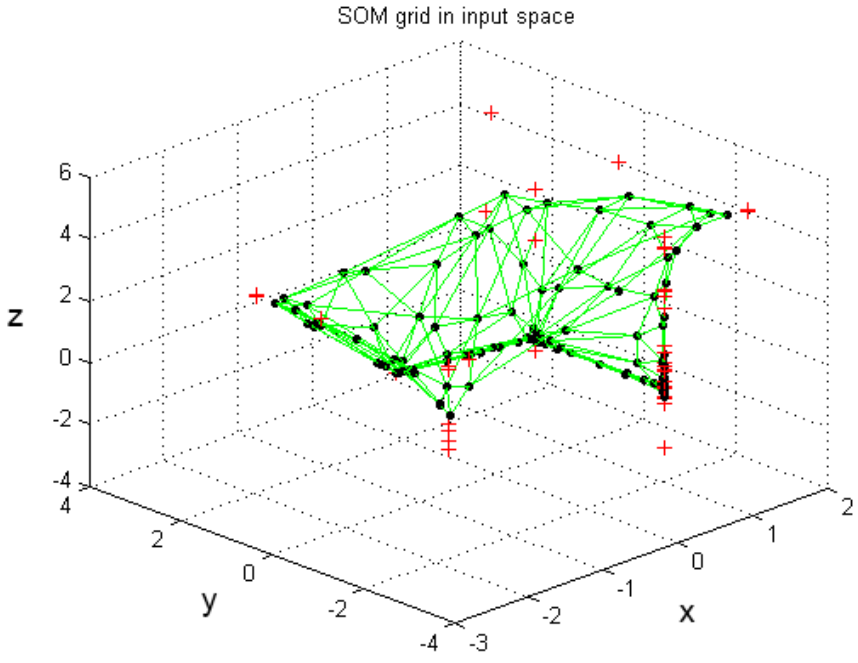


Figure 5.4: A SOM organised using a hexagonal lattice structure.



(a) Untrained SOM



(b) Trained SOM

Figure 5.5: The SOM grid before and after training. In a) an untrained SOM map is shown and b) shows the trained SOM projected into the input space together with the input data.

5.2.3.2 SOM algorithm

Algorithm 5.2 describes the general SOM algorithm. Each of the input data is iteratively presented to the SOM and the map-node closest to the input in terms of Euclidean distance is referred to as the Best Matching Unit (BMU). All nodes in the neighborhood are then updated in order to be more similar to the input that was presented. Determining neighbors can be done in a number of ways. One common method is to choose map nodes that directly neighbor the BMU according to the map topology. Hence for a hexagonal structure there are 6 immediate neighbours. The neighbourhood size can be adjusted to affect the learning process of the map.

Variants of the algorithm include the popular 'batch' algorithm which presents a group or batch of inputs to the map. For each of the input records in the batch the algorithm keeps track of the corresponding best matching unit (BMU). Then the map units are updated by using the average of the input vectors in its neighborhood. This is repeated for a number of batches. This training method is less computationally expensive than the iterative one, but has still been shown to produce good results, hence it will be used in the subsequent work.

Algorithm 5.2 The iterative SOM algorithm

1: Initialise the map unit weights.
2: **while** more input vectors exists **do**
3: Choose an input vector.
4: Iterate over all map units and find the one closest to the input in terms of Euclidean distance. This node is referred to as the best matching unit (BMU).
5: Update the BMU and its neighbors (as defined by the neighborhood function) by adjusting their weights to make them more similar to the input vector.
6: **end while**

5.2.3.3 SOM visualisation

There exist a number of visualisation methods for presenting a SOM. The most common is the Unified distance matrix (U-matrix) which displays a number of cells, where each cell corresponds to a unit in the SOM map. The U-matrix displays the distance between the map units, usually using a color coding. This can be done for the entire map as in Figure 5.6, or per dimension/attribute in the input data; as in Figure 5.7.

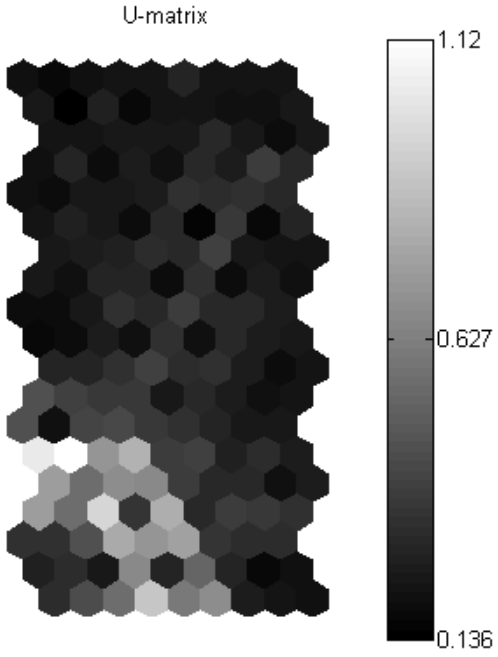


Figure 5.6: Unified distance matrix of a trained SOM.

5.2.4 Mann-Kendall trend detection and Sen slope

Mann-Kendall is a statistical test for trend detection (Mann, 1945; Kendall, 1975). The algorithm essentially proceeds as follows: For a given dataset of size n

$$T = x_1, x_2, \ldots, x_n$$

calculate the statistics S using (5.3).

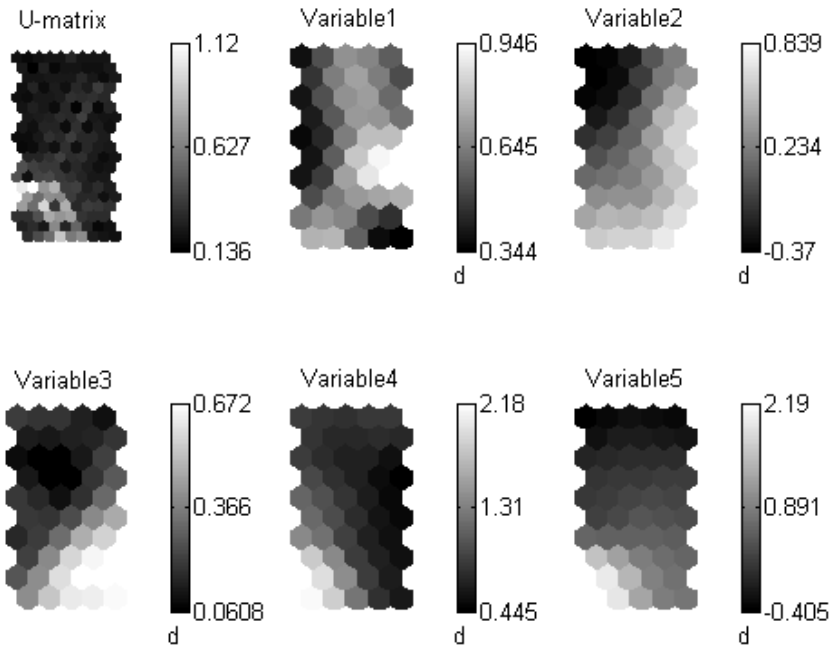


Figure 5.7: Unified distance matrix of the map and of each dimension in the input data.

$$S = \sum_{i=1}^{n} \sum_{j=i+1}^{n} y_j i,\ where\ y_i j = \left\{ \begin{array}{c} 1\ if\ x_j > x_i \\ 0\ if\ x_j = x_i \\ -1\ if\ x_j < x_i \end{array} \right\} \tag{5.3}$$

S is then used to calculate the statistic Z as shown in (5.4) which is then used in a Z-test with [H0: No trend] and [H1: A trend exists]. Hence the result of the test will always be 1 or 0 representing that a trend is present or not respectively. $Var(S)$ is the normal approximation of variance and gives good results as long as the number of samples n is greater than 10.

$$Z = \left\{ \begin{array}{l} \frac{(S-1)}{\sqrt{Var(S)}},\ S > 0 \\ \frac{(S+1)}{\sqrt{Var(S)}},\ S < 0 \end{array} \right\},\ \ Var(S) = \frac{(n(n-1)(2n+5))}{18} \tag{5.4}$$

5.2.4.1 Sen slope

The Sen slope of a dataset is a measure of the slope of the trend, i.e. the rate at which the data is changing. Given a dataset of n observations it is found by calculating the values Qji between all points in the sample as shown in 5.5. The Sen slope is then defined as the *median* of the set Q.

$$Q_ij = (x_j - x_i)/(j - i),\ i < j < n \tag{5.5}$$

5.3 Analysis of the datasets

In order to understand the data and to investigate whether error detection is likely to be successful based on the dataset preprocessing, analysis and visualisation of the data has been performed. In the following sections the two datasets, IS and ERS, are investigated in turn.

5.3.1 IS data

The IS datasets consists of 107 message structures published by 37 different applications, hence some applications publish more than one information structure (here meaning a set of information). All these applications take part in the flow of data in the experimental setup. The information published by each application was stored periodically throughout the experimental tests and the corresponding time series were then created.

Figure 5.8 shows an example of the information published by one of the L2PU application taking part in the experimental setup. The figure shows a time series plot for each piece of the information published by that particular L2PU application.

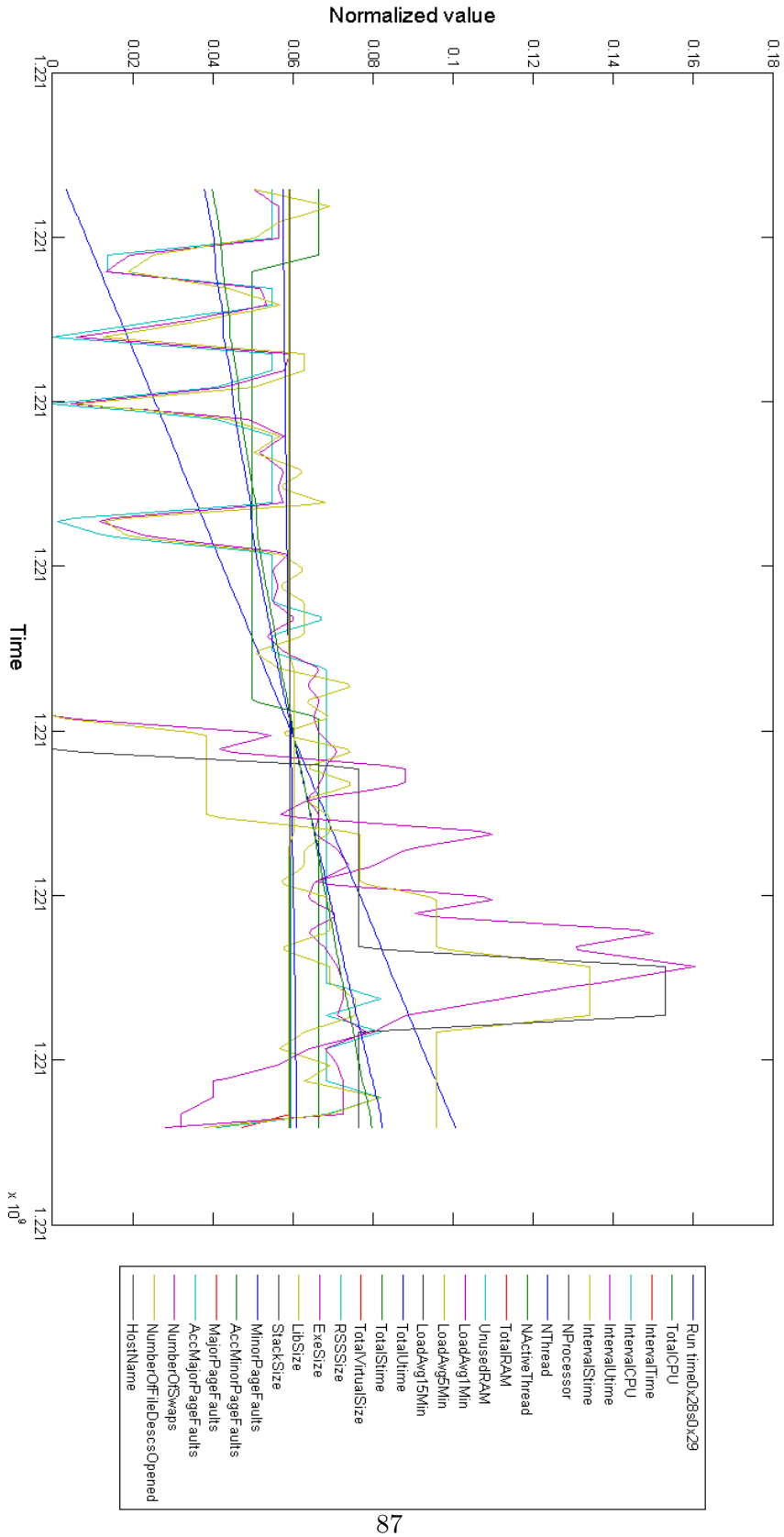


Figure 5.8: Example data published in the Information Service (IS) by an application in the TDAQ system.

5.3.1.1 Pre-processing

In order to work with the IS data some pre-processing was necessary. While most of the data is available as numerical values, some exceptions exist where the data is available only as text. These need to be transformed into numerical values which was accomplished using a simple hash function. The pseudocode for the hash function is shown in Algorithm 5.3. This particular hash functions was chosen for several reasons:

(i) The algorithm is simple and fast. This may be of particular importance if preprocessing is to be performed 'on-line' and a limited amount of time is available.

(ii) The algorithm puts emphasis on the leading characters. Typical hash functions aim to provide random numbers even if the initial string values are very similar and no emphasis is put on any parts of the strings. Using the algorithm described here means that text values that start similarly will produce numerical values that are close. This is important as much of the textual data contains a structure which encodes some information about the underlying system. An example is the computer/host name which is created using a specific pattern. Consider the host name 'pc-TDAQ-one-01'. In this case the 'pc' indicates that it is a computer, 'TDAQ' indicates that it is a part of the TDAQ system and 'one' specifies that it is a part of the online software. Hence, using the outlined hash function computers within the online part of the TDAQ system will all correspond to very similar numerical values. As each sub-system is assigned to a specific group of hosts and is named correspondingly this means that the numerical values will maintain this inherent 'grouping' in the textual data.

Based on these factors this algorithm is better suited for this particular type than an algorithm that produces more randomly distributed numbers, something that many (if not most) hash functions are designed to do.

Algorithm 5.3 Calculate hash value of string S

1: $HashValue = 0$;
2: $k = 1$;
3: **for all** characters c in string S **do**
4: $\quad HashValue+ = ascii\_value(c) * 2^{(24-k)}$
5: $\quad k++$
6: **end for**
7: **return** HashValue;

5.3.1.2 Applying PCA to the IS data

After converting the textual data into numerical ones a total of 978 measurements exists for the IS data containing a total of 1588 variables. A number of these variables are of no interest. In particular observations that are constant or increasing/decreasing with a constant value throughout the measurements do not contribute to the variance of the data and will therefore not provide any important information.

In order to reduce the amount of variables PCA analysis was performed for each of the 107 information structures retaining only the three most significant principal components. This resulted in a dataset of size 978x321. All constant values and values which are monotonically changing are then removed leaving a final dataset of size 978x234. Again, we remove those values that are linear as they do not contribute to the variance of the dataset.

This dataset was then further reduced again using PCA. Figure 5.9 shows that virtually all the variance in the data can be explained by retaining just the 3 most significant principal components. However, following further inspection of the resulting datasets some problems were found. Figure 5.10 shows the dataset plotted using the 3 most significant principal components. The observations corresponding to the different types of errors do not form any obvious clusters in the dataset. Indeed they are interleaved with observations corresponding to non-error situations. Based on this it is not likely that error detection techniques could be developed to

predict these types of errors based on this dataset.

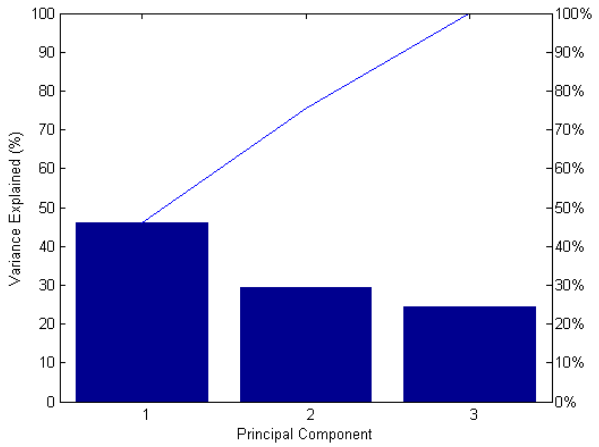


Figure 5.9: Cumulative variance explained by the first 3 principal components.

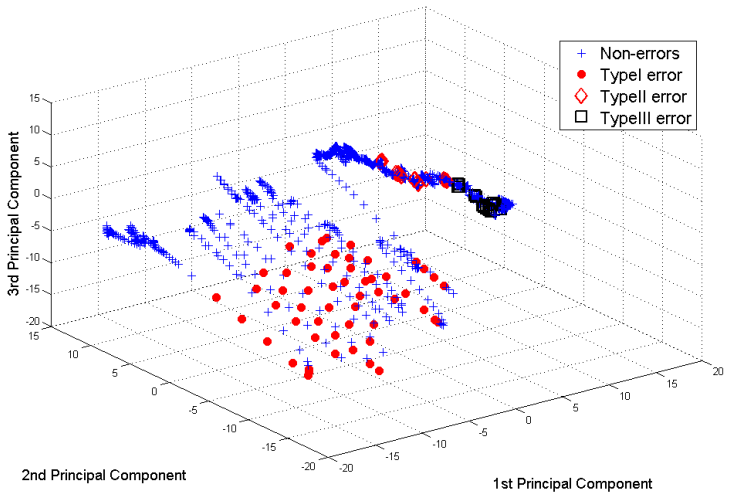


Figure 5.10: Principal component analysis of the IS data.

In order to improve upon this a different approach to processing the data was taken. As the IS data is inherently time-series using a trend detection algorithm could lead to better results. Such an approach also has the advantage that it does not rely on the actual values contained in the dataset, but rather the trends and relative changes within the dataset. Utilising such an approach will therefore be

more robust in terms of handling different datasets gathered using other forms of setups of the TDAQ system. The dataset was preprocessed as before but each of the observations were analysed using the Mann-Kendall algorithm and the Sen slope was calculated as described in Section 5.2.4. The trend and Sen slope were calculated using 10 seconds intervals. This interval was chosen as the delay in detection must not be too large while still containing enough relevant information to determine whether a trend is present or not. In order to verify that the 'correct' interval was chosen a visual inspection was performed comparing the result of the Mann-Kendall algorithm to the observed values over a number of different intervals. 10 seconds was found to give a good tradeoff giving a good measure while being within a reasonable time-limit to allow error detection to take place.

After calculating the trend and Sen slope for all the observations PCA was again applied in order to reduce the dataset. The first 25 principal components were kept which explained 80% of the variance in the data. These were chosen as each component beyond 25 accounts for less than 0.5% of the variance in the data. Hence, the limit was set in order to keep the number of variables limited so that the supervised learning procedures applied in Chapters 6, 7 and 8 can operate efficiently. Figure 5.11 shows the resulting dataset plotted in terms of the first 3 principal components. It can be seen that the observations corresponding to the error situations are better grouped and easier to separate from the other observations. This data was therefore utilised for the following analysis and in the subsequent chapters.

5.3.1.3 SOM analysis and visualisation

In order to identify possible clusters in the data a SOM was trained using the entire 978x234 dataset. The U-matrix of the trained map is shown in Figure 5.12a. The SOM map was then clustered using FCM. The clustering was performed using between 2 and 12 clusters and for each number of clusters FCM was run a number of times and the best result was recorded. The FCM is run multiple times as the results can vary due to random initialisation of the FCM. Running the clustering 5 times for each cluster number was found to be a reasonable trade-off in terms of

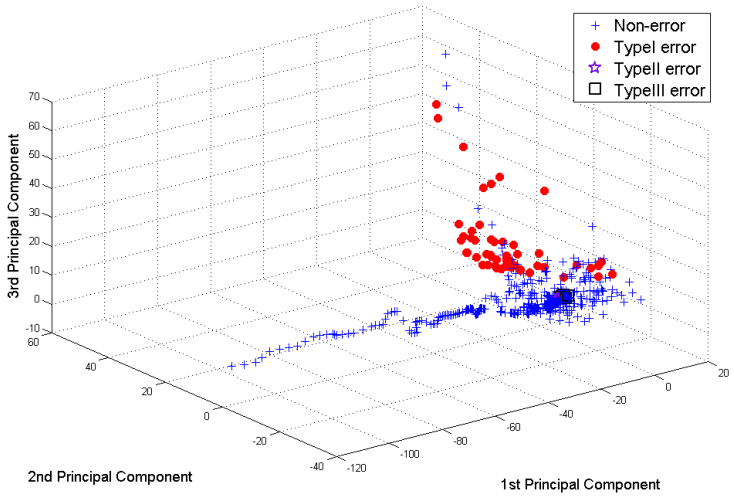
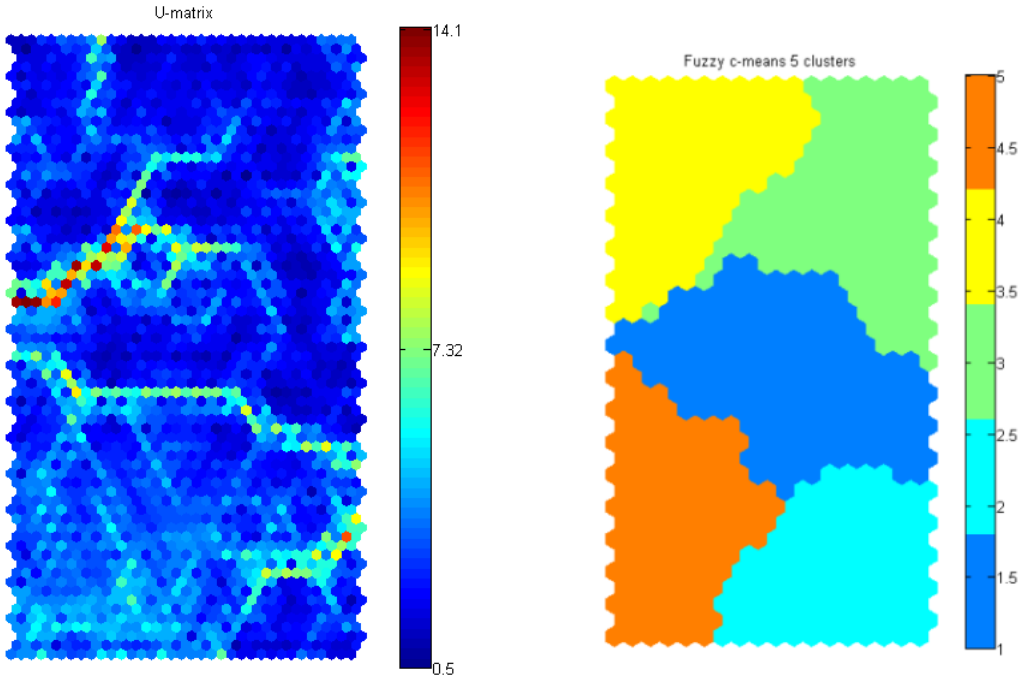


Figure 5.11: Principal component analysis of the IS data processed using the trend detection approach.

efficiency and accuracy. An optimal of 5 clusters were obtained using the Davies-Bouldin (DB) index (Davies and Bouldin, 1979). The DB index has been shown to compare well to other methods of choosing clusters (Bezdek and Pal, 1998). The result of the FCM clustering of the SOM map is shown in Figure 5.12b, and the DB index and Sum Squared Error (SSE) for the clustering process are shown in Figure 5.13.

5.3.1.4 Observations

There appears to be some distinct clusters in the data. While it is not trivial to draw any conclusions based on the analysis so far, it does seem likely that some of these clusters could correspond well to the different error scenarios that were introduced while gathering the dataset. In Chapter 6, 7 and 8 different ITS techniques will be trained to learn from these datasets showing whether or not there are indeed patterns corresponding to the error scenarios.



(a) U-Matrix (b) FCM clustering of U-Matrix

Figure 5.12: In a) the U-matrix of the trained SOM for the IS data is shown. In b) the FCM clustering of the trained SOM.

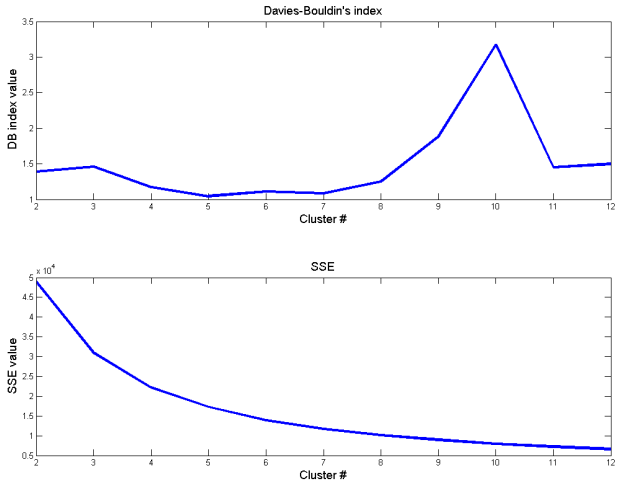


Figure 5.13: Davies-Bouldin index and SSE for the FCM clustering of the SOM map.

5.3.2 ERS data

The ERS dataset consists of 4258 messages produced by all applications in the setup. As all messages are by default stored in an database the messages could be easily retrieved after the experiments where concluded.

5.3.2.1 Initial overview

Looking first at how the messages are distributed in time several observations can be made. There are some distinct periods where a large number of messages are being sent. These periods correspond well with the times when the TypeI errors where introduced into the system. However there are few or no messages corresponding to the TypeII and TypeIII errors. The message distribution is shown in Figure 5.14 where the start and end time of the introduced errors are marked with vertical lines.

Figure 5.15 shows the number of messages per application. One can see that the majority of messages originate from the L2PU type applications. Also a number of messages arrived from the EFD, SFI and SFO applications. All these applications play a major part in the chain of data in the system, and are therefore affected by the introduced errors. Hence it is to be expected that a majority of messages originates from these applications.

5.3.2.2 Pre-processing

In order to perform the different types of data analysis such as PCA and SOM the data must first be pre-processed. The ERS data consists of a number of textual values which must be converted into numerical values. This was achieved, as for the IS data, by using the custom hash function described in Algorithm 5.3. The algorithm was chosen for that same reasons as described in Section 5.3.1.1 as the ERS data also contains textual information built using a specific structure which should be retained in the numerical data. After converting the textual values the data was then normalised so that all input lie in the range of [0,1].

After applying pre-processing the variance of the different variables can be determined as shown in Figure 5.16, though for some of the variables this depends

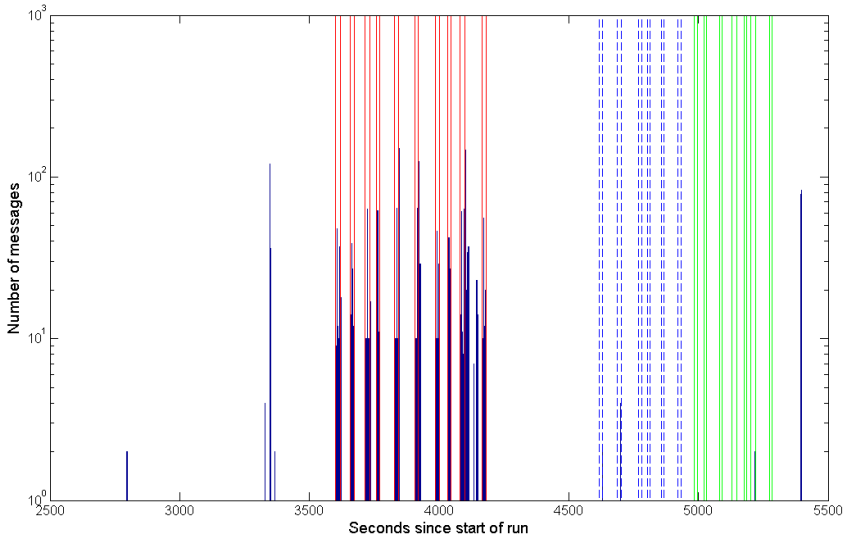


Figure 5.14: Number of messages in time. The vertical lines indicate the start and end time of the introduced errors.

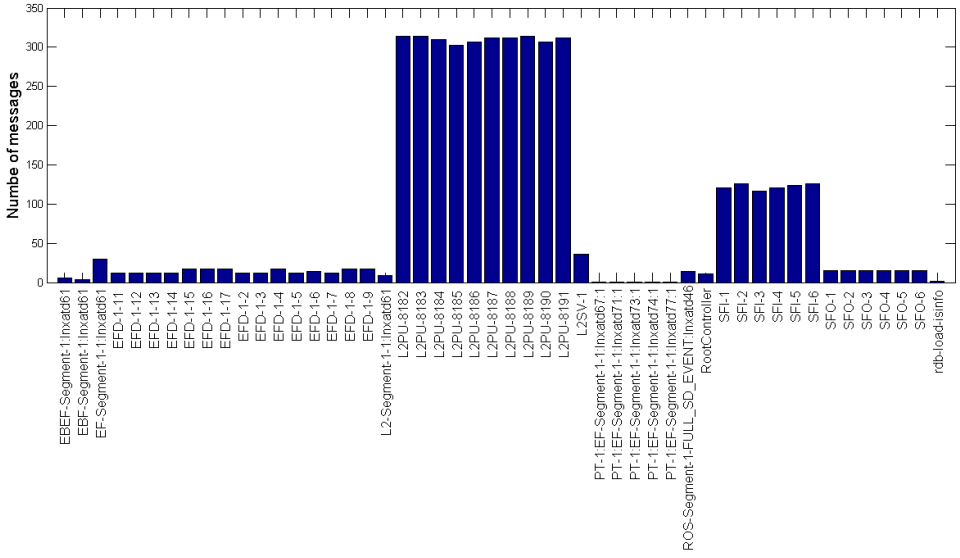


Figure 5.15: Number of messages sent by each application type.

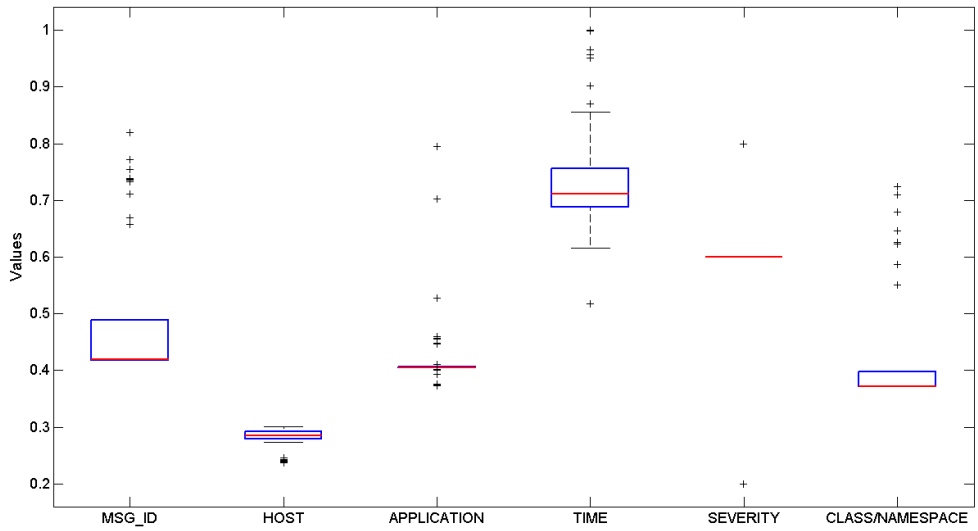


Figure 5.16: Variance in the ERS data after pre-processing

on how the mapping from textual to numerical values is done. In the figure the box lines represent the lower quartile, the median and the upper quartile. Outliers are defined as values larger than $q_3 + 1.5(q_3 - q_1)$ where q_3 is the upper quartile and q_1 is the lower quartile. The outliers are marked with a cross in the figure. For example, looking closely at the TIME variable one can observe that the majority of messages are grouped together with an upper quartile at approximately 0.75 and a lower quartile at 0.68. Hence, 50% of the messages are located within this range. There are also some outliers towards the end of the range. Looking back at Figure 5.14 one can see that this seems like a reasonable result as the majority of messages are grouped together (corresponding to the TypeI errors) and also contain a burst of messages towards the end.

5.3.2.3 Principal component analysis

PCA was performed using the dataset after the initial pre-processing. As can be seen from Figure 5.17 the first three components explain 91% of the variance in the dataset. With such a high percentage explained by the 3 most significant principal components one should be able to identify any clusters in the data by plotting the

data projected into these dimensions. Figure 5.18 shows the data projected into the first 3 principal components, first in pairs and then finally in a 3d plot with the first three principal components represented on the axes. Clearly, there are some clusters in the data. However, due to the dimensionality reduction that was performed combined with the discreet nature of the dataset there may be several observations corresponding to the same points in the plot. In order to visualise such cases in a better way, a plot where the marker size corresponds to the number of observations at the given point is shown in Figure 5.19. From this figure it can be seen that some of the points correspond to a large number of messages. This makes the clusters in the data even more evident and the fact that a single point in the PCA plot corresponds to a number of messages may indicate that a number of very similar messages where issued, for example only differing slightly in time.

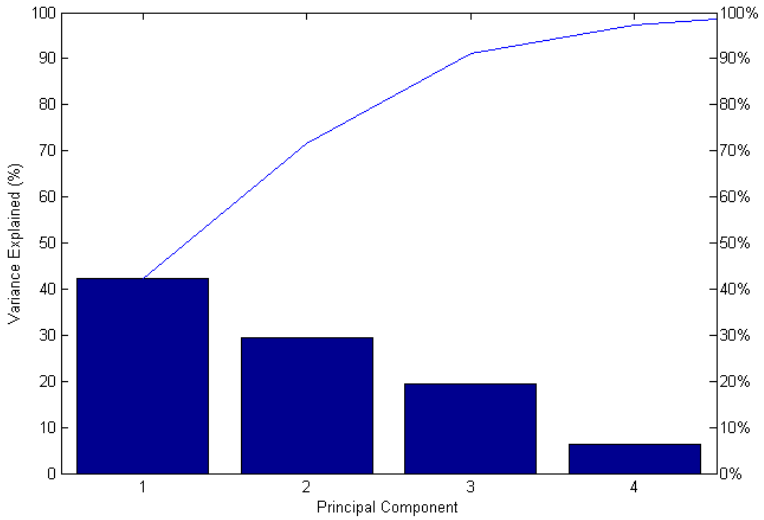


Figure 5.17: Variance explained by the 4 first principal components

By investigating which messages corresponds to the different points in the PCA plot it is evident that one of the larger clusters correspond to L2PU applications while another one corresponds to SFI applications. This is visualised by labeling the data points as shown in Figure 5.20. Note that some points are left unlabeled in order to maintain the readability of the figure.

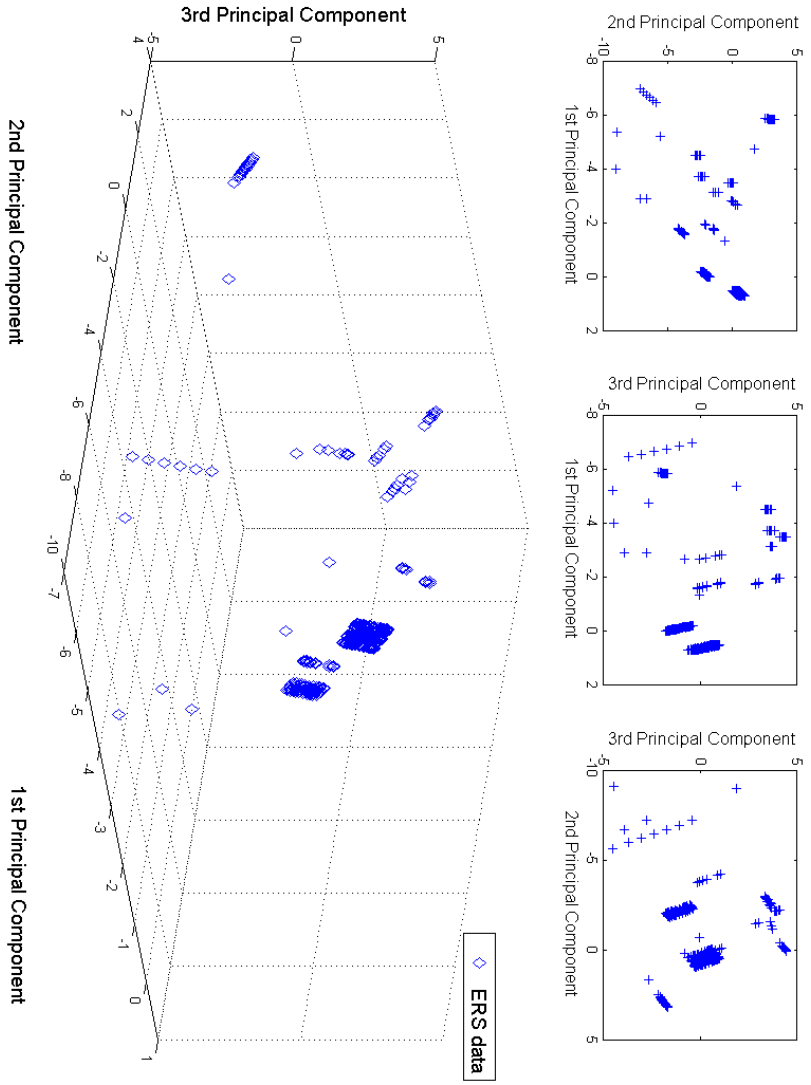


Figure 5.18: The ERS data projected onto the three first principal components

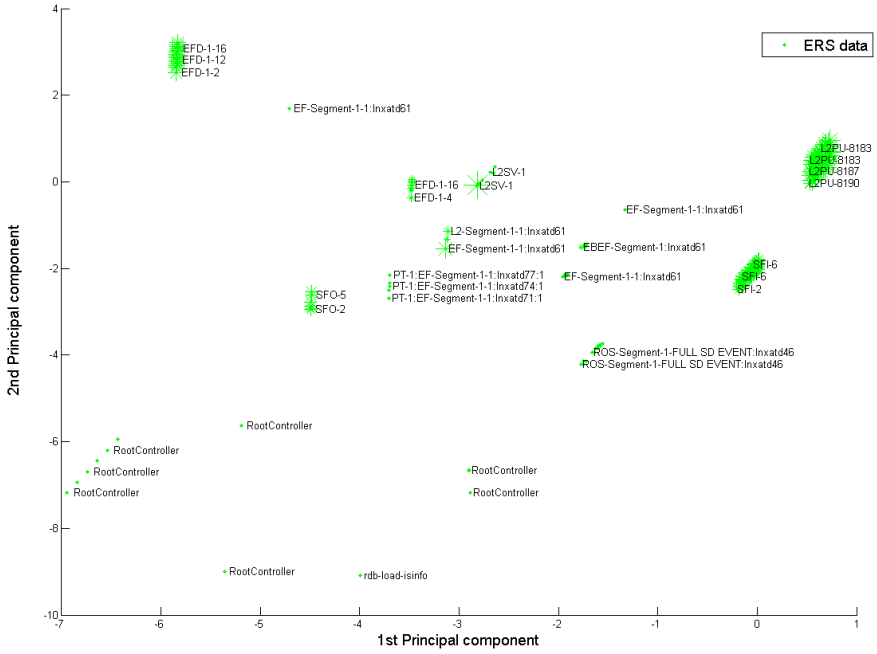


Figure 5.19: A principal component plot of the ERS dataset for the first two components. A larger marker indicates a greater number of observations at the given point.

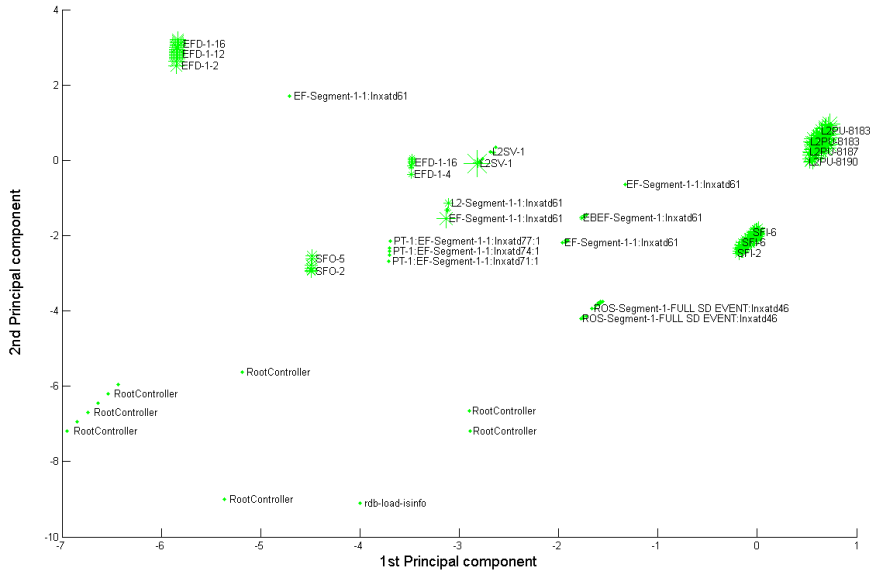


Figure 5.20: A principal component plot of the ERS dataset for the first two components with points labeled with the corresponding application name.

5.3.2.4 SOM analysis and visualisation

In order to verify and add to the result of the PCA, SOM analysis and visualisation was also performed. A SOM was created using default parameters in the MATLAB® implementation of SOM using a hexagonal lattice structure and the training was performed using the 'batch' training algorithm as described in Section 5.2.3.2.

The U-matrix for the trained SOM including maps of how the different attributes correspond to the map units are shown in Figure 5.21. From this it can be seen that some clusters exist in the data and it is possible to visualise how the different parameters take part in the clusters. For example, if one considers the cluster on the top right side of the U-matrix, one can observe that a corresponding area is present in each of the attribute maps hence the cluster corresponds to messages with the same name, severity, message id, machine name and name space. They are however, as can be seen from the corresponding attribute map, distributed somewhat in time. Similar interpretation could be made for the other clusters in the map, though the question as to how to determine a cluster will necessarily arise. In order to deal with this problem in an *automated* way, the identification of clusters in the SOM was done using FCM clustering (see Section 5.2.2 for more details) of the map units. FCM clustering was chosen as it provides an easy and automatic way of doing the clustering. FCM has been shown to give better results than that of other clustering techniques such as K-clustering (MacQueen, 1967). An optimum of 8 clusters was found using the DB index, and the clustering of the map can be seen in Figure 5.22.

5.3.2.5 ERS data for error detection

As discovered in Section 5.3.2.1 it is clear that for TypeII and TypeIII error types there are no or very few ERS messages being sent/issued in the system. It is therefore very unlikely that any error detection for these types of errors will currently be possible based on the ERS data. However, for the TypeI error there are some clusters in the data that seem to correspond very well to the time when the errors were introduced/simulated. By closer analysis it is evident that a large number of

the messages are being sent from the applications taking part in the data flow in the system. Based on this it is likely that such errors could be detected and classified by using the available ERS data.

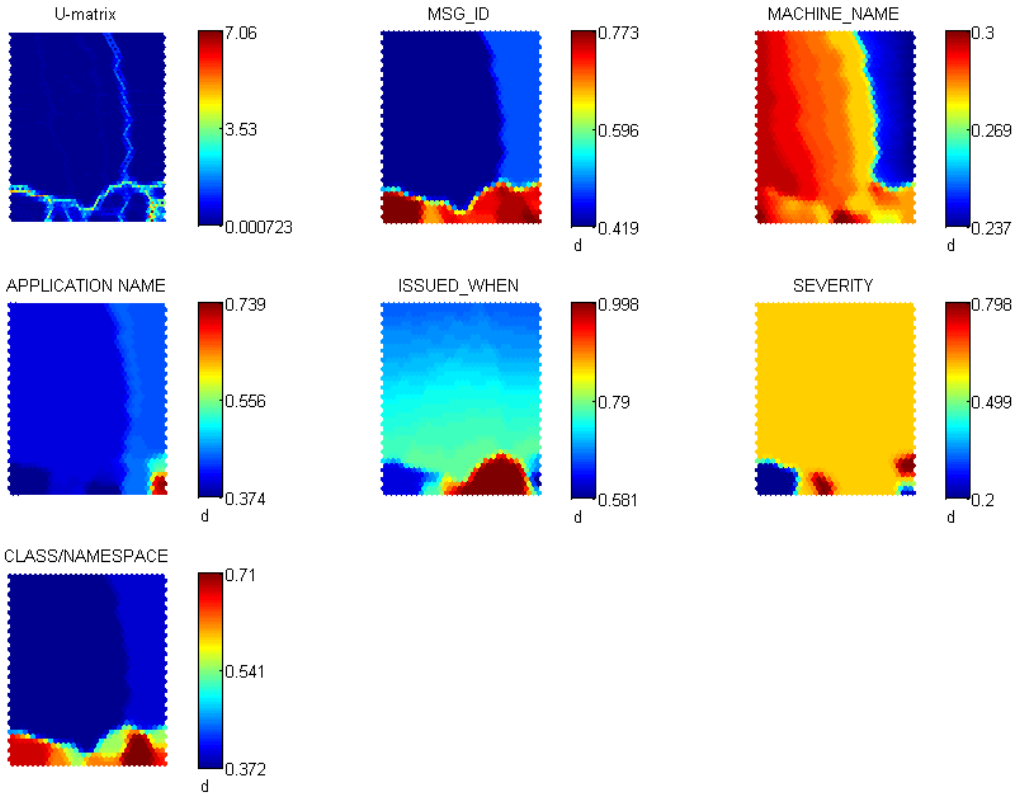


Figure 5.21: U-matrix and attribute maps for the SOM trained using the ERS data.

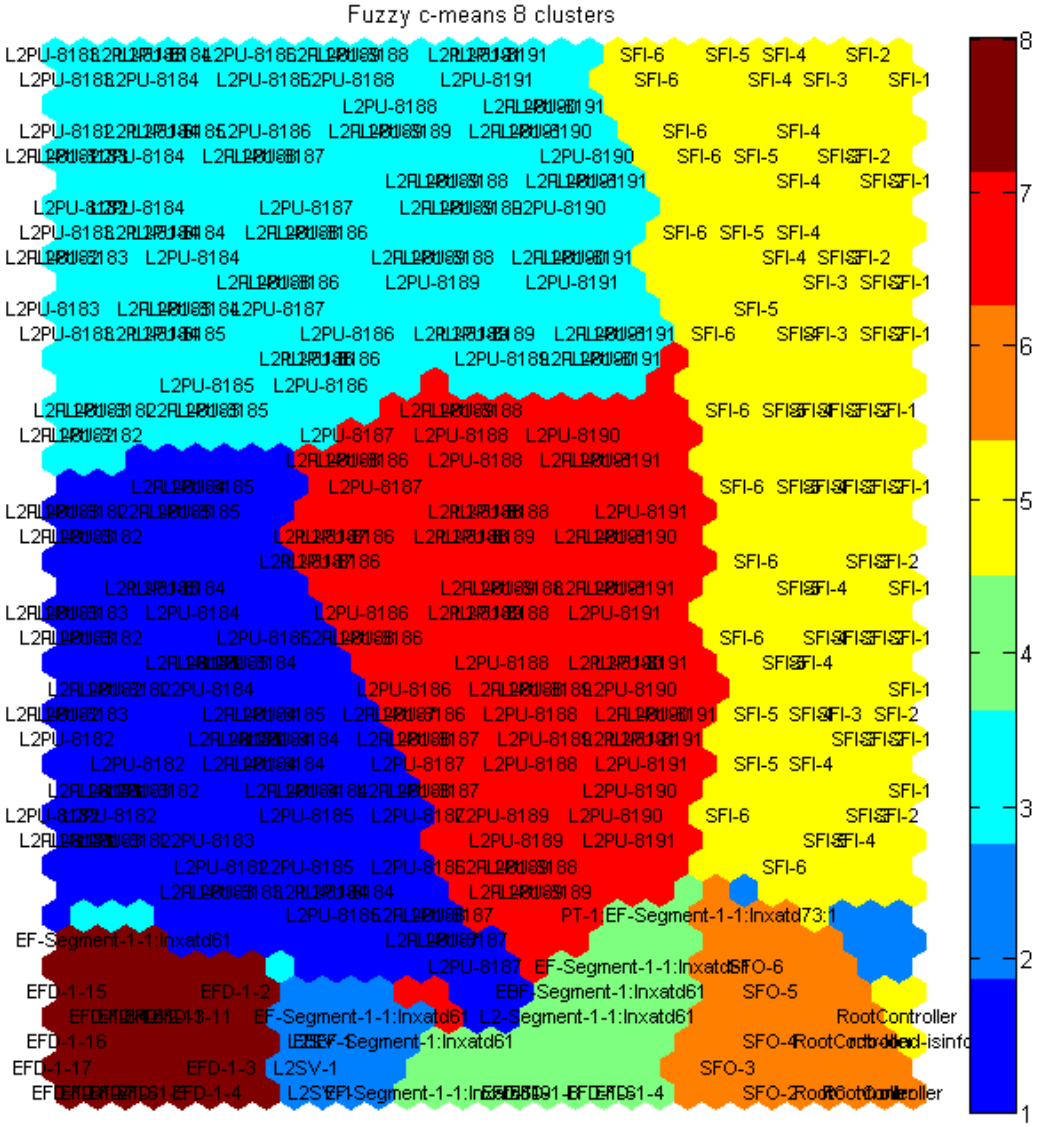


Figure 5.22: Result of clustering of the SOM using FCM.

5.4 Discussion and conclusions

Overall the presented techniques provide several ways to preprocess, analyse and visualise the datasets. The main aim of this chapter was to determine whether the datasets are suitable for training and testing different types of error detection systems. However, the techniques presented here may also be of use in other contexts and could provide a way to gain better knowledge of the effects of errors in the system.

For the IS data better results were achieved by first applying trend detection techniques to the data. After applying trend detection the clusters in the data were more clear and the observations corresponding to the error times were easier to separate, compared to the case where no trend detection was performed. PCA and SOM analysis both confirmed that there are clusters in the data and hence error detection is likely to be possible using the IS dataset.

For the ERS dataset, as discovered in Section 5.3.2.1, it is likely to be useful for detecting only one of the three introduced errors. This is not the expected behavior for the system and could be due to any number of reasons, which may include:

(i) The applications affected by the errors may not be detecting that anything is wrong, and are hence not issuing any errors or warnings.

(ii) The applications are indeed noticing something is wrong, but are not issuing any errors or warnings. This could be by choice of the developer or possibly by mistake. This can only be discovered by discussion with the developer and/or close inspection of the code.

(iii) The relevant applications may rely on information from other components to discover the problem. If for some reason this information is missing, the error might not be discovered and hence no messages will be issued by the application.

The exact reason can only be discovered through consultation with the relevant

system/application expert. While it is of course of importance to find out, it is not covered in the thesis work. This is due to the fact that such an investigation involves a large number of people and is an ongoing process. The author also does not have the 'authority' to enforce such a review and any time scales are therefore very difficult to estimate.

To summarise it is likely that error detection is possible based on the gathered datasets. In the following chapters several methods for developing error detection and classification systems will be investigated based on the datasets analysed in this chapter. In Chapter 6 different types of ANNs are investigated in order to detect and classify the different types of errors. In Chapter 7 a SVM approach using the same datasets will then be investigated and compared to the ANN approach. Finally, in Chapter 8 a CGP approach is investigated and compared to the results obtained using both the ANN and SVM approaches.

References

James C Bezdek. *Pattern recognition with fuzzy objective function algorithms.* Plenum, New York,, 1981.

J.C. Bezdek and N.R. Pal. Some new indexes of cluster validity. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28(3):301–315, 1998. ISSN 1083-4419. doi: 10.1109/3477.678624.

D.L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Trans. Pattern Anal. Machine Intelligence*, 1:224–227, 1979.

J.C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. 3, 1973.

M.G. Kendall. Rank correlation methods. London,, 1975. Griffin.

T Kohonen. Self-organised formation of topologically correct feature map. *Biological Cybernetics*, 43:56–69, 1982.

Yibing Liu, Zhiyong Ma, He Qian, and Peng Lv. Early identification of machine fault based on kernel principal components analysis. In *Proc. First International Conference on Innovative Computing, Information and Control ICICIC '06*, volume 3, pages 149–152, 30–01 Aug. 2006. doi: 10.1109/ICICIC.2006.446.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

Henry B. Mann. Nonparametric tests against trend. *Econometrica*, 13(3):245–259, 1945.

J.E. Sloper, G.L. Miotto, and E. Hines. Dynamic error recovery in the atlas tdaq system: Using multi-layer perceptron neural networks. In E. Hines, M. Leeson, M. M.Ramn, M. Pardo, E. Llobet, D. Iliescu, and J. Yang, editors, *Intelligent Systems: Techniques and Applications.*, chapter 8, pages 241–268. Shaker publishing, 2008.

D.X. Tien, K.-W. Lim, and L. Jun. Comparative study of pca approaches in process monitoring and fault detection. *Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE*, 3:2594–2599 Vol. 3, 2-6 Nov. 2004. doi: 10.1109/IECON.2004.1432212.

Liu Wei-min and Chang Chein-I. Variants of principal components analysis. *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*, pages 1083–1086, 23-28 July 2007. doi: 10.1109/IGARSS.2007.4422989.

Chapter 6

Classification of errors using Artificial Neural Networks

In the previous chapter the data gathered from the experimental setup were analysed using a number of different visualisation and clustering techniques. Clusters were identified which corresponded well with the error situations that had been introduced into the system. In order to confirm this correlation, artificial neural networks (ANNs) were trained in order to detect the different types of errors based on the two datasets gathered.

This chapter provides an introduction to ANNs in general in Section 6.1. Section 6.2 then describes how the different ANNs were applied to the datasets gathered from the experimental setup as described in Chapter 5. The results are discussed and the application of ANNs are evaluated in the context of the TDAQ system. As this is the very first attempt at detecting errors in the TDAQ system using ANNs a number of different network types have therefore been applied.

In Section 6.2.3 an ensemble of the ANNs is then developed in order to try to improve upon the results obtained using a single ANN. The previously trained ANNs were combined using a genetic algorithm in order to find the optimal combination.

Finally, in Section 6.3.1 a new method named Genetic Neural Mathematical Method (GNMM) is applied in order to identify variables that may not be needed when training the ANNs. After identifying the variables that are not needed or contain redundant information, they are removed and the ANNs are retrained using

only the remaining variables. The results are then compared to the results obtained previously using all the available variables and the results are discussed.

6.1 Introduction to Artificial Neural Networks (ANNs)

This section will provide an introduction to the concepts of ANNs. It then moves on to describe the four types of network used in this chapter, namely the multilayer perceptron (MLP) network, the time delay neural network (TDNN), the radial basis function network (RBFN) and the probabilistic neural network (PNN). It also includes the reasoning for choosing each of the ANNs described. Unless otherwise stated the MATLAB® implementation of the ANNs was used throughout the chapter.

6.1.1 ANN background

The idea behind ANNs is to mimic the way the biological brain operates. The basic computational unit in the nervous system is the nerve cell commonly referred to as a neuron. A neuron, in general terms, consists of:

- Dendrites (inputs)
- A cell body
- Axon (output)

A neuron receives input from other neurons (typically many thousands) through its dendrites. All the inputs are 'summed' and if this sum exceeds a certain threshold level, the neuron discharges an electrical pulse that travels down the axon, to the next neuron(s) and so on. A schematic view of a biological neuron can be seen in Figure 6.1.

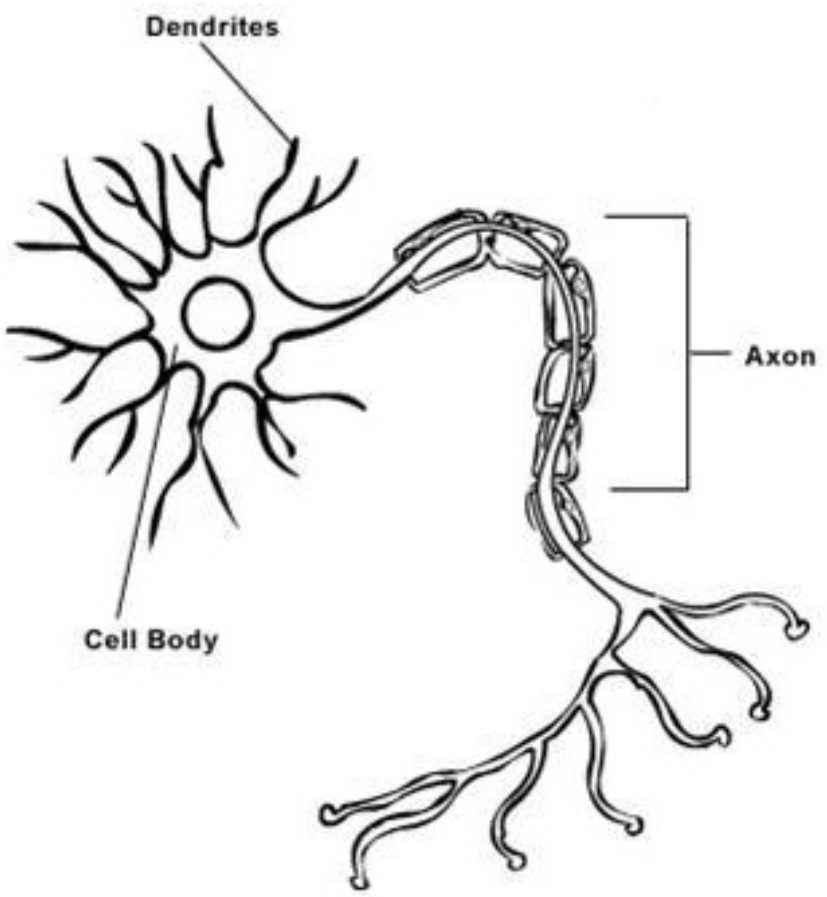


Figure 6.1: Schematic representation of a biological neuron. Adapted from (Carlson, 1992).

6.1.1.1 The artificial neuron

The *artificial* neuron was first proposed by McCulloch and Pitts in 1943 (Mcculloch and Pitts, 1943). It is based on the biological neuron, but uses a simplified mathematical model. As in the case of the biological neuron, the artificial neuron usually consists of 3 main components:

- X_m - m number of weighted inputs corresponding to the dendrites of a biological neuron. This is equivalent to the dendrites in the biological neuron.
- $f(x)$ - an activation function of the accumulated input. This simulates the cell body and the activation threshold in the biological neuron.

- y - the accumulated output y as shown in 6.1. This is equivalent to the electrical charge passing through the axon in the biological neuron.

$$y = f(\sum x_i w_i) \tag{6.1}$$

In addition to the sum of individual weights w_i it is common to use an arbitrary bias α such that:

$$y = f(\sum x_i w_i - \alpha) \tag{6.2}$$

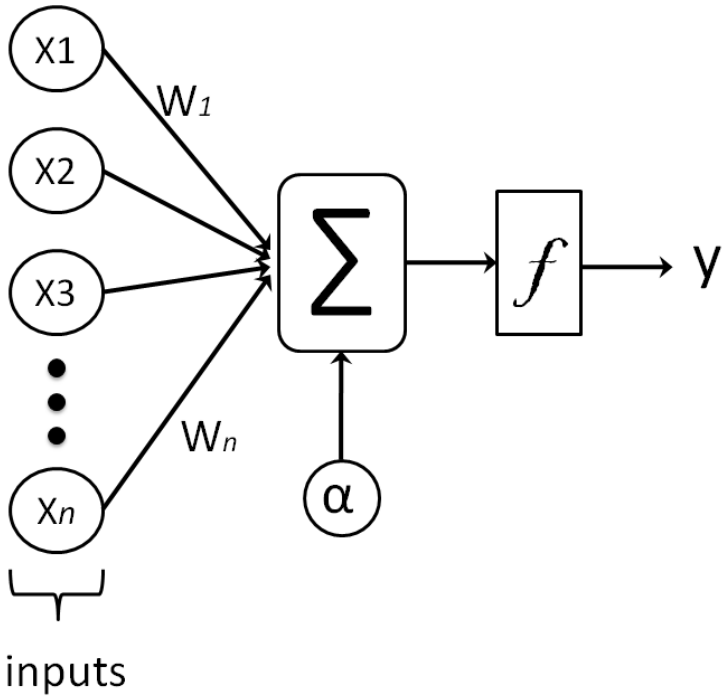


Figure 6.2: Schematic representation of the artificial neuron

A schematic representation of the artificial neuron is shown in Figure 6.2. The activation function can vary depending on implementation, but the sigmoid function:

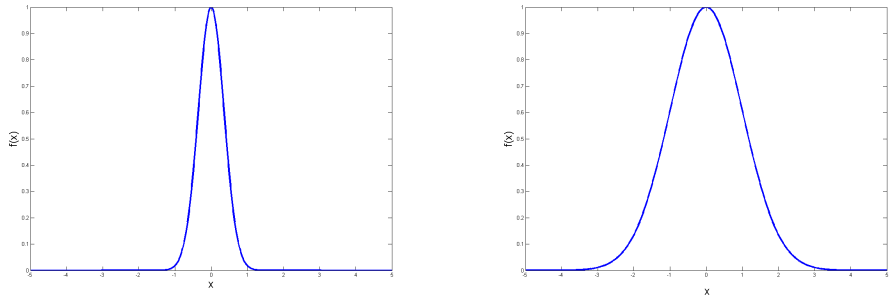
$$y(x) = \frac{1}{1 + e^{-x}} \tag{6.3}$$

where x is the input, is a popular choice due to the fact that it is continuously differentiable, something that is useful in the learning process as discussed in Sec-

tion 6.1.3.3. Another common choice of activation function is the radial basis function (RBF) which is a function where the value depends only on the distance from the origin, so that $f(x) = f(||x||)$. A common type of RBF is the Gaussian:

$$f(x) = \frac{1}{e^{-\beta x^2}}, \beta > 0 \tag{6.4}$$

where x is the input and β controls the width of the function as shown in Figure 6.3.



(a) $\beta = 4$ (b) $\beta = .5$

Figure 6.3: Shows the effect of changing β for function (6.4).

By interconnecting a number of artificial neurons, and by varying the activation functions, one can build variants of ANNs. There are no limits on the variants of ANNs that can be created in this way, but some common 'classes' exist and the ones used in this work will be described in the following section.

6.1.2 ANN variants

This section presents the different variants of ANNs used in this thesis work. The training of ANNs in general is then discussed before the specifics of training each of the presented ANN variants is described. Finally there is a comparison between the different variants of ANNs presented in this section.

6.1.2.1 Multi Layer Perceptron (MLP)

MLP is a variant of ANN where there is one or more hidden layers between the input and output layers. The MLP nodes include non-linearity in their output usually

represented by a logistic function. Normally each node in a layer is connected to every node in the next layer. This connection between the neurons is weighted and can be modified when training the MLP. Figure 6.4 shows a schematic overview of an example MLP consisting of an input layer, a hidden layer and an output layer containing a single output.

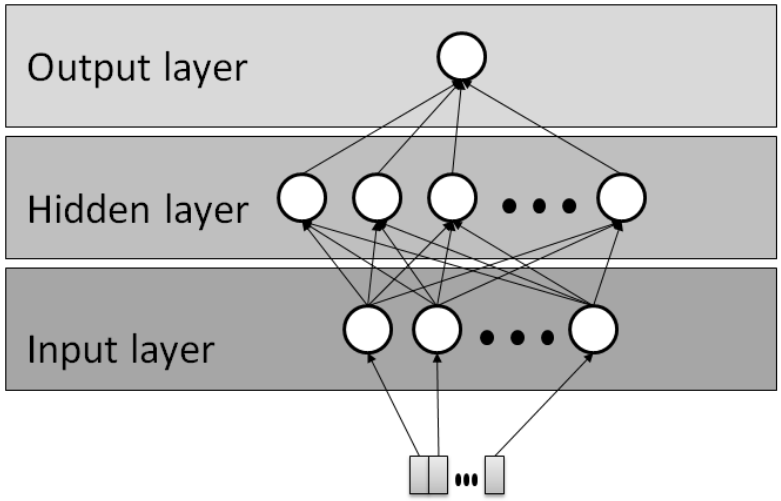


Figure 6.4: A simple MLP with one hidden layer and a single output

The MLP was chosen as it is the most commonly used ANN (Haykin, 1998) and provides the basis for many of the other techniques such as the TDNN described below. It has been shown that an MLP, provided that there are sufficient neurons in the hidden layer and sufficient data available, can approximate almost any continuous function (Valiant, 1988; Cybenko, 1989). Its applications are wide, ranging from atmospheric science (Gardner and Dorling, 1998), vision problems (Khotanzad and Chung, 1998) and classification problems (Ritschel et al., 1994) to name a few.

6.1.2.2 Radial Basis Function Network (RBFN)

RBFNs (Orr, 1996) are a variant of ANNs using a radial basis function as the activation function of the neurons. The topology of a RBFN is usually similar to that of the MLP, in that it consists of input nodes, one or more hidden layers of neurons and an output layer. The hidden neurons have a (usually) non-linear radial basis activation function while the output neuron is linear. The output of a RBFN can therefore be described as:

$$F(x) = \sum_{i=1}^{N} W_i \phi(||x - r_i||) \tag{6.5}$$

where N is the number of neurons in the hidden layer, ϕ is the radial basis function with a centre at r_i and W are the weights of the linear output neuron. A schematic view of a RBFN is shown in Figure 6.5.

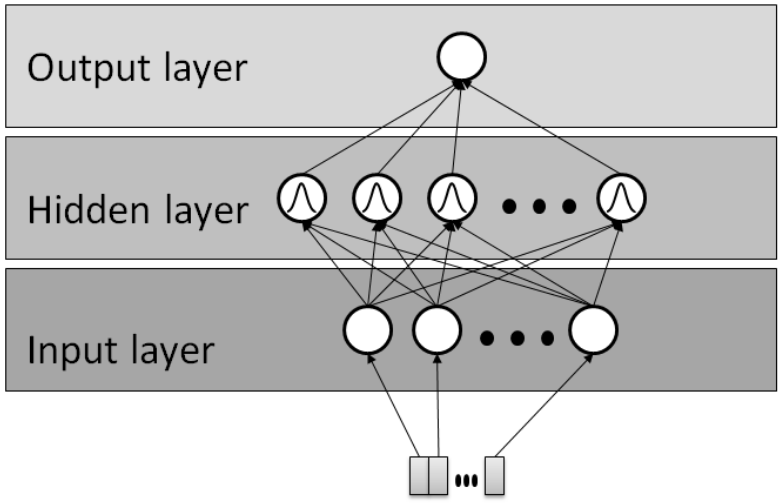


Figure 6.5: A simple RBFN with one hidden layer and one output. The neurons in the hidden layer all uses a RBF as the activation function.

The RBFN was chosen as it is capable of universal approximation (Park and Sandberg, 1991) and has been successfully applied to a wide range of problems such

as obtaining dispersion coefficients (de Almeida et al., 2000), forecasting (Mori and Awata, 2007) and importantly for the classification of faults (Leonard and Kramer, 1991; Meng et al., 2010).

6.1.2.3 Probabilistic Neural Network (PNN)

PNNs were first introduced by Specht (Specht, 1990) and further developed in (Specht, 1992). The PNN can be thought of as a type of normalised RBF network. It consists of three layers; namely the input layer, a hidden layer and the output layer. Let us consider each in turn:

Input layer: This layer consists of one unit per variable/dimension in the training data.

Hidden Layer: The hidden layer consists of n hidden units, where n is the number of training cases. The hidden units usually use some sort of probability density function, such as the Gaussian, as the activation function. The activation function of each unit is centred at a corresponding training case.

Output layer: The output layer consists of k nodes, where k corresponds to the number of classes in the training set. The weight between a hidden unit and an output unit is 1 if the training case corresponding to the hidden unit belongs to that output class, otherwise it is 0. Alternatively, prior known probabilities of class memberships may be used instead, in particular, in cases where the actual probabilities are known to differ from the relative representation in the training set.

PNNs have been shown to work very well in a number of classification related applications such as fault diagnosis (Danfeng et al., 2009; Liang et al., 2007; Yang et al., 2006), credit risk assessment (Huang and Tian, 2008), speech recognition (Ganchev et al., 2004) and satellite image classification (Tian et al., 2000).

6.1.2.4 Time-Delay Neural Network (TDNN)

TDNNs were first introduced by Waibel in 1987 for use in speech recognition (Waibel et al., 1989). TDNNs are able to learn temporal patterns in the dataset. This is achieved by utilising a number of delayed inputs, such that the same input patterns are presented several times to the network. Compared to the regular MLP (as described in Section 6.1.2.1) each unit in the first hidden layer is modified to be presented with n delayed input vectors in addition to the un-delayed input. If we consider an input vector to be of size $i = 3$ and a delay of $n = 2$ the hidden units will each have 9 weighted inputs. An example TDNN is shown in Figure 6.6.

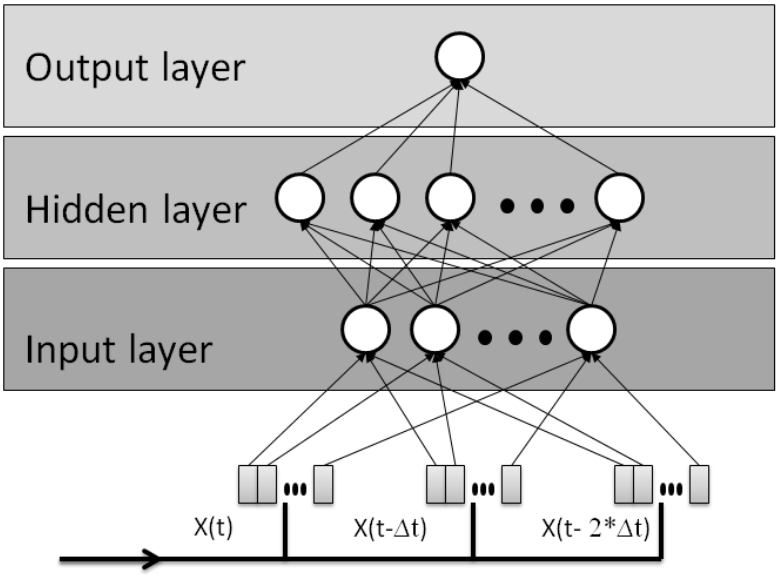


Figure 6.6: A TDNN with 2 time delays.

As each input vector is presented to the hidden units at $n+1$ different points in time with a delay of δt, this allows the TDNN to correlate any temporal changes in the input data. It is also possible to introduce delays at any subsequent hidden layers using the same approach, but in this work only the form of TDNN with delays at the input layer was used.

6.1.3 Training ANNs

This section describes the various training algorithms used to train ANNs. First the general algorithms and approaches to training ANNs are presented. It then presents the specific training approaches used for the different ANNs presented in the previous section.

6.1.3.1 Batch and incremental training

Two major types of training methods exists for ANNs; batch training and incremental training. The basic algorithm for batch training is as described in Algorithm 6.1 and the algorithm for incremental training is described in Algorithm 6.2. Incremental training presents a single training record to the ANN at a time and updates all weights using the single record. A new record is then presented and so on. The batch approach on the other hand presents a subset of the dataset at a time and uses the average error for all samples in the subset in order to update the weights. Both algorithms continue until a predefined criteria is reached. This criteria can be defined in a number of ways, though some common ones include a combination of the following:

(i) The number of training epochs have reached some defined limit

(ii) The average ANN error is below some defined tolerance limit

(iii) The gradient of the average ANN error is relatively small (i.e. below a defined limit)

6.1.3.2 On-line vs off-line training

In on-line training an input record is presented to the network and used for training and then discarded after it has been processed. Hence, each record can only be accessed once and only once. On-line training is always incremental.

In off-line training on the other hand all the training records can be consulted multiple times. This enables the use of more advanced training algorithms. Among

Algorithm 6.1 Batch training algorithm

1: Initialise the weights
2: **while** Predefined criterion is not reached **do**
3: Present a subset (may be the entire set) of the training data to the ANN and calculate *average* of the error
4: Update all the weights using the calculated error
5: Choose a new subset of the training data
6: **end while**

Algorithm 6.2 Incremental training algorithm

1: Initialise the weights
2: **while** Predefined criterion is not reached **do**
3: Present a *single* record of the training data to the ANN and calculate the error
4: Update all the weights using the calculated error
5: Choose a new input record
6: **end while**

other things one can perform multiple random initialisations of the network in order to for example overcome the possible problem of being stuck in a local minima. Throughout this chapter off-line training will be used unless stated otherwise.

6.1.3.3 Training algorithms

There exist several algorithms for updating the weights when training ANNs, however by far the most popular is the Back Propagation Algorithm (BPA) (Rumelhart et al., 1986) or some variant of it. The goal of the algorithm is to minimise the difference between the network output and the target output. This difference is referred to as the estimated error which is usually calculated as the Sum-Squared Error (SSE). Let o_i^p be the output of the ith neuron in the output layer for a given input pattern p and let t_i^p be the corresponding target output. The SSE E^p is then given by:

$$E^p = \frac{1}{2}\sum_i^n (t_i^p - o_i^p)^2 \tag{6.6}$$

The BPA makes two passes through the network. First the output at each node is calculated by feeding input data through the network in what is usually referred to as the *forward* pass. The output is then compared to a known correct/desired value and the error is calculated. A *backward* pass is then performed where the derivative of this error is propagated back through the network and all weights are updated so as to minimise the error.

It is important to note that the BPA does not guarantee that the network finds an optimal solution. It is possible for the algorithm to become stuck in a local minimum in the error surface. This is due to the fact that the algorithm uses a gradient descent method (Rumelhart et al., 1986).

A popular variation of the back-propagation is the Levenberg-Marquardt Algorithm (LMA) (Levenberg, 1944; Marquardt, 1963). This algorithm is generally faster than the BPA while achieving similar results. The LMA is therefore used throughout this work unless stated otherwise.

6.1.3.4 Training MLPs

MLPs are usually trained using the BPA or variants of it such as the LMA as described in the previous section. Both the batch and the incremental training methods can be used in order to train MLPs.

MLPs are prone to over-fitting (see Section 4.2.2.4) if the model is trained for a very large number of steps. However, determining what is the optimal number of training steps is non-trivial and often application specific. In order to avoid this problem a validation set (see Section 4.2.2.5) may be used in order to help determine when to stop the training. Such an approach is often referred to as *early-stopping*. The exact approach may vary, but a common method is to test the performance of the MLP on the validation set at each step of the training. If the performance on the validation set does not improve for a fixed number of generations the training is stopped and the current value is accepted to be the best result.

Alternatively, one can keep track of the epoch in which the validation error is the lowest and utilise those weights for the final network, though such an approach does not actually constitute early-stopping, it may reduce the chance of over-fitting. It is also not prone to natural fluctuations in the error value which may cause the previous approach to stop the training prematurely.

Note that in all cases the validation set *must* be different from the test set or one is effectively training using the test set which would lead to a overly optimistic measure of the generalisation performance.

6.1.3.5 Training RBFNs

The training of RBFNs differs from that of MLPs and is usually performed in two passes; first by adjusting the parameters of the radial basis functions and then by optimising the weights of the second layer. The first step is realised in an unsupervised manner (using only the input data), while the second step is performed using both input and output data. The computational cost of training RBFNs is quite low compared to that of the BPA. However, if the input dataset is large, the number of neurons might become very large leading to slow training and execution speed. For RBFNs it may be possible to find a 'perfect' interpolation of the training data using one neuron in the hidden layer per record in the input data. That such an interpolation is possible can be shown as follows:

Given a set of N distinct points $x_i, i = 1, 2, .., N$ and a corresponding point y_i, find a function F such that

$$F(x_i) = y_i, \ \forall i \in N$$

By creating a RBFN using N neurons in the hidden layer and choosing a centre for each neuron to correspond to a distinct input value, x_i, we have that:

$$g_{ij} = \phi(||x_j - x_i||) \tag{6.7}$$

By evaluating g, for all points and substituting for (6.6) we have that:

$$\begin{vmatrix} g_{ij} & \cdots & g_{iN} \\ \vdots & \ddots & \vdots \\ g_{Nj} & \cdots & g_{NN} \end{vmatrix} \begin{vmatrix} W_i \\ \vdots \\ W_N \end{vmatrix} = \begin{vmatrix} y_i \\ \vdots \\ y_N \end{vmatrix} \tag{6.8}$$

Equation (6.7) can be solved using linear algebra and thus the selection of weights constituting a perfect interpolation of the training samples is determined.

However, if the input data is noisy the RBFN may be over-fitting the problem (see Section 4.2.2.4), something that can lead to a much degraded generalisation performance. The problem can be illustrated as follows:

Let us consider a dataset generated using the function $f(x) = sin(x)$. Noise is then added to the dataset and the RBFN is 'solved' for the input using 6.8. The RBFN output will pass through every data point as shown in Figure 6.7, and one can see that the generalisation performance is not very good. The RBFN fits the data perfectly, but does not follow the underlying function very well. If we then create a RBFN using just 4 nodes in the hidden layer and train it using the same input data, then this leads to a much better generalisation performance as shown in Figure 6.8. In this case the output of the RBFN is more closely matched to that of the underlying function. This however, raises the question of how many nodes one should use in the hidden layer?

In order to overcome this possible problem an approach referred to as *constructive* RBFN is often used. This method starts out with a single neuron in the hidden layer and performs the training of the network. If the error is within a predefined tolerance, the RBFN is considered complete and no further action is taken. If the training error is larger than the predefined tolerance a single neuron is added to the hidden layer and the network is retrained. This process continues until the error is low enough or the number of neurons have reached a limit as described in Algorithm 6.3. This training method was used throughout the training of the RBFN in this chapter unless stated otherwise.

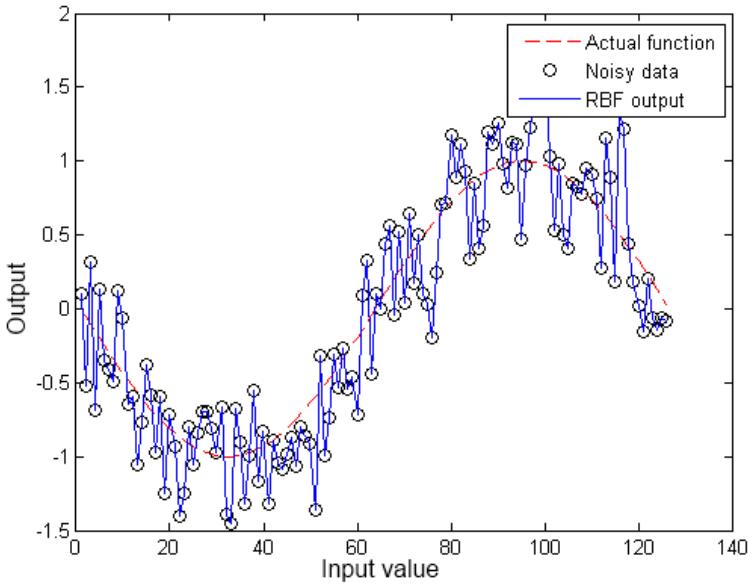


Figure 6.7: Exact interpolation using a RBF network. This leads to very bad generalisation performance.

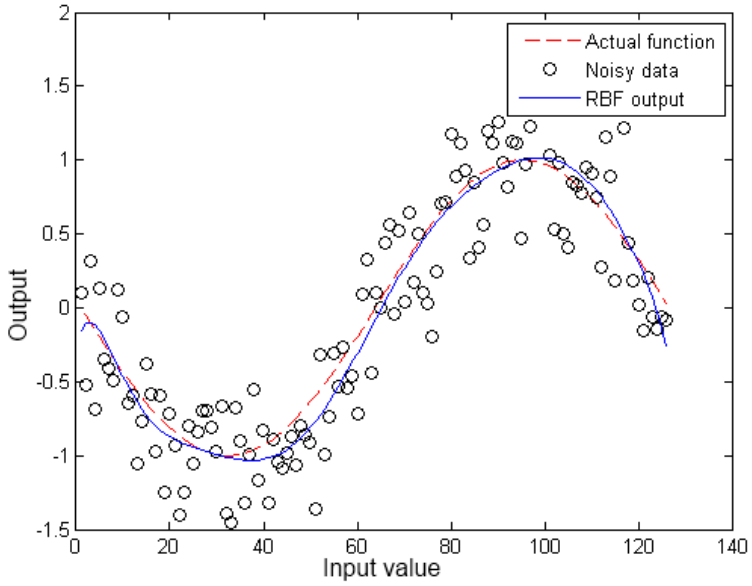


Figure 6.8: RBFN using just 4 nodes in the hidden layer.

Algorithm 6.3 Constructive RBFN

```
1: Create network with 1 neuron
2: Train the network
3: Error ← Calculate average error of network
4: while Error < error tolerance AND #neurons ≤ maxNeurons do
5:    Add a neuron to the hidden layer
6:    Retrain the network
7:    Error ← Calculate average error of network
8: end while
```

6.1.3.6 Training PNN

The weight connections in PNNs are either 1 or 0 depending on the corresponding class of the input data. The number of neurons corresponds to the number of training records and the centre of the hidden units are determined by the values of the individual training records. This means that the *only* parameter to be set during training is the width of the radial basis function. Hence, the training of the PNN can be quite fast, though determining the 'correct' width can potentially be rather time consuming. Incremental training is naturally not possible using PNNs (at least not using the most common algorithms) as all input cases are needed in order to determine the size and parameters of the network.

6.1.3.7 Training TDNNs

TDNNs normally use the BPA for training, modified to take into account the delayed inputs. As the delayed inputs are copies of previous ones, the weights corresponding to each time shift are constrained to be the same. In practice this is achieved by first determining the error gradient using the BPA for each time shifted input separately. The corresponding connections are then all modified by the *average* of the individual values. Hence the training is very similar to that of the MLP though somewhat more computationally expensive. Methods such as batch and incremental learning also apply to TDNNs.

6.1.4 Comparison of the presented ANNs

Each of the different variants of ANNs presented have some possible advantages and disadvantages. Table 6.1 provides a summary of the main points for each of the ANN variants.

6.2 Applying ANNs for error detection

As described in Chapter 5 data have been gathered from two different sources, and thus two separate datasets exists, namely the ERS dataset and the IS dataset. The following is a description of how the different ANNs were applied to detect errors using the two available datasets. Each of the datasets will be considered in turn:

6.2.1 Applying ANNs to the IS data

In this section the different ANNs will be applied to the IS data in order to detect and classify the three different error types. A measure to compare the performance of the different ANNs is first presented. The data are then prepared before the different ANNs are applied in turn.

6.2.1.1 Comparing ANN performances

In order to make it easier to compare the overall performance of the classifiers an overall performance measure was calculated. The performance measure was defined as the product of classification accuracy, Sensitivity, Specificity, PPV and NPV. Hence the performance of a ANN classifier for class i is given by:

$$Perf_i(ANN) = Accuracy_i * Specificity_i * Sensitivity_i * PPV_i * NPV_i;$$

Using this measure a classifier achieving good results in all categories will achieve a better performance rating compared to a classifier which achieves a higher accuracy in some categories, but lower in others. This is of importance in cases where the number of samples in one category are much larger than in the other as in the case of

Table 6.1: Advantages and disadvantages of the various ANNs

Variant	Advantages	Disadvantages
MLP	• A number of training algorithms exists with BPA being a popular choice. • Easy to implement with vast number of examples and literature available. • Good at ignoring irrelevant input.	• Suffers from getting stuck at local minima.
RBFN	• Training is fast. • It is possible to find solution which interpolates all the training samples and thus achieve zero error on the training set.	• Cannot easily ignore irrelevant input. • If training set is large, the number of nodes may become unmanageable which can also lead to slower execution speeds.
PNN	• Fast and easy training. • There is only a single parameter to be set by the user. • Can easily incorporate previously known probabilities.	• Unable to ignore irrelevant input, much like the RBFNs. • Suffers from the 'curse-of-dimensionality'. If the amount of input data is very large the execution time of the network might be very slow compared to that of a MLP or other types of ANNs.
TDNN	• Can take into account temporal patterns in the data and can therefor improve upon MLP in many cases.	• Training is slightly more complex and can be much more computationally expensive compared to MLPs.

the IS dataset. Indeed a classifier always reporting a negative result would achieve a classification accuracy of 95+%, but would clearly not do as good a job of classifying the errors!

Also, the execution time of a given ANN may be of importance if it is to be used in an online system. As execution time largely depends on implementation and which hardware it is running on the *relative* execution time was calculated. This was done by calculating the average execution time for each ANN and dividing by the lowest average execution time among all the different ANNs.

6.2.1.2 Data preparation

As one can recall from Section 5.3.1.2 the IS dataset was processed using a trend detection algorithm which was applied over intervals of 10 seconds. The corresponding target vector was created by noting the time intervals for each of the input data rows and if the system was in an error state for the majority of that interval (i.e. more than 5 of the 10 seconds) the target vector is set as positive. Hence the target vector is a binary vector containing 3 dimensions, one for each class of errors. The dataset is of relatively limited size containing 974 observations and in particular a limited number of positive cases containing just 33 positive samples divided over the three error types.

Noisy replicates of the input data were therefore added to the dataset. This has been shown to be beneficial to the generalisation error of the network (Koistinen and Holmstrom, 1991; An, 1996) in particular in cases of binary classification where the samples are eschewed (i.e. the majority of samples are of one class) (Lee, 2000) as is the case for the IS dataset. As the problem concerns dealing with binary classification the target vectors were left unchanged and simply replicated. This addition of noise is in general possible due to the fact that ANNs gives a similar output for very similar input values. A mathematical justification of the addition of noise is given in (Holmstrom and Koistinen, 1992). The process of adding noisy replicates is also referred to in the literature as *jitter*.

Addition of noise was found to be beneficial for the IS data and a comparison

of the results with and without noise is given in Section 6.2.1.7. Each input value was replicated twice with a signal-to-noise ratio (SNR) of 15, hence a total of 200% of noisy replicates was added to the dataset. A relatively high value of SNR was chosen to keep the input values similar for the same target (as described above the ANNs while produce similar output for very similar input values) while still producing different enough values so as to reinforce the ANN learning. However a detailed study of the effects of adjusting the SNR is left for future work. The dataset was then randomly split into a training, a validation set and a test set, using 50% of the observation for the training set, 25% for the validation set and the remaining 25% for the test set. Four different types of ANNs were trained to detect errors based on the IS data, namely MLP, TDNN, RBFN and PNN. The training procedure and the results of each of these will be considered in turn:

6.2.1.3 MLP results

As the datasets contains 25 dimensions an input layer of 25 neurons was used. Similarly, as there exists 3 different classes of errors to be detected, 3 neurons where used in the output layer. Unfortunately there does not exist any general method for determining the optimal size of the hidden layer. Some 'rules-of-thumb' exist, such as:

- the size of this hidden layer to be somewhere between the input layer size and the output layer size.
- The number of hidden nodes is calculate using a general rule of: (Number of inputs + outputs) * $\frac{2}{3}$.
- The hidden layer should never be more than twice as large as the input layer.

Though usually different approaches work well on different types of problems. Hence, a trial and error approach is usually required and was performed in this work as follows:

MLPs with a hidden layer ranging from 1 to 25 neurons were tested, and each MLP was trained 10 separate times and the average performance was calculated

using the validation set. The results of the these tests are shown in Figure 6.9 where a higher performance is better. As a result of these tests the model using 21 neurons in the hidden layer is the best. Hence a MLP of 25-21-3 was chosen as the final model.

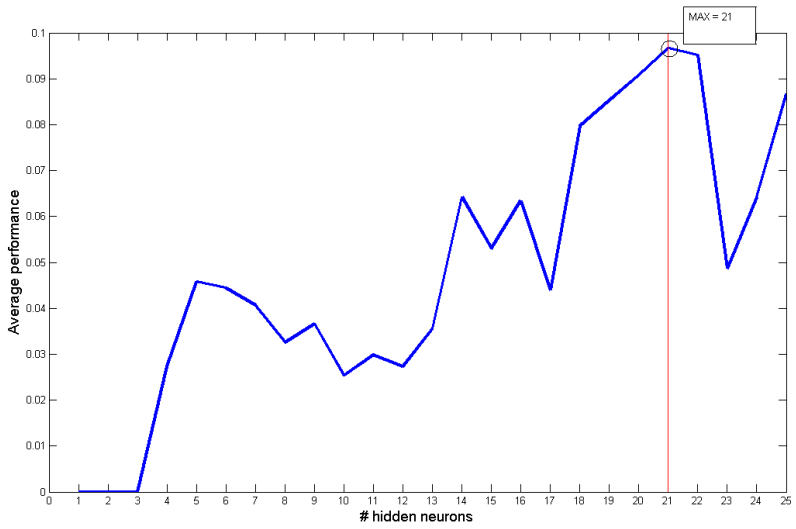


Figure 6.9: Performance of the MLP for different numbers of neurons in the hidden layer.

The MLP was then trained using the LMA algorithm on the validation dataset to avoid over-fitting as described in Section 6.1.3.4. Given that the output y of the MLP lies in the range of $[0, 1]$ a 'cut-off' point α was used such that any output greater than α is considered a positive classification while any value below this point was considered a negative classification. A different cut-off point was chosen for each of the outputs of the network, hence the outputs for class i are given by (6.9).

$$y_i = \begin{cases} 0 & \text{if } y_i < \alpha_i \\ 1 & \text{if } y_i >= \alpha_i \end{cases} \tag{6.9}$$

By varying α the classification performance of the MLP will also vary. The impact of varying the cut-off point for Type III error for the training data is shown in Figure 6.10. Naturally, one cannot optimise the cut-off point using the test data as this would mean that the network is trained/optimized using the test data

and the generalisation performance would therefore be biased. Also, optimising the cut-off point for the training data might lead to the problem of over-fitting as an optimised value for the training data might not correspond to the optimal value for the general case. In order to overcome this problem the *validation* set was used in order to calculate the optimal cut-off point α which was found to be at 0.26, 0.73 and 0.44 for the TypeI, TypeII and TypeIII errors respectively.

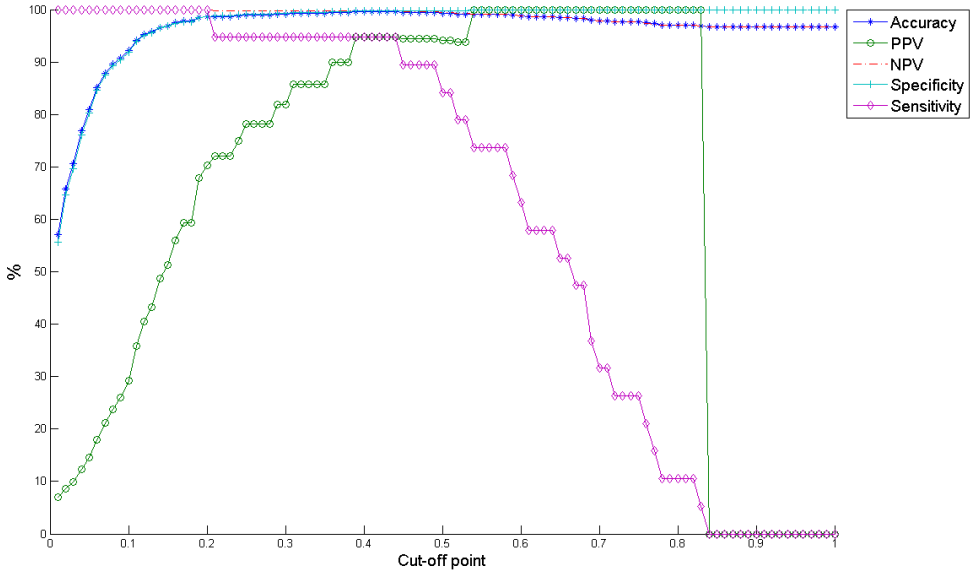


Figure 6.10: Performance of the MLP for Type III errors by varying the cut-off value for the output.

After determining the cut-off point the MLP performance was calculated using the test set and is summarised in Table 6.2. Overall the MLP achieves a very good generalisation performance, as estimated using the test set, achieving a classification accuracy of 99.18%, 98.63% and 99.04% for the TypeI, TypeII and TypeIII errors respectively. It is worth noting that for the TypeII error the sensitivity is at 50% meaning that only half of the positive cases are detected making it less useful for detecting such errors.

Table 6.3 shows the relative performance and execution time compared to the other ANNs. One can note that the MLP does not match the best ANNs, but does have the fastest execution time. This could be of importance depending on

the exact constraints for the final application of the classifier which could include a strict performance requirement in terms of speed.

Table 6.2: Results of the application of ANNs to the IS data

Network type	Error Type	dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
MLP	Type I	Training	99.38	0	9	99.35	100	89.16	100
		Test	99.18	2	4	99.41	95.83	92.00	99.71
	Type II	Training	99.79	3	0	100	88.46	100	99.79
		Test	98.63	10	0	100	50.00	100	98.61
	Type III	Training	99.93	0	1	99.93	100	97.56	100
		Test	99.04	0	7	99.02	100	68.18	100
TDNN	Type I	Training	98.77	9	9	99.35	87.84	87.84	99.35
		Test	99.32	3	3	99.71	93.75	95.74	99.56
	Type II	Training	98.91	13	3	99.79	50.00	81.25	99.10
		Test	98.49	11	0	100	45.00	100	98.47
	Type III	Training	99.66	3	2	99.86	92.50	94.87	99.79
		Test	99.86	0	1	99.86	100	93.75	100
RBFN	Type I	Training	98.70	0	19	98.63	100	79.57	100
		Test	98.63	2	8	98.83	95.83	85.19	99.70
	Type II	Training	99.73	0	4	99.72	100	86.67	100
		Test	99.18	0	1	99.86	100	95.24	100
	Type III	Training	98.91	1	15	98.95	97.50	72.22	99.93
		Test	98.36	1	11	98.46	93.33	56.00	99.86
PNN	Type I	Training	100	0	0	100	100	100	100
		Test	99.73	2	0	100	95.71	100	99.71
	Type II	Training	100	0	0	100	100	100	100
		Test	99.86	1	0	100	95.50	100	99.86
	Type III	Training	100	0	0	100	100	100	100
		Test	99.59	0	3	99.58	100	83.33	100

6.2.1.4 TDNN results

As for the MLP network, determining an optimal network topology was realised on a trial and error basis using the validation dataset. In addition to the number of neurons in the hidden layer one also needs to determine the ideal number of delayed inputs, and hence grid search was performed. TDNN topologies ranging from 1 to 3 delayed inputs and 1 to 30 neurons in the hidden layer was tested. Delayed inputs was presented with a time-delay of 5 seconds compared to the previous input. This

Table 6.3: Overall performance for the IS data with and without the addition of noise. Also shows the relative execution time of the ANNs.

Network type	Error Type	Performance (w/noisy replicates)	Performance (wo/replicates)	Execution time (w/noise)
MLP	TypeI	86.67	61.48	1
	TypeII	48.63	27.84	
	TypeIII	66.87	58.60	
TDNN	TypeI	88.49	64.91	1.3645
	TypeII	43.65	38.46	
	TypeIII	**93.49**	51.01	
RBF	TypeI	79.34	41.49	4.8376
	TypeII	**94.97**	49.97	
	TypeIII	50.54	37.12	
PNN	TypeI	**95.29**	50.54	36.12
	TypeII	94.74	43.49	
	TypeIII	82.64	58.60	

value was chosen as the errors should be detected within a reasonable amount of time. A large time delay would mean that the TDNN will react too slow, while very short delays might make it unable to detect any temporal patterns in the data. A detailed investigation of the effect of changing the time delay is left for future research. Each topology was tested 15 times and the average value was used as an estimate of the performance. The resulting TDNN had 2 delayed inputs and 14 hidden units.

As each input is presented three times to the network the training of the TDNN was markedly slower than for the MLP, increasing the training time by a factor of 4 compared to the 25-21-3 MLP network. The execution time of the network is also a factor of 1.36 slower compared to the MLP as can be seen from Table 6.3. The cutoff point was determined using the validation dataset and found to be optimal at 0.57, 0.40 and 0.40 for the TypeI, TypeII and TypeIII errors respectively. The overall results of the TDNN network for each error type are shown in Table 6.2.

One can observe that the TDNN achieves a classification accuracy ranging from 98.49% for the TypeII problem to 99.86% for the TypeIII problem. As for the MLP approach the TDNN does not achieve a very good result for the TypeII error

with a Sensitivity value of 45.00%. However, it does achieve excellent results for the TypeIII problems with all classifications correct save a single false positive. Consulting Table 6.3 one can note that the TDNN achieves a similar performance to the MLP for TypeI and TypeII errors while the TDNN achieves the best classification results for the TypeIII errors for all the ANNs.

6.2.1.5 RBFN results

In order to determine the number of RB neurons the constructive approach was used as described in Section 6.1.3.5. In this approach neurons are iteratively added to the network until a predefined goal is reached, in this case that is that the training error is below some threshold. Hence, there are two parameters to be defined while training the RBFN using this method; the *goal* which defines the error tolerance and the *width* of the radial basis function which determines how smooth the output is. In order to find the optimal choice for these values a grid of each combination of variables in the range $[10^{-8}, 10^{0}]$ for the goal and $[1, 15]$ for the width was performed. The performance was then evaluated using the validation set. Figure 6.11 shows the performance for each value of goal and width. As can be seen the optimal performance is achieved when $goal = 10^{-3.5}$ and $width = 8$. These values were then used for the final model. The RBFN was then trained resulting in a RBFN network consisting of 128 nodes. The cut-off point α were determined as in the case of MLP and TDNN by finding the optimal cut-off point for the validation set and was found to be at 0.11, 0.23 and 0.23 for the TypeI, TypeII and TypeIII errors respectively.

The results of the final RBFN are summarised in Table 6.2. It achieves classification accuracies of 98.63%. 99.18% and 99.36% for the TypeI, TypeII and TypeIII errors respectively. Looking at the overall performance from Table 6.3 one can see that the RBFN achieves the best performance of all ANNs for the TypeII error with a performance rating of 94.97 with only a single misclassified value. However, one should note that the execution time is a factor of 4.84 slower compared to the MLP.

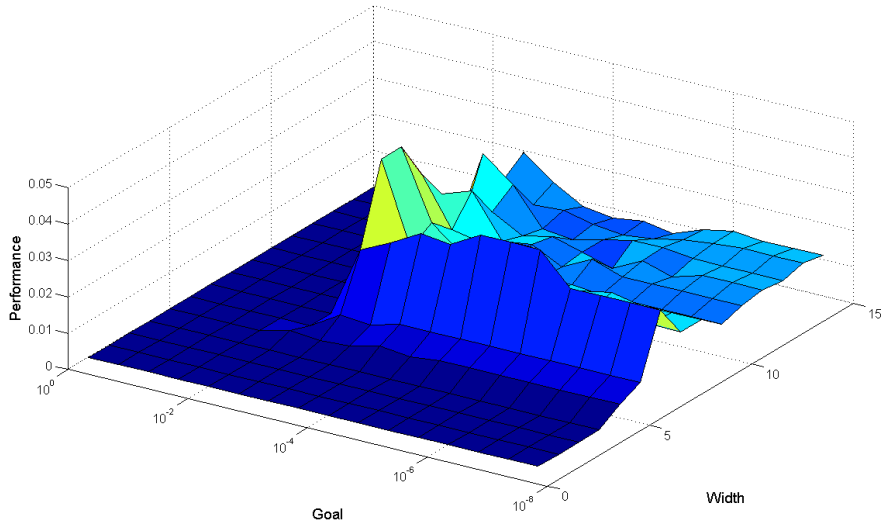


Figure 6.11: Performance of the RBFN for different combination of goal and width values.

6.2.1.6 PNN results

As mentioned in Section 6.1.3.6 the size of the PNN is determined solely by the size of the training dataset. In this case the training set consisted of 1462 values and hence the PNN consists of 1462 neurons in the hidden layer and 3 output nodes, one for each class. The only value that needs to be determined for the PNN is the width of the radial basis function σ. This was done through trial-and-error for values ranging from $\sigma = 0$ to $\sigma = 4$. The results are shown in Figure 6.12 and the optimal value was found to be 0.4. As the output of the PNN is 0 or 1 no cut-off point needs to be determined.

The PNN results are summarised in Table 6.2. One can note that the PNN achieves very good results in all categories achieving a classification accuracy for the test set of 99.73%, 99.86% and 99.59% for TypeI, TypeII and TypeIII error respectively and it achieves the best results of all the ANNs for TypeI errors as shown in Table 6.3. One should note that the PNN is significantly slower to execute being a factor of 36.12 slower than the MLP. This is due to the fact that the PNN includes a neuron per training sample. Indeed, as noisy replicates were added to the

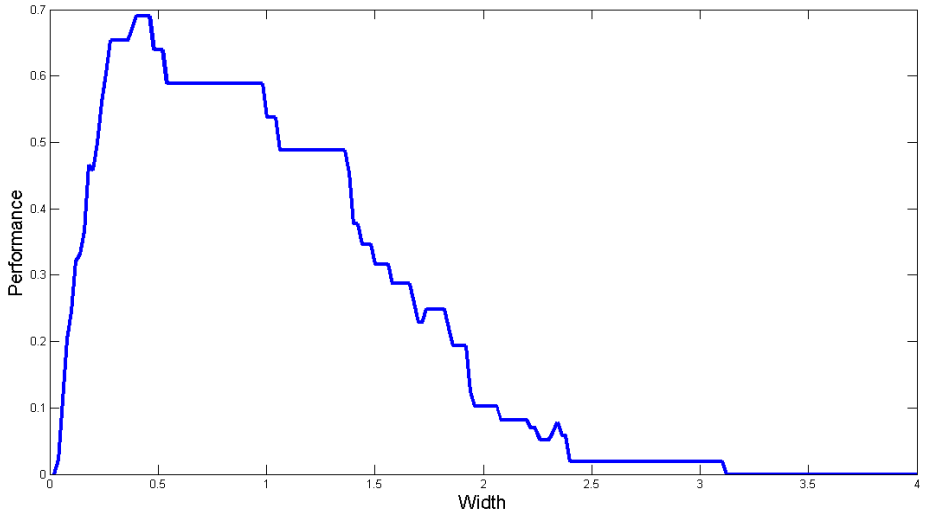


Figure 6.12: Performance of PNN for different values of σ.

data this further worsens the execution speed of the PNN.

6.2.1.7 Results without addition of noise

The results presented so far were obtained using the dataset including noisy replicates, which was found to provide better results. The results obtained without the addition of noisy replicates are shown in Table 6.4. One can observe that the overall results are very similar in terms of classification percentage, though if one investigates the other measures one can note a clear difference.

As an example let us consider the first PNN, trained using noisy replicates. This PNN has a classification accuracy of 99.59% for the TypeIII problem while the second PNN, trained using the original dataset, has a classification accuracy of 99.23%. The difference is seemingly insignificant. However, looking at the PPV the first PNN has a value of 83.33% while the second has only 71.43%, a clear difference.

Table 6.4: Results of the application of ANNs to the IS data without addition of noise

Network type	Error Type	Data set	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
MLP	Type I	Training	99.32	4	0	100	88.57	100	99.28
		Test	97.94	3	5	98.65	84.21	76.19	99.18
	Type II	Training	98.97	6	0	100	53.85	100	98.96
		Test	98.71	5	0	100	28.57	100	98.71
	Type III	Training	97.61	14	0	100	26.32	100	97.59
		Test	99.23	1	2	99.48	83.33	71.43	99.74
TDNN	Type I	Training	99.15	5	0	100	85.71	100	99.10
		Test	98.2	3	4	98.92	84.21	80.0	99.19
	Type II	Training	99.49	1	2	99.65	92.31	85.71	99.82
		Test	98.46	2	4	98.95	71.43	55.56	99.47
	Type III	Training	100	0	0	100	100	100	100
		Test	98.97	1	3	99.22	83.33	62.5	99.74
RBFN	Type I	Training	100	0	0	100	100	100	100
		Test	96.66	6	7	98.11	68.42	65.00	98.37
	Type II	Training	100	0	0	100	100	100	100
		Test	98.97	2	2	99.48	71.43	71.43	99.48
	Type III	Training	100	0	0	100	100	100	100
		Test	98.71	2	3	99.22	66.67	57.14	99.48
PNN	Type I	Training	100	0	0	100	100	100	100
		Test	96.91	3	9	97.57	84.21	64.00	99.18
	Type II	Training	100	0	0	100	100	100	100
		Test	98.71	2	3	99.21	71.43	62.5	99.48
	Type III	Training	100	0	0	100	100	100	100
		Test	99.23	1	2	99.48	83.33	71.43	99.74

6.2.2 Applying ANNs to the ERS data

Similarly, as for the IS data the ERS data was preprocessed prior to training the ANNs. The dataset consists of 4258 messages gathered from the system during its operation. As shown in Section 5.3.2.1 it is already known that only one of the types of errors can be detected based on the ERS dataset. Naturally the information contained within a single message is not sufficient to detect errors within the system. The data is therefore grouped in blocks of n messages. In order to determine a good value for n different MLPs were trained using different values for n ranging from 1 to 10. Using this approach the best value was found to be at $n = 5$. As each message includes 6 fields, this results in a dataset consisting of 851 inputs with 30 dimensions. One should note that as the system size increases the number of messages produced overall will also increase. The number of messages needed to determine any patterns in the system may therefore increase as well in future systems.

An initial test of adding noisy replicates to the ERS data was not successful. Table 6.5 shows the effect on the training of an MLP network with an addition of 0%, 25%, 50% and 100% of noisy replicates. As one can see, for the ERS data this had no positive effect and did in many cases lead to worse results. The dataset was therefore left in its original state for the following experiments. It was divided into a training, a validation and a test set using 50%, 25% and 25% of the messages respectively.

Table 6.5: The effect of the addition of noisy replicates to the ERS data.

Percentage of noisy replicates	0%	25%	50%	100%
MLP classification accuracy	81.73%	81.73%	79.37%	71.1%

As for the IS data four types of ANNs were trained, namely the MLP, TDNN, RBFN and PNN. Let us consider each in turn:

6.2.2.1 MLP

As for the IS data the optimal network size was found using a trial and error approach. The classification error was estimated using the validation dataset. MLPs with 1 to 30 hidden neurons were each tested 15 times in order to obtain a good indication of the average performance. The MLP with the highest average performance for the validation dataset was then selected. Figure 6.13 shows the average performance of each of the MLP sizes.

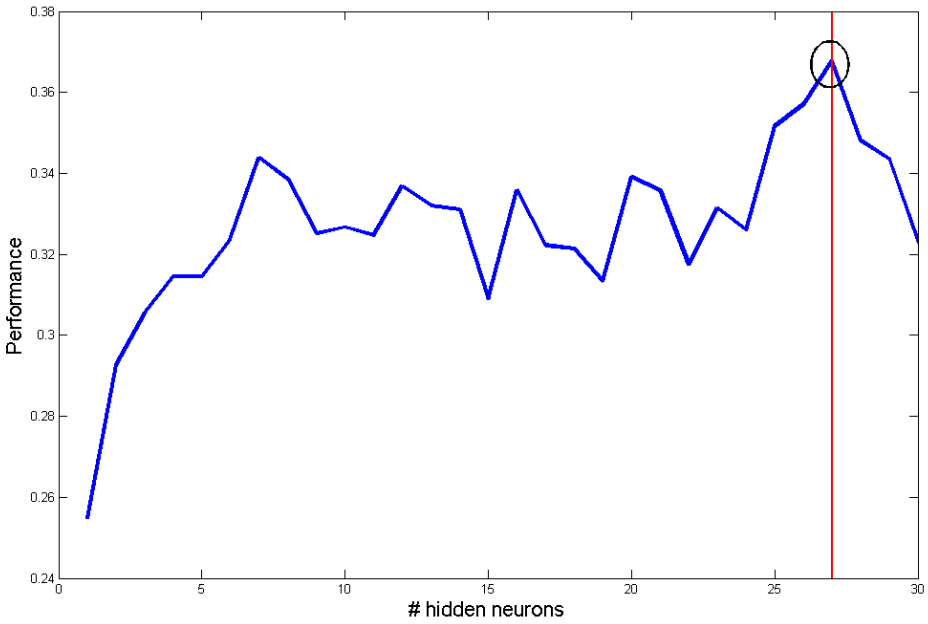


Figure 6.13: Performance of the MLP for different numbers of neurons in the hidden layer.

An 30-27-1 MLP was then trained and the optimal cut-off value α was determined using the validation set and was found at $\alpha = 0.44$. The MLP achieved a classification accuracy of 93.84% and 92.16% for the training and the test datasets respectively as is shown in Table 6.6. For the test set, the network achieved sensitivity and specificity of 92.71% and 91.82%, together with PPV and NPV of 95.42% and 87.25%.

Table 6.7 shows the comparison with the other ANNs. The overall performance of the MLP is worse than that of the TDNN and RBFN, but achieves a

better results than the PNN.

6.2.2.2 TDNN results

As for the MLP the optimal size was determined by trial and error using the validation set in order to estimate the performance. A grid search was performed as for the IS data as described in Section 6.2.1.4. The results are shown in Figure 6.14 and one can observe that the optimal performance is achieved using 2 delayed inputs and 17 hidden neurons.

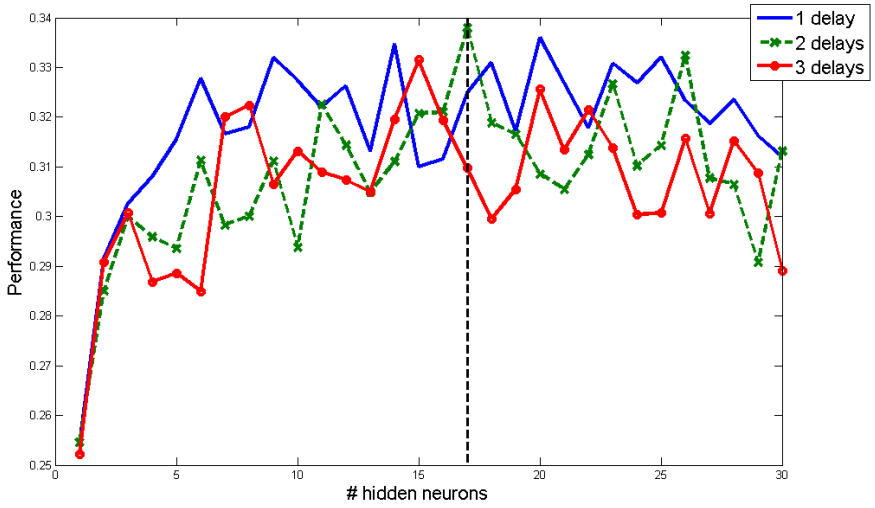


Figure 6.14: Performance of the TDNN for different combinations of delays and number of neurons in the hidden layer.

The TDNN using 2 delayed inputs and 17 neurons in the hidden layer was then trained and the optimal cut-off value was determined to be at $\alpha = 0.47$. The results are summarised in Table 6.6.

The TDNN achieved an overall better performance for the test set compared to the MLP with a classification accuracy of 94.51% though the specificity was slightly lower with a value of 88.54% as opposed to 92.71% for the MLP. One can see from Table 6.7 that the TDNN achieves a better overall performance using the measure introduced in Section 6.2.1.1 with a value of 74.08 compared to 65.32 as achieved by the MLP.

6.2.2.3 RBFN results

As noted in Section 6.2.1.5 we need to determine the width, σ, of the radial basis function and the goal error value used to stop the training. As before this was determined using a grid-search of a combination of σ and goal error values and the result of this search is shown in Figure 6.15. Using these values a RBFN was trained using the constructive algorithm.

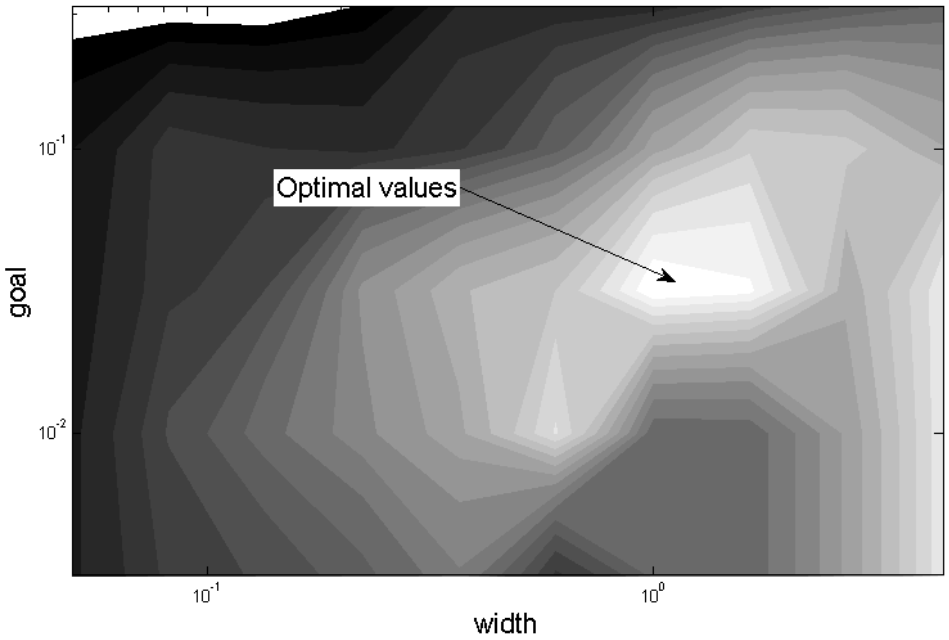


Figure 6.15: Performance of the RBFN for different combinations of width and goal values.

The final RBFN consisted of 96 neurons in the hidden layer with a cut-off value of $\alpha = 0.27$. The RBFN achieved the results as summarise in Table 6.6. The overall classification accuracy was 94.51%, identical to that of the TDNN. It does however achieve slightly better overall performance with a value of 74.66 compared to the 74.08 achieved by the TDNN as seen in Table 6.6. The execution time of the RBFN is also nearly identical to that of the TDNN.

6.2.2.4 PNN results

For PNN the only value to determine is the width, σ, of the radial basis function. As for the IS data σ was determined using a brute force search testing values within the range $\sigma = e^{-15}$ to $\sigma = e^{4.5}$. The result of this search is shown in Figure 6.16 and the optimal value was found to be $\sigma = e^{-4.5}$. The PNN uses one node per training value, hence the final PNN consisted of 341 neurons in the hidden layer. The results are summarised in Table 6.6 where one can see that the results are significantly worse than for the other approaches achieving a classification accuracy for the test set of 82.75%. There can be a number of reasons for this, such as a bad choice of σ (despite the testing using the validation data). Alternatively, the PNN could suffer from over-fitting and is known to be unable to ignore irrelevant or duplicated inputs, even more so due to the fact that it uses *all* the training inputs to create its hidden layer.

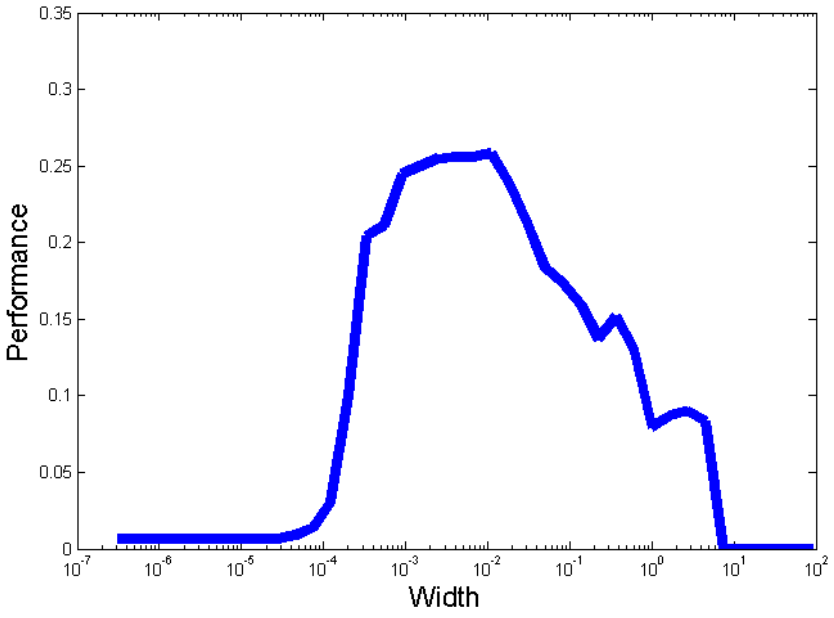


Figure 6.16: Performance of the PNN for different values of width.

Table 6.6: Results of the application of ANNs to the ERS data

Network type	Error Type	dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
MLP	Type I	Training	98.53	1	4	97.20	99.49	98.01	99.29
		Test	92.16	13	7	92.71	91.82	95.42	87.25
TDNN	Type I	Training	95.01	2	15	89.51	98.99	92.89	98.46
		Test	94.51	3	11	88.54	98.11	93.41	96.59
RBFN	Type I	Training	97.65	3	5	96.27	98.55	97.61	97.73
		Test	94.51	6	8	92.23	96.05	94.81	94.06
PNN	Type I	Training	96.77	6	5	96.50	96.97	97.46	95.83
		Test	82.75	26	18	81.25	83.65	88.08	75.00

Table 6.7: Overall performance and relative execution time of the ANNs for the ERS data

Network type	Error Type	Classifier performance	Relative execution time
MLP	TypeI	65.32	1.1367
TDNN	TypeI	74.08	**1**
RBF	TypeI	**74.66**	1.0015
PNN	TypeI	37.15	1.8714

6.2.3 Ensemble approach

In order to try to improve upon the results achieved using single individual ANN an ensemble approach will be explored. It has been shown that the error can be reduced by combining a number of networks in an ensemble (Webb and Zheng, 2004). A number of different methods for constructing such ensembles exists (Torres-Sospedra et al., 2005), though in this work the 'weighted average' method was used as it was found to give good results while keeping the approach relatively simple. The weighted average method works by calculating the average of the weighted output of all the networks in the ensemble. Such an ensemble is shown in Figure 6.17.

In order to determine the weights in the best possible way one must do a multivariate optimisation. A brute force search may not be ideal as even if only n different values are tested for each weight the number of combinations that must be tested would be n^4, which leads to the curse-of-dimensionality and is impractical or

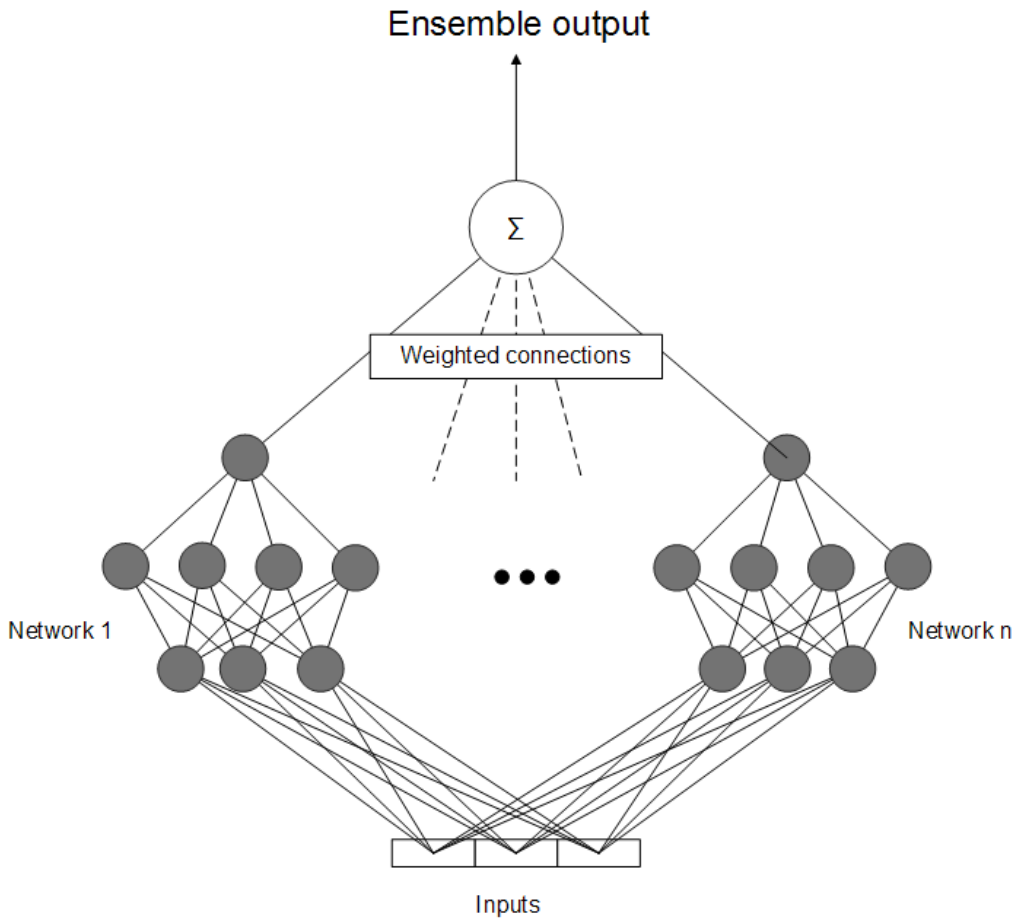


Figure 6.17: Ensemble of ANNs using weighted sum.

even infeasible for a large n. A number of techniques exists to solve such problems, such as simulated annealing (Kirkpatrick et al., 1983), Powell's method (Powell, 1964) and various hill climbing algorithms to name a few. However, the optimisation was effected using a Genetic Algorithm (GA) approach as GAs have been shown to be very effective for such problems and has also been previously used in order to combine ANNs in an ensemble (Osowski et al., 2009).

The following is a short introduction to Genetic algorithms in general before the technique is applied to the determine the optimal ensemble of the previously developed ANNs.

6.2.3.1 Introduction to Genetic Algorithms

GAs were first developed by John Holland (Holland, 1962, 1992) in 1962 and have since become popular in a variety of fields. GAs are an optimisation technique based on the natural selection and genetics as we know it from nature (Goldberg, 1989). GAs work by first creating a number of candidate solutions to a given problem called a *population*. Each candidate solution is evaluated using a 'fitness function' which is a measure of how well the particular solution solves the problem at hand. New generations of the populations are then created using so called evolutionary operators, also inspired by nature. The most common evolutionary operators include *crossover*, *mutation* and *reproduction*. A brief description of these operators follows:

Crossover : Two individuals from the populations are selected as *parents*. These are then recombined resulting in *children* individuals which can be used in the new generation. There exists a number of ways to perform crossover, often dependent on the representation used.

Mutation : A part of the individual is randomly changed leading to a slightly different individual. The new individual can then be used in the new generation.

Reproduction : An individual is copied into the new generation without any changes.

The basic principle behind GAs is that a fit chromosome, or parts of it, has a greater chance of surviving within a population and take part in future generations. The population should therefore converge towards an optimal or near-optimal solution. Having many candidate solutions together with random changes (mutation) avoids the problems of getting 'stuck' at local minima/maxima. The basic algorithm for GAs is visualised using a flow chart in Figure 6.18. GAs are particularly useful when the solution space is any combination of the following:

Very large : This makes exhaustive searches unfeasible. This includes for example NP-hard problems (for an in-depth explanation of NP-hard problems the reader is referred to Garey and Johnson (1979)).

Contains local maxima/minima : Many local hills/valleys in the solution landscape can cause a number of conventional approaches to get stuck.

Discrete : GAs do not need a differentiable solution space to work. In this sense it differs from the gradient approaches.

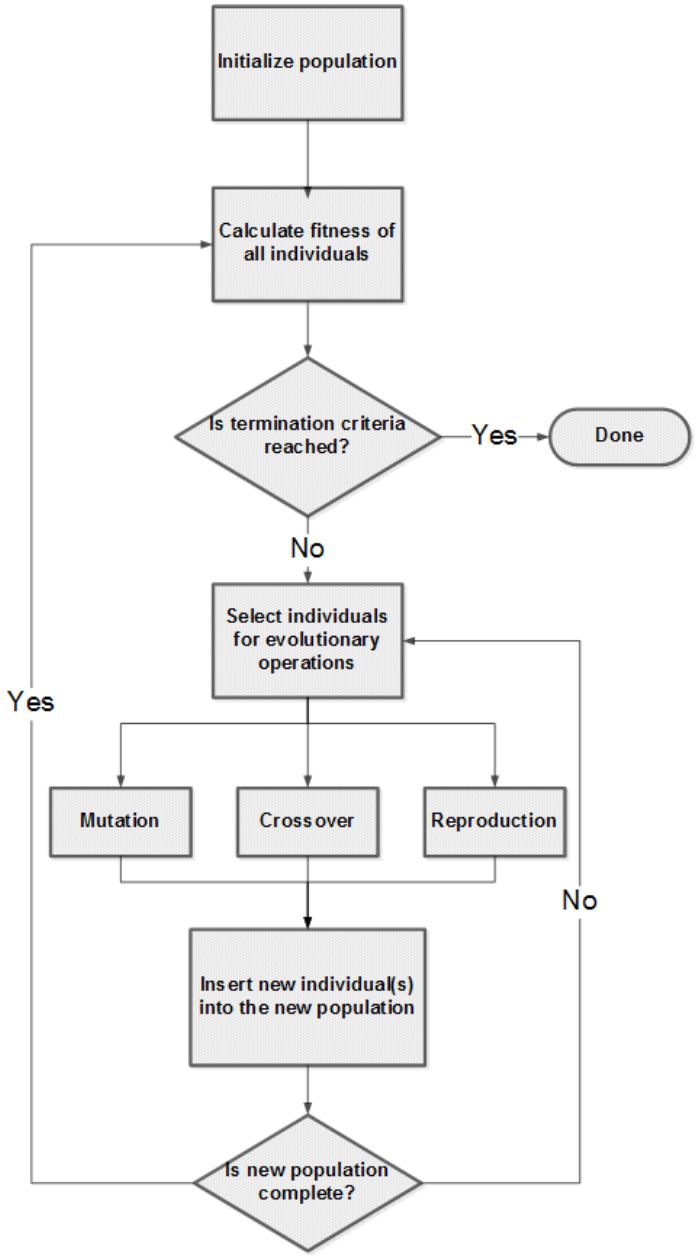


Figure 6.18: Flowchart of a conventional GA

6.2.3.2 Choosing GA parameters

Perhaps a natural question is how to choose the different parameters of a GA; the population size, the rate of mutation and crossover and any other parameters that may be available (different implementations offer a different set of configurable parameters). The fact that GAs are usually applied to problems where the fitness landscape is unknown makes the problem harder as the fitness landscape determined by the parameter tuning has the same unknown properties (Lobo et al., 2007). As the GA parameters are not independent choosing good parameters can be a very time-consuming process and, if one are not careful, lead to combinatorial explosion as the possible number of combinations of parameters becomes to large. There is also no guarantee that the optimal parameters have been found even if a significant effort has been made in the search. However it has been shown that it is often not necessary to find the exact optimal values, but that a range of values yields a sufficiently good performance (Lobo et al., 2007). Different implementations of GAs therefore often come with standard values that have been shown to work well for a number of problems.

On the other hand, the No Free Lunch Theorem (NFLT) (Wolpert and Macready, 1997) states that no algorithm is more efficient on average than any other for all possible problems. One must therefore be careful to accept such default values for GAs. Still, if optimal performance is not of the essence one can usually use values found in the literature with good results.

The values chosen throughout the thesis are all based on similar problems found in the literature, unless otherwise stated. Typical ranges of parameters, as found in (Prieto and Prez, 2008) and (Digalakis and Margaritis, 2002), are shown in Table 6.8. The choice of GA parameters may or may not be optimal, however a detailed study of the effects of parameter tuning for the problems in this thesis is left for future research.

Table 6.8: Typical GA parameters as found in the literature.

Parameter	Value
Population size	$[50 - 400]$
Crossover probability	$[0.6 - 0.95]$
Mutation probability	$[0.001 - 0.10]$

6.2.3.3 Applying GAs to determine ensemble weights

One of the important questions is how to represent/encode the different solutions as individuals in the population. It has been decided to use a 'straightforward' approach in which each individual consists of n bits and the weight of each network is represented using m bits such that $n = 4 * m$. In order to decode the individual one simply converts the corresponding bits for each into a decimal value. This was done such that the value lies in the range of $[0, 3]$ which is achieved by setting $val = decimaleValue * \frac{3}{2^m - 1}$. The process of encoding an individual is visualised in Figure 6.19. The fitness value was calculated as follows: First the weighted output of the 4 ANNs was calculated based on the individual encoding. The cut-off point was then determined as for the other ANNs using the validation set. The fitness value was then set to $Fit = 1 - Perf$ where Perf is the performance as calculated in Section 6.2.1.1.

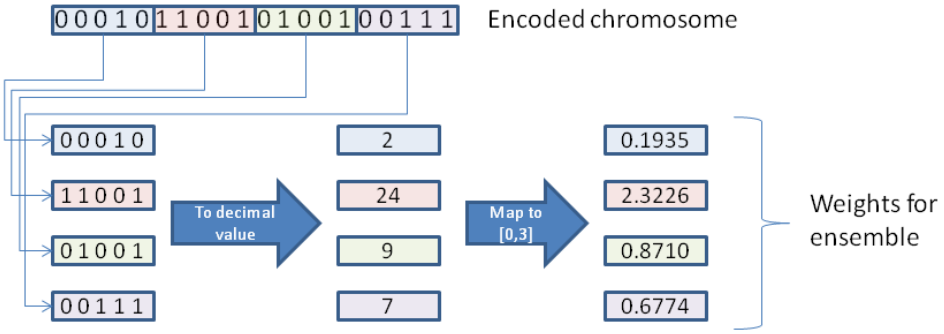


Figure 6.19: Encoding and translation of GA

The ensemble approach was used both for the ERS and the IS data. 6 bits were used for each weight such that $2^6 = 64$ different values for each weight was possible (and hence 2^{24} different combinations exists). For each of the error types the GA was run using the parameters listed in Table 6.9.

Table 6.9: GA parameters used for the ensemble approach.

Parameter	Value
Population size	50
Generations	500
Crossover probability	0.7
Mutation probability	0.02

6.2.3.4 Ensemble results

The final ensemble for the IS data was unable to improve significantly on the performance of the best single ANN (the PNN) achieving a classification accuracy of 99.73%, 99.86% and 99.32% for the TypeI, TypeII and TypeIII errors respectively.

For the ERS data the ensemble provides a small improvement over the best ANN (the RBFN) with one less FN and otherwise identical results. Again as the results using single ANNs were so good it makes it difficult for an ensemble approach to make significant improvements. Table 6.10 shows the results of the ensemble approach for the ERS data.

6.3 Utilising the Genetic Neural Mathematical Method

In this section a method called Genetic Neural Mathematical Method (GNMM) was used to select variables for training neural networks. The approach is compared to the results achieved using all available variables. Section 6.3.1 provides a brief introduction to GNMM before it is applied to the data from the TDAQ system.

Table 6.10: Results of the ensemble approach for the ERS data

Network type	Error Type	dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
Ensemble for IS data	Type I	Training	100	0	0	100	100	100	100
		Test	99.73	2	0	100	95.83	100	99.71
	Type II	Training	100	0	0	100	100	100	100
		Test	99.86	0	1	99.86	100	95.24	100
	Type III	Training	99.86	0	2	99.86	100	95.24	100
		Test	99.32	1	4	99.44	93.33	77.78	99.86
Ensemble for ERS data	Type I	Training	97.65	3	5	96.27	98.55	97.61	97.73
		Test	94.90	5	8	92.23	96.71	94.84	95.00

6.3.1 Introduction to Genetic Neural Mathematical Method

Genetic Neural Mathematical Method (GNMM)(Yang et al., 2008b,a) is a data driven method which optimises the input selection for neural network. This both simplifies the network structure and accelerates the training procedure.

GNMM uses GAs as a variable selection tool in order to determine the optimal set of variables to be used as input for an MLP network. Individuals in the GA are encoded using bit strings, where a 1 indicates that a variable is used while a 0 indicates that it is not used. The fitness of each individual is found by training MLPs networks based on the individuals using only the variables as given by the individual's encoding. The performance of the MLP is then used as the fitness value of the individual.

The initial step of the GNMM therefore follows the regular GA model:

1. Initialise a population of randomly chosen individuals.
2. For each individual, train a MLP network using the input determined by the individual. The result of the trained network constitutes the fitness of that individual.
3. Apply evolutionary operators and create a new generation.
4. This is repeated for n generations and the best individual for each generation is stored.

After the GA has completed, the concept of *appearance percentage* is used to determine the optimal set of input variables. The appearance percentage of a variable is defined as the percentage of winning individuals within a given run of the GA which contains that particular variable. Depending on the population size and number of generations used in the GA, it might be necessary to re-run the process until a clear distinction in appearance percentages becomes evident. An MLP can then be trained using just the variable set identified through the use of GA. This allows for faster training and can also lead to better performance compared to using all the available variables.

GNMM also incorporates the possibility of rule extraction through a mathematical model. It is presumed that the hyperbolic tangent (tanh) function (6.10) is being used as an activation function for the hidden neurons in the MLP. It has been shown in (Tsaih and Lin, 2004) that the tanh function can be approximated as shown in (6.11) where $\beta_1 = 1.0020101308531$, $\beta_2 = -0.251006075157012$ and $k = 1.99607103795966$. The output of the hidden neurons can therefore be approximated using this function and the corresponding rules can be extracted.

$$f(t) = \frac{2}{1 + e^{-2t}} - 1 \tag{6.10}$$

$$g(t) = \begin{cases} 1 & t >= k \\ \beta_1 t + \beta_2 t^2 & 0 <= t <= k \\ \beta_1 t - \beta_2 t^2 & -k <= t <= 0 \\ -1 & t <= -k \end{cases} \tag{6.11}$$

6.3.2 Applying GNMM to the TDAQ data

The GNMM was applied to both the ERS and IS datasets. Let us consider each one in turn:

6.3.2.1 IS data

The IS data contains 25 variables, hence an encoding using 25 binary values was used. The GA was run for 1000 generations using 200 individuals in the population. The mutation rate was 80%.

Appearance percentage: The appearance percentage of each of the 25 variables were then calculated and are shown in Figure 6.20. All variables with a percentage above 80% were selected leaving a dataset with 14 variables.

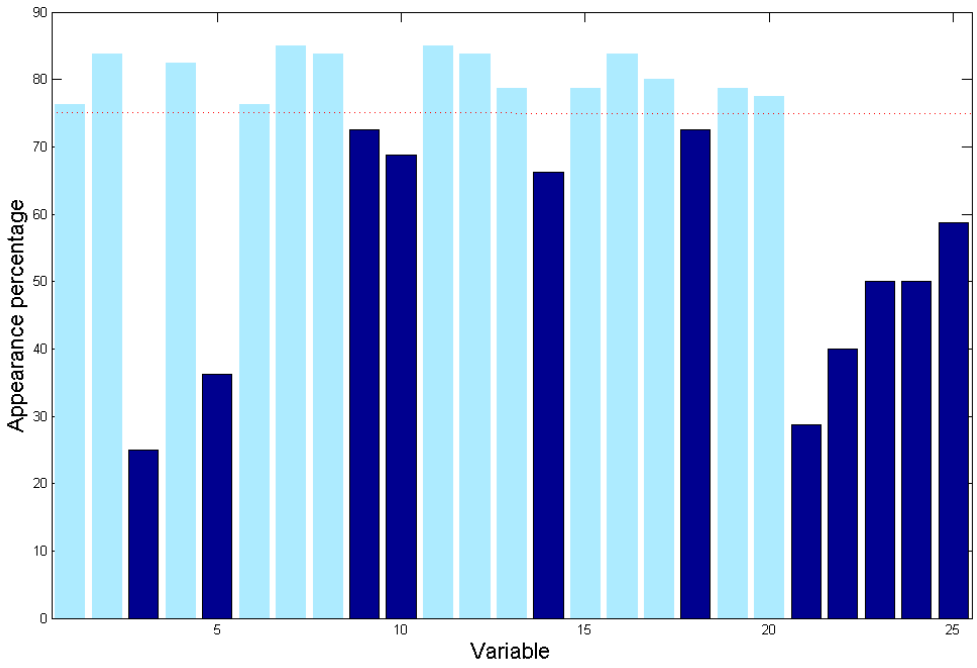


Figure 6.20: Appearance percentage for the IS data. All variables with an appearance percentage of above 80% were selected.

Results: Using only the 14 variables identified using the appearance percentage the four ANNs were re-trained. The input layers were updated to account for the reduction in variables while all other topologies were kept identical. The results of the new networks are summarised in Table 6.11. The overall performance, using the performance measure from Section 6.2.1.1 before and after dimensionality reduction,

is shown in Table 6.12. The MLP, TDNN and RBFN all achieve better results for TypeI and TypeII errors, while worse results were achieved for the TypeIII errors. The PNN achieves identical performance except for a reduction for the TypeII errors mainly due to a reduction of Specificity from 95.50% to 85.50%.

Table 6.11: Results of the application of ANNs to the IS data after variable selection using GNMM

Network type	Error Type	Dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
MLP	Type I	Training	99.52	3	4	99.71	95.95	94.69	99.78
		Test	98.90	7	1	99.85	85.42	97.62	98.98
	Type II	Training	99.04	4	10	99.30	84.62	68.75	99.72
		Test	98.22	4	9	98.73	80.00	64.00	99.43
	Type III	Training	98.97	12	3	99.79	70.00	90.32	99.16
		Test	98.77	4	5	99.30	73.33	68.75	99.44
TDNN	Type I	Training	99.79	1	2	99.86	98.65	97.33	99.93
		Test	99.45	3	1	98.85	93.75	97.83	99.56
	Type II	Training	99.18	10	2	99.86	61.54	88.89	99.31
		Test	98.90	7	1	99.86	65.00	92.86	99.02
	Type III	Training	98.70	15	4	99.72	62.50	86.21	98.95
		Test	97.67	7	10	98.60	53.33	44.44	99.02
RBFN	Type I	Training	99.52	1	6	99.57	98.65	92.41	99.93
		Test	98.77	4	5	99.27	91.67	89.80	99.41
	Type II	Training	99.86	0	2	99.86	100	92.86	100
		Test	99.86	0	1	99.86	100	95.24	100
	Type III	Training	99.59	3	3	99.79	92.50	92.50	99.79
		Test	98.49	2	9	98.74	86.67	59.09	99.72
PNN	Type I	Training	100	0	0	100	100	100	100
		Test	99.73	2	0	100	95.71	100	99.71
	Type II	Training	100	0	0	100	100	100	100
		Test	99.59	3	0	100	85.00	100	99.58
	Type III	Training	100	0	0	100	100	100	100
		Test	99.59	0	3	99.58	100	83.33	100

Discussion The improvement in performance after reducing the number of variables indicates that some variables in the dataset may be redundant. However, the improvement was limited to the TypeI and TypeII errors while decreasing the performance for the TypeIII errors. The variable selection may have been dominated

Table 6.12: Comparison of ANN performance before and after variable selection using GNMM

Network type	Error Type	Before GNMM	After GNMM
MLP	TypeI	**86.67**	81.51
	TypeII	48.63	**49.37**
	TypeIII	**66.87**	49.17
TDNN	TypeI	88.49	**90.68**
	TypeII	43.65	**59.03**
	TypeIII	**93.49**	22.60
RBF	TypeI	79.34	**80.23**
	TypeII	**94.97**	**94.97**
	TypeIII	**50.54**	49.66
PNN	TypeI	**95.29**	**95.29**
	TypeII	**94.74**	84.29
	TypeIII	**82.64**	**82.64**

by the first two error types, leading to a 'better' overall performance at the cost of decreasing the performance for the TypeIII error. Still the improvements for the first two types of errors show that improvements of the overall performance can be made by using a sub-set of the available parameters.

6.3.2.2 ERS data

There are 30 available parameters in the ERS data, hence each individual consists of 30 binary fields each representing one of the available parameters. The GA stage of GNMM was run for 1000 generations using 200 individuals in the population. The mutation rate was 80%. The exact GA parameters are of less importance as we are not trying to converge on an optimal solution, but rather evaluating the impact of the different variable combinations.

Appearance percentage: The appearance percentage of the 30 available parameters were then calculated and are shown in Figure 6.21. While there is not a 100% clear distinction of variables some are definitely more used than others. All variables with an appearance percentage of more than 60% were kept, while the other 20 variables

were discarded and a new training set was created.

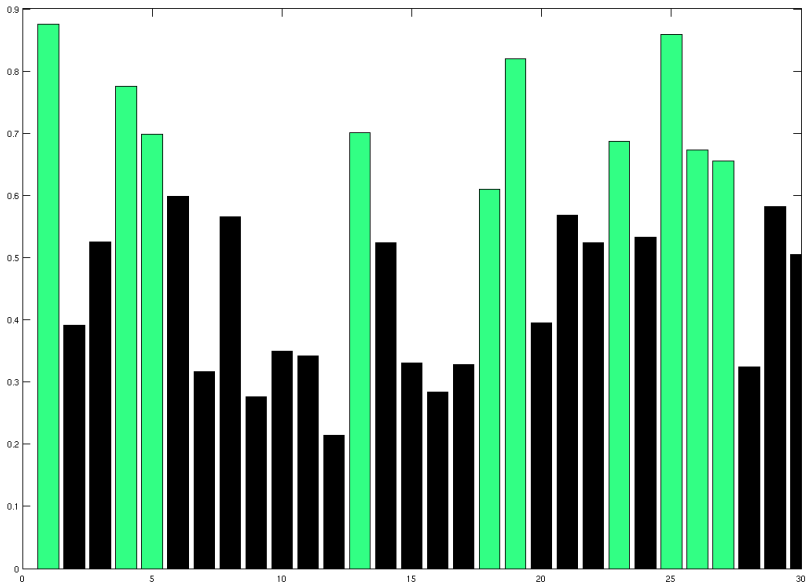


Figure 6.21: Appearance percentage. The utilised variables are marked in a lighter color.

Results: Using the variables found through the GA stage of GNMM, the four different ANNs were re-trained. The ANN sizes were kept the same with the exception that the input layer is adjusted to the reduced dimensions in the input data (10 vs 30). The results of the new ANNs are shown in Table 6.13. An overall comparison of the performance before and after variable selection is shown in Table 6.14. One can note that for MLP, TDNN and PNN the results surpass those obtained using all the variables in the ERS data while for the RBFN network the results are slightly worse.

Discussion The improved results indicates that some of the dimensions in the dataset do not provide useful information and can be removed without a loss in performance. Reducing the number of dimensions leads to a smaller input set which means that the training of the ANNs is faster and the resulting ANNs will be simpler and thus

Table 6.13: Results of the application of ANNs using the simplified ERS data

Network type	Error Type	Dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
MLP	Type I	Training	95.89	1	13	90.85	99.50	93.84	99.23
		Test	92.55	6	13	87.13	96.10	91.93	93.62
TDNN	Type I	Training	96.48	1	11	92.25	99.50	94.74	99.24
		Test	95.69	2	9	91.09	98.70	94.41	97.87
RBFN	Type I	Training	95.67	6	8	96.11	96.97	96.46	95.83
		Test	92.16	9	11	88.54	94.34	93.17	90.43
PNN	Type I	Training	98.63	4	3	98.58	98.66	98.99	98.12
		Test	93.82	6	15	88.72	97.10	93.06	95.16

Table 6.14: Comparison of ANN performance on the ERS data before and after variable selection using GNMM

Network type	Error Type	Before GNMM	After GNMM
MLP	TypeI	64.86	**66.70**
TDNN	TypeI	74.08	**79.49**
RBF	TypeI	**76.66**	64.84
PNN	TypeI	37.15	**71.57**

faster to execute as well.

6.4 Advantages and Disadvantages of an ANN approach

This section describes some of the overall advantages and disadvantages of ANNs in general. This allows for a general comparison with other techniques investigated in this thesis. Let us first look at the potential advantages of ANNs:

- Most of the ANNs are easy to set up and train and examples and literature is readily available.
- Execution time of trained ANN is usually very fast, though some exceptions can occur for large networks especially for RBFNs and PNNs or similar networks.
- ANNs are a black box approach. By this it is meant that there is no need to

Table 6.15: A comparison of performance values for each of the approaches explored

		IS data			ERS
Approach	Network	TypeI	TypeII	TypeIII	TypeI
Regular	MLP	86.67	48.63	66.87	65.32
	TDNN	88.49	43.65	93.49	74.08
	RBFN	79.34	94.97	50.54	74.66
	PNN	95.29	94.74	82.64	37.15
Weighted ensemble	Ensemble	95.29	94.97	71.59	76.26
Using GNMM	MLP	81.51	49.37	49.17	66.70
	TDNN	90.68	59.03	22.60	79.49
	RBFN	80.23	94.97	49.66	64.84
	PNN	95.29	84.29	82.64	71.57

know the underlying model of the system being modelled.

Likewise, following is a list of some of the potential disadvantages of ANNs:

- There exists no explanation facility for ANNs. This means that the results are not necessarily easy to understand and interpret. It can also make the process of incorporating an ANN into an existing system more difficult.
- ANNs can be time consuming, in particular the training, but also the execution in some cases.
- There does not exist a 'fool-proof way of determining the optimal topology of the networks, leading to the use of trial-and-error approaches.
- Some networks such as PNN suffer from the 'curse-of-dimensionality' making them difficult to use when the number of inputs is very high.

6.5 Discussions and conclusions

In this chapter it has been shown that errors in the TDAQ system can be detected based on both the IS and ERS data.

For the IS data the best overall results are achieved by the PNN network achieving 100% classification accuracy for all errors using the training data and a worst case of 99.59% using the test data. Though it is worth noting that all the different network types achieve very good results in achieving classification accuracies of above 98% for all error types. The fact that all the networks achieve such a high classification rate indicates that there are very clear patterns in the data corresponding to the error scenarios we have investigated. It has also been shown that the addition of noisy replicates to the dataset improved the overall classification rates.

The situation in the case of the ERS data is a bit different. The ERS data are only suitable for the TypeI error scenario (see Section 5.3.2.1). It does however show that a very high detection rate can be achieved using this type of data as well. The best results were achieved by the TDNN and the RBF both achieving a classification accuracy of 94.51% for the test set. However the RBF achieves better results using the performance measure introduced in Section 6.2.1.1 and can therefore be considered a slightly better overall classifier.

The ensemble approach shows that even better results can be obtained for both datasets by combining the outputs of the individual ANN approaches. The ensemble was not able to improve the classification rates for the IS data, but provided some improvements for the ERS data achieving a classification accuracy of 94.90% for the test dataset. The overall performance was also increased to 76.26 compared to 74.66 achieved using the RBFN. Naturally the ensemble approach will require more computational resources in order to train it, and hence it is often better used after other networks have been applied. In cases where training time is not a major factor the tradeoff of using an ensemble could be beneficial. The execution time of the ensemble will always be equal to the slowest component that is a part of it and thus the ANNs, or other classifiers, taking part in it should be chosen with care.

In the following chapter different SVM approaches will be explored using the same data as was used for the different ANNs. This is done in order to validate the results found so far using ANNs and to compare the ANN and SVM approaches in order to determine whether one of the two approaches is more suitable than the other to be used within the context of the ATLAS TDAQ system.

References

Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Comput.*, 8(3):643–674, 1996. ISSN 0899-7667.

Niel. A. Carlson. *Foundations of Physiological Psychology.* Simon & Schuster, 1992.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

Du Danfeng, Ma Yan, and Guo Xiurong. Application of pnn to fault diagnosis of ic engine. In *Proc. Second International Conference on Intelligent Computation Technology and Automation ICICTA '09*, volume 2, pages 495–498, 2009. doi: 10.1109/ICICTA.2009.354.

M. B. de Almeida, A. P. Braga, J. P. Braga, J. C. Belchior, and G. F. G. Yared. Radial basis function networks for obtaining long range dispersion coefficients from second virial data. *Physical Chemistry Chemical Physics (Incorporating Faraday Transactions)*, 2:103–107, 2000. doi: 10.1039/a906489c.

Jason G. Digalakis and Konstantinos G. Margaritis. An experimental study of benchmarking functions for evolutionary algorithms. *In Proceedings of IEEE Conference on Transactions, Systems, Man and Cybernetics*, 79:3810–3815, 2002.

T. Ganchev, N. Fakotakis, D. K. Tasoulis, and M. N. Vrahatis. Generalized locally recurrent probabilistic neural networks for text-independent speaker verification. In *Proc. IEEE International Conference on Acoustics, Speech, and*

Signal Processing (ICASSP '04), volume 1, pages I–41–4 vol.1, 2004. doi: 10.1109/ICASSP.2004.1325917.

M. W. Gardner and S. R. Dorling. Artificial neural networks (the multilayer perceptron)–a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14-15):2627 – 2636, 1998. ISSN 1352-2310. doi: DOI:10.1016/S1352-2310(97)00447-0.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979. ISBN 0716710455.

David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0201157675.

Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998. ISBN 0132733501.

John H. Holland. Outline for a logical theory of adaptive systems. *J. ACM*, 9(3): 297–314, 1962. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321127.321128.

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992. ISBN 0262581116.

L. Holmstrom and P. Koistinen. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1):24–38, 1992. ISSN 1045-9227. doi: 10.1109/72.105415.

Yuansheng Huang and Chengfang Tian. Research on credit risk assessment model of commercial banks based on fuzzy probabilistic neural network. In *Proc. International Conference on Risk Management & Engineering Management ICRMEM '08*, pages 482–486, 2008. doi: 10.1109/ICRMEM.2008.33.

A. Khotanzad and C. Chung. Application of multi-layer perceptron neural networks to vision problems. *Neural Computing & Applications*, 7(3):249–259, September 1998.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. ISSN 00368075.

P. Koistinen and L. Holmstrom. Kernel regression and backpropagation training with noise. pages 367–372 vol.1, Nov 1991. doi: 10.1109/IJCNN.1991.170429.

Sauchi Stephen Lee. Noisy replication in skewed binary classification. *Computational Statistics & Data Analysis*, 34(2):165–191, August 2000.

J.A. Leonard and M.A. Kramer. Radial basis function networks for classifying process faults. *Control Systems Magazine, IEEE*, 11(3):31 –38, apr 1991. ISSN 0272-1708. doi: 10.1109/37.75576.

K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathmatics*, II(2):164–168, 1944.

Yong-Chun Liang, Xiao-Yun Sun, Dong-Hui Liu, and Hui-Qin Sun. Application of combinatorial probabilistic neural network in fault diagnosis of power transformer. In *Proc. International Conference on Machine Learning and Cybernetics*, volume 2, pages 1115–1119, 2007. doi: 10.1109/ICMLC.2007.4370311.

Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007. ISBN 978-3-540-69431-1.

Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2): 431–441, 1963. doi: 10.2307/2098941.

Warren Mcculloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, December 1943. doi: http://dx.doi.org/10.1007/BF02478259.

K. Meng, Z. Y. Dong, D. H. Wang, and K. P. Wong. A self-adaptive rbf neural network classifier for transformer fault analysis. *Power Systems, IEEE Transactions on*, PP(99):1 –1, 2010. ISSN 0885-8950. doi: 10.1109/TPWRS.2010.2040491.

H. Mori and A. Awata. Data mining of electricity price forecasting with regression tree and normalized radial basis function network. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 3743 –3748, oct. 2007. doi: 10.1109/ICSMC.2007.4414228.

M.J.L. Orr. Introduction to radial basis function networks. Technical report, Institute for Adaptive and Neural Computation of the Division of Informatics at Edinburgh University, Scotland, UK, 1996.

Stanislaw Osowski, Robert Siroic, and Krzysztof Siwek. Genetic algorithm for integration of ensemble of classifiers in arrhythmia recognition. In *Proc. IEEE Intrumentation and Measurement Technology Conference I2MTC '09*, pages 1496–1500, 2009. doi: 10.1109/IMTC.2009.5168692.

J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991. doi: 10.1162/neco.1991.3.2.246.

M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964. doi: 10.1093/comjnl/7.2.155.

J. A. Fernndez Prieto and Juan R. Velasco Prez. Adaptive genetic algorithm control parameter optimization to verify the network protocol performance. In *12th International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU)*, pages 785–791, 2008.

W. Ritschel, T. Pfeifer, and R. Grob. Rating of pattern classifications in multi-layer perceptrons: theoretical background and practical results. pages 142–144, 1994. doi: http://doi.acm.org/10.1145/326619.326684.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel distributed processing: explorations in the microstructure of cognition*, 1:318–362, 1986.

D.F. Specht. Enhancements to probabilistic neural networks. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 1, pages 761–768 vol.1, Jun 1992. doi: 10.1109/IJCNN.1992.287095.

Donald F. Specht. Probabilistic neural networks. *Neural Netw.*, 3(1):109–118, 1990. ISSN 0893-6080. doi: http://dx.doi.org/10.1016/0893-6080(90)90049-Q.

B. Tian, M. R. Azimi-Sadjadi, T. H. Vonder Haar, and D. Reinke. Temporal updating scheme for probabilistic neural network with application to satellite cloud classification. 11(4):903–920, 2000. ISSN 1045-9227. doi: 10.1109/72.857771.

J. Torres-Sospedra, M. Fernandez-Redondo, and C. Hernandez-Espinosa. A research on combination methods for ensembles of multilayer feedforward. In *Proc. IEEE International Joint Conference on Neural Networks IJCNN '05*, volume 2, pages 1125–1130 vol. 2, 2005.

Ray Tsaih and Chih-Chung Lin. *The Layered Feed-Forward Neural Networks and Its Rule Extraction*, pages 377–382. Springer Berlin / Heidelberg, 2004.

L. Valiant. Functionality in neural nets. In *Annual Workshop on Computational Learning Theory: Proceedings of the first annual workshop on Computational learning theory*, 1988.

A. Waibel, A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. 37(3):328–339, 1989. ISSN 0096-3518. doi: 10.1109/29.21701.

G.I. Webb and Z. Zheng. Multistrategy ensemble learning: reducing error by combining ensemble learning techniques. 16(8):980–991, 2004. ISSN 1041-4347. doi: 10.1109/TKDE.2004.29.

David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Jianhua Yang, Evor L. Hines, Ian Guymer, Daciana D. Iliescu, and Mark S. Leeson. Multi-input optimisation of river flow parameters and rule extraction using genetic-neural technique. In E.L. Hines et al, editor, *Intelligent Systems: Techniques and Applications*, number 978-90-423-0345-4. Shaker publishing, 2008a.

Jianhua Yang, Evor L. Hines, Ian Guymer, Daciana D. Iliescu, Mark S. Leeson, Gregory P. King, and XuQuin Li. Genetic algorithm-artificial neural network method for the prediction of longitudinal dispersion coefficient in rivers. In *Advancing Artificial Intelligence through Biological Process Applications*, pages 358–374. Idea Group Inc., 2008b.

Kuihe Yang, Ganlin Shan, and Lingling Zhao. Application of wavelet packet analysis and probabilistic neural networks in fault diagnosis. In *Proc. Sixth World Congress on Intelligent Control and Automation WCICA 2006*, volume 1, pages 4378–4381, 2006. doi: 10.1109/WCICA.2006.1713204.

Chapter 7

Classification of errors using Support Vector Machines

This chapter provides an introduction to the theory of Support Vector Machines (SVMs). It shows how non-linear and non-separable data can be dealt with by using SVMs and also presents approaches for dealing with multi-class data. SVMs are then applied to detect errors in the TDAQ system using the same input data as was used in the previous chapters. The results are compared to that of the ANN approach in Chapter 6 and the advantages/disadvantages of a SVM approach are then discussed along with conclusions. SVMs where chosen as the technique is particularly useful for sparse and/or imbalanced datasets. This is true for the TDAQ data where the number of positive cases are low compared to the total size of the dataset.

7.1 Introduction to support vector machines (SVMs)

This section provides a brief overview of SVM theory. For a complete description the reader is advised to consult (Burges, 1998).

SVMs are generalised linear discrimination machines that take advantage of operating in a transformed, typically higher dimensional, space in order to best discriminate classes of data (Vapnik, 1995). SVMs are an area of considerable research and has a range of applications including image classification (Chapelle et al., 1999),

novelty detection (Schölkopf et al., 2000), protein folding (Rangwala and Karypis, 2005), face recognition (Osuna et al., 1997), speech recognition (Ganapathiraju et al., 2000), video classification (hao Lin, 2002) and robotic control (Pelossof et al., 2004) to name a few.

Let us first consider the core concepts of SVMs before moving on to consider nonlinear and non-separable data:

7.1.1 Separating hyperplanes

A hyperplane in an m dimensional space can be defined by

$$a_1 x_1 + \cdots + a_m x_m = b \tag{7.1}$$

This hyperplane will divide the feature space into two halves:

$$a_1 x_1 + \cdots + a_n x_n <= b$$

and

$$a_1 x_1 + \cdots + a_n x_n > b$$

hence providing *some* classification of the feature space. A *separating* hyperplane can then be defined as follows:

Given a set of data $[x_i, y_i]$, $i = 1, \ldots, n$ where all the data points belong to either one of two classes. The labeling of the classes is set to -1 or 1 for practical reasons and without loss of generality so that $y_i \in [-1, 1]$. A separating hyperplane is a hyperplane such that all data points of the same class are on the same side of the hyperplane and all points of the other class are on the other side. A separating hyperplane is therefore a hyperplane subject to the constraint of:

$$y_i(w \cdot x_i + b) > 0,\ i = 1, \ldots, n \tag{7.2}$$

If a hyperplane satisfying (7.2) exists, the dataset is said to be *linearly* separable. In such cases it is always possible to re-scale w such that:

$$\min_{1 \leq i \leq n} y_i(w \cdot x_i + b) \geq 1,\ i = 1, \ldots, n \tag{7.3}$$

The distance from a hyperplane to the closest data point is called the *margin*. The hyperplane with the largest possible margin is referred to as the Optimal Separating Hyperplane (OSH) and is always unique. Hence the OSH can be found by maximising the margin. This is equivalent to minimising $\frac{2}{||w||^2}$ subject to (7.3) and can be solved using Quadratic Programming (QP). The hyperplanes found while maximising the margin are called *support vectors*, hence the name of the method as a whole. As an example consider a dataset of points, $(x_1, y_1), \ldots, (x_n, y_n)$, belonging to one of two separate classes as shown in Figure 7.1. The OSH can then be calculated and is shown together with the support vectors in Figure 7.2.

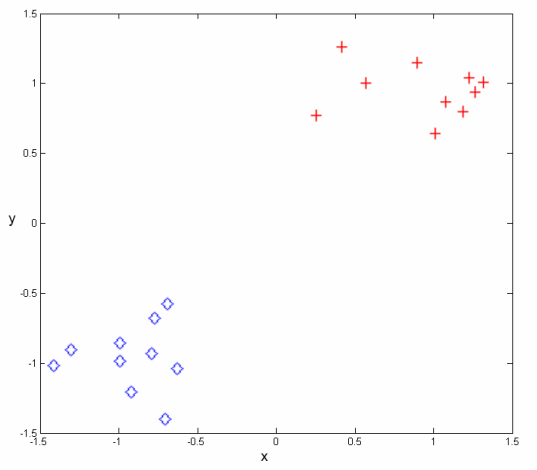


Figure 7.1: An example dataset where the points belong to two separate classes.

7.1.1.1 Nonlinear SVM

In many, or perhaps even most, practical problems the data is not linearly separable. An example of such a dataset is shown in Figure 7.3. If one attempts to classify the data using a linear kernel the results will usually be unsatisfactory as shown in Figure 7.4.

In order to deal with this problem one can use *nonlinear* SVMs. Nonlinear SVMs are based on the idea that the input data can be mapped to a high-dimensional *feature space* and the optimal separating hyperplane can be constructed in this

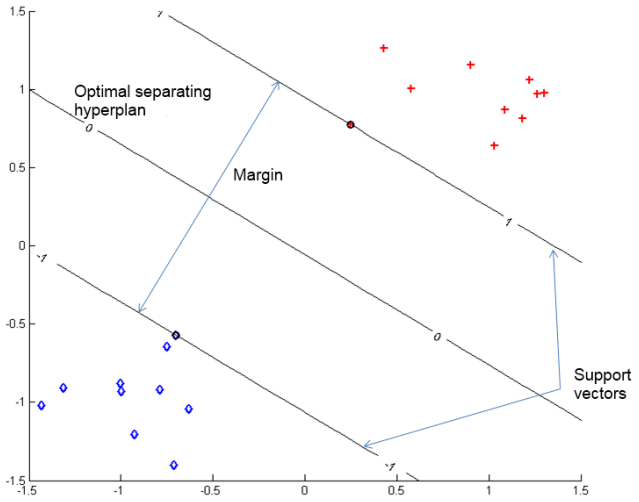


Figure 7.2: Shows the optimal separating hyperplane and the corresponding support vectors for an example dataset containing points belonging to two separate classes.

feature space instead. Consider a mapping of:

$$\Phi : R^d \rightarrow \mathcal{H} \tag{7.4}$$

From (7.3) we have that only the dot product $x_i \cdot x_j$ is needed for training, hence only the dot products in $\mathcal{H}$ would be needed to calculate the OSH, i.e. $\Phi(x_i) \cdot \Phi(x_j)$. It has been shown in (Vapnik, 1995) that an old trick first presented by Aizerman in (Aizerman et al., 1964) can be used to solve the problem. By choosing a kernel function K such that $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, then it follows that only K is needed in order to train the SVM. There is no need to actually map the data into the feature space, nor indeed is there a need to know the mapping function at all. Figure 7.5 shows how a nonlinear SVM using a gaussian kernel can classify data which is not linearly separable in the input space.

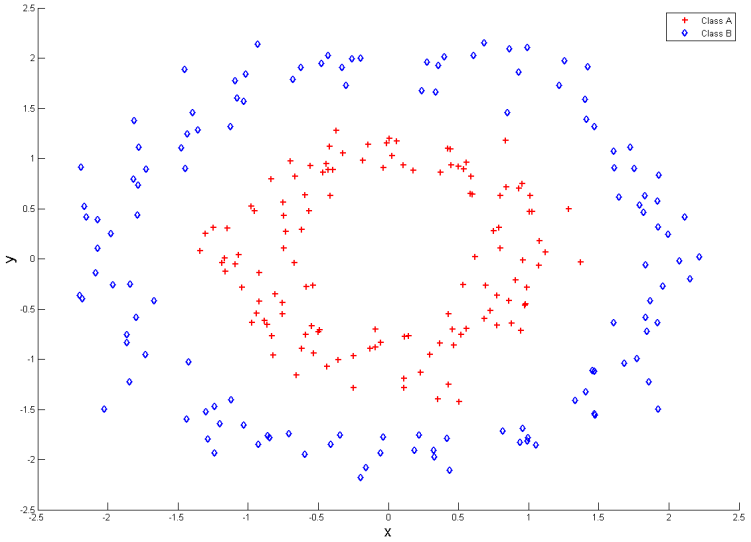


Figure 7.3: Example of a dataset which is not linearly separable in the input space.

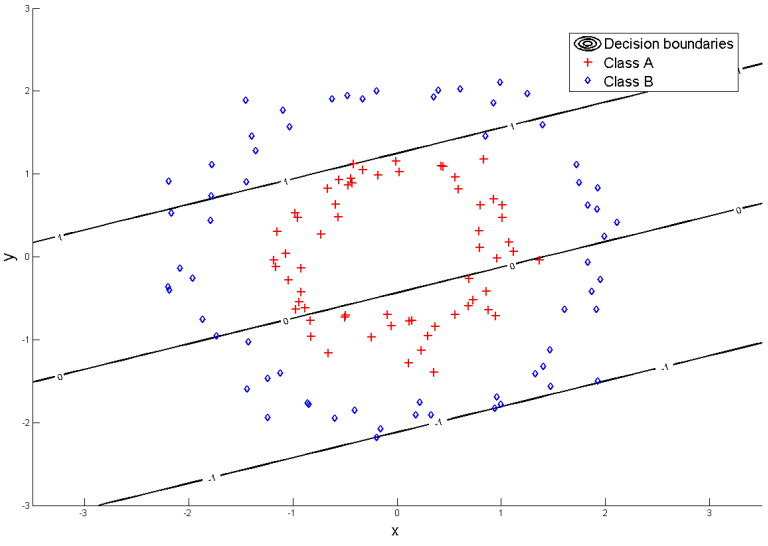


Figure 7.4: A linear SVM is unable to classify the data.

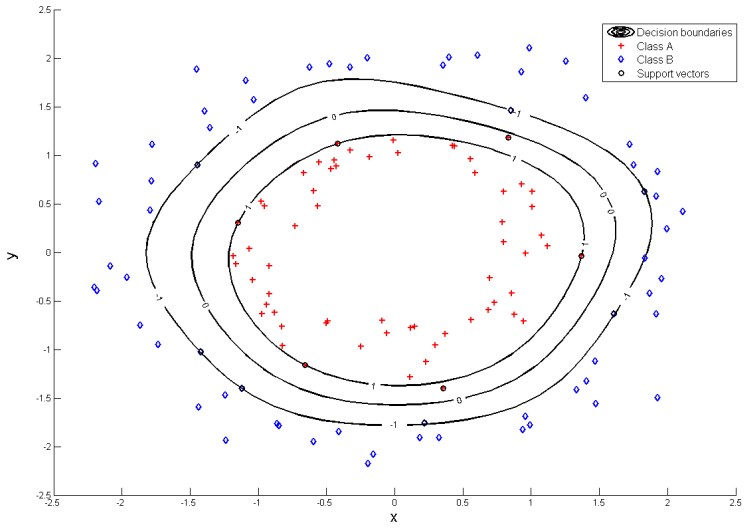


Figure 7.5: Example of how a non-linear SVM can classify data which is not linearly separable.

7.1.2 Non-separable data

Also for many cases the data might not be separable at all. For example the addition of noise can lead to data where there exists an overlap of the different classes. In this case the regular training using the data would try to map to this noise and lead to a poor generalisation performance. In order to deal with this problem it is common to introduce an error margin as described in more detail below:

7.1.2.1 Error margin

An error margin, also referred to as a *soft margin*, can be introduced in order to allow some samples of the data to violate the constraint in 7.2. Each variable is assigned an error margin ξ_i:

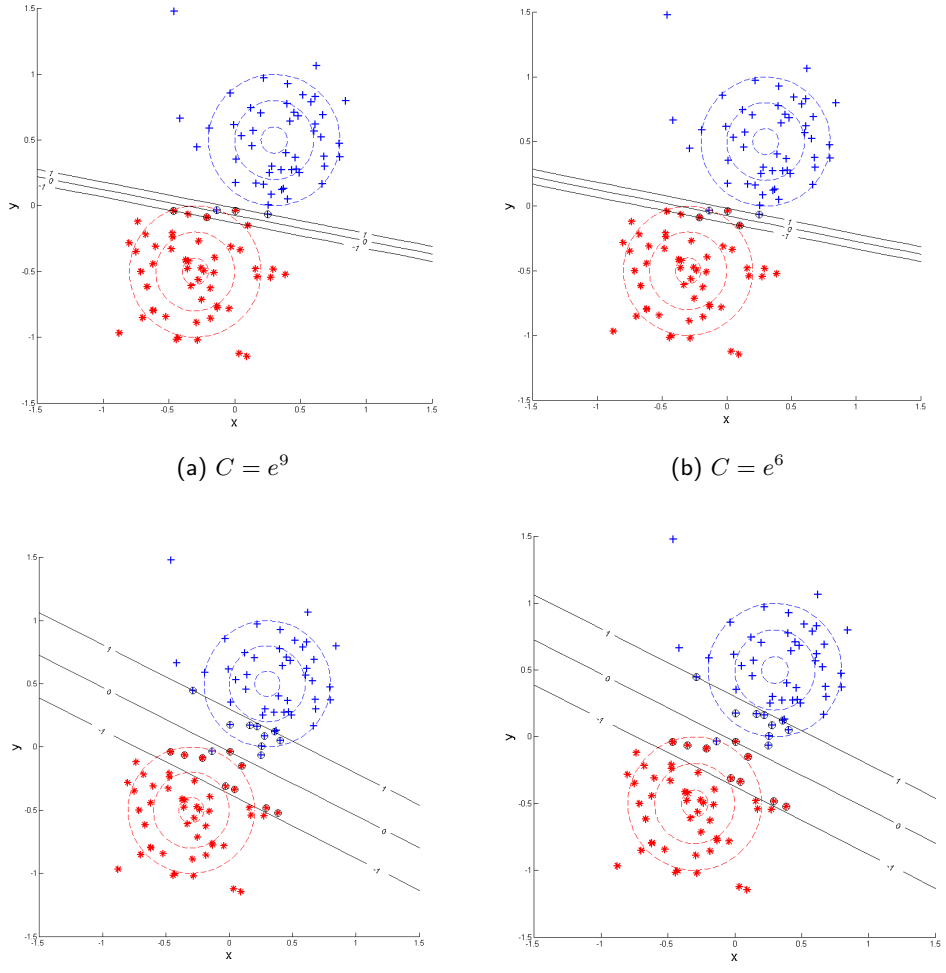
$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, i = 1, \ldots, n \tag{7.5}$$

It therefore follows that any sample with $\xi > 1$ is misclassified and hence $\sum \xi_i$ is an upper bound on the number of misclassified points. QP can now be used to minimise:

$$C \sum \xi_i + \frac{2}{||w||^2} \tag{7.6}$$

where the constant factor C controls the penalty for misclassification. The factor C can be configured by the user, and a larger value of C will yield a higher penalty for each misclassification. One of the benefits of using a soft margin is that this also reduces the impact of outliers in the data, ensuring a better generalisation ability of the trained SVM in cases where outliers exist. However, it must be chosen carefully as too high a value of C means that any misclassified result will dominate (7.5) and may therefor 'force' the SVM to fit the data too closely leading to overfitting. Similarly, a very low value for C will mean that only a very low penalty is given for misclassified sample something that could lead to a poor generalization performance as the SVM separates the data too 'loosely'. Figure 7.6 illustrates the effect of changing the parameter C for an example single class problem. The data has been generated using a Gaussian distribution to create two clusters of points at the centers $[-.3, -.5]$ and $[.3, .5]$ with the width of the Gaussian function of $\sigma = 0.3$. As can

be seen in Figure 7.6a a high value of C (e^9) leads to a good fit of the sample data, but does not separate well the actual underlying distribution (represented by the coloured circles). Hence, it would achieve a high classification accuracy for the training data, but a poorer performance for the general dataset (i.e. data drawn from the Gaussian distribution around the centers $[-.3, -.5]$ and $[.3, .5]$) compared to a choice of $C = e^3$ as shown in Figure 7.6c.



(a) $C = e^9$ (b) $C = e^6$

(c) $C = e^3$ (d) C=1

Figure 7.6: The performance of a SVM classifier trained with different values for C.

7.1.3 Strategies to deal with multi-class problems

The basic SVM approach as described above are intrinsically binary (Melgani and Bruzzone, 2004) and thus ill-suited when the data contains multiple classes. This is a very common situation and is also present in the data from the TDAQ system where a number of different error types should be recognised. There exist several approaches as to overcome this problem in order to classify multiple classes. Let us consider the two most common approaches in turn:

7.1.3.1 One-against-one (OAO)

In OAO classification a separate SVM is trained per class *pair*. That is, given a dataset of n classes the data are divided into all possible class pairs and one SVM is trained to distinguish each such pair. In order to determine the class of a given data sample a voting system is used where the class getting the most number of votes is chosen. The number of SVMs needed for this technique will increase quickly given by: $\binom{N}{2} = \frac{n!}{2!(n-2)!} = \frac{(n(n-1))}{2}$, where N is the number of classes. Hence, this approach is more computationally expensive compared to that of One-Against-All (OAA) (see below) and might be unfeasible if the number of classes to be classified is very large. As an example consider a dataset containing 6 different classes. In that case there would be needed $\binom{6}{2} = \frac{6!}{2!(6-2)!} = 15$ different SVM classifiers. For a dataset containing 10 classes the number of classifiers would be $\binom{10}{2} = \frac{10!}{2!(10-2)!} = 45$ and so would quickly becoming unmanageable.

The result of an example OAO classification is shown in Figure 7.7. Each line represents the decision boundary of a single SVM.

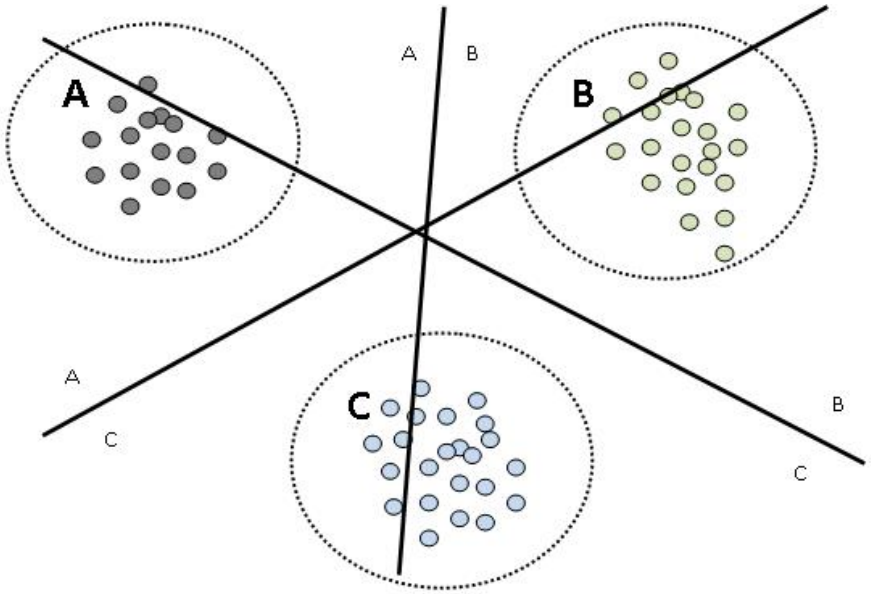


Figure 7.7: 'One-Against-One' algorithm.

7.1.3.2 One-against-all (OAA)

In the OAA, one SVM is used to distinguish each class from *all* the other classes. So given a dataset of n classes, n SVMs will have to be trained. An illustration of this technique is shown in Figure 7.8 where three different SVMs each classify one class of the data as distinct from all the others. Note that this method might leave a region of feature space 'unclassified' in the sense that all of the SVMs will reject it and no class will therefore be assigned to the sample. Still, the computational effort increases linearly with the number of classes in the dataset and is therefore significantly computationally cheaper then OAO when the number of classes is high.

7.1.4 Determining SVM kernel and parameters

Two main parameters must be determined when using SVM. First of all, a kernel must be chosen before any parameters specific to the kernel are chosen. Let us consider each in turn:

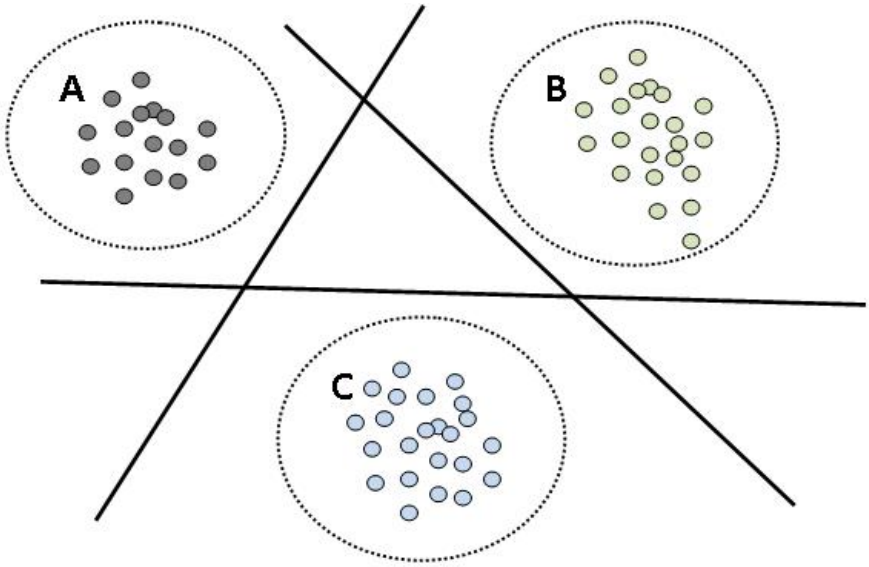


Figure 7.8: 'One-Against-All' algorithm.

7.1.5 Choosing kernel function

An important parameter when training SVMs is the choice of kernel function. A number of possible kernels for SVM exist, including the Gaussian (7.7), polynomial (7.8) and sigmoidal (7.9) kernels, respectively

$$e^{-\frac{||x-y||^2}{2\sigma}} \tag{7.7}$$

$$((x \cdot y) + \beta)^d \tag{7.8}$$

$$\tanh(\delta(x \cdot y) + \beta) \tag{7.9}$$

There is no general way to choose the correct kernel for a given application (ichi Amari and Wu, 1999). Often, prior knowledge about the dataset in question is needed by an expert user, something that often is not possible. However the kernels listed above are commonly used in the literature and work well for a large number of problems.

7.1.5.1 Kernel parameters

Depending on the kernel used there will be one or more parameters that must be selected by the operator. If an error margin is used this must also be determined.

Naturally one would aim to choose these parameters in such a way as to optimise the generalisation error of the SVM. Sometimes the parameters can be estimated by an expert user, but often this is not the case. In such cases other approaches to determining the parameters are needed. While the straightforward approach of doing a simple grid-search, wherein a range is set for each parameter and each combination of parameters are subsequently tested, is still popular a number of better approaches have been proposed. The area is of great interest as grid-search is intractable if the number of parameters are any more than two. Some of the possible extensions include the use of GAs (Rojas and Fernandez-Reyes, 2005; Frölich et al., 2003; Liu et al., 2005) and the calculation of the inter-cluster distance in the feature space (Wu and Wang, 2009). Chapelle proposes a gradient descent methods using various bounds of the generalisation error with respect to the kernel parameters (Chapelle et al., 2002) and shows very promising results as an automatic way of selecting the parameters. Hence a number of different approaches exists and their usefulness depends to some extent on the particular application. Grid search is therefore still widely used in current literature.

7.2 Applying SVM to error detection in TDAQ

In the following section we will apply SVMs in order to detect errors in the TDAQ system as was done in Chapter 6 using the ANN approach. The exact same data was used as for the ANNs, both training and testing, in order to properly compare the methods. An SVM implementation by Canu et al. (2005) was used throughout this chapter.

7.2.1 SVM using the IS data

As mentioned in Section 7.1.5 there is no foolproof way of choosing a kernel beyond experimental tests on a particular problem. However the polynomial and Gaussian kernels are common choices in literature and were therefore used in this work. The parameters for each of the kernels were determined using the grid-search approach.

The grid search was performed by evaluating all combinations of a range of values for C and the kernel parameter σ. σ represents the polynomial order when using the polynomial kernel and the width of the radial basis function when using the Gaussian kernel. While this is computationally expensive it was used for the sake of simplicity in the implementation, especially considering that we are working with multi-class SVMs. Computational time was not a key consideration during these initial tests of SVMs, but might have to be considered if an SVM approach is attempted on a large scale or within an online learning system in the future.

The OAA algorithm was used with both the polynomial and the Gaussian kernel. As there are 4 different classes (TypeI, TypeII, TypeIII and no error), this results in 4 distinct SVMs being trained. The specifics of the two approaches will be discussed in turn:

7.2.1.1 Polynomial kernel

There are two parameters to be set for the polynomial kernel, namely the error bound C and the polynomial order σ. Figure 7.9 shows a contour plot of the results of the grid search for the polynomial kernel. One can observe that the best results are obtained for value of $C = 10^{-3}$ and $\sigma = 2$. These values were then used resulting in 4 SVMs with 208, 44, 77 and 116 support vectors respectively.

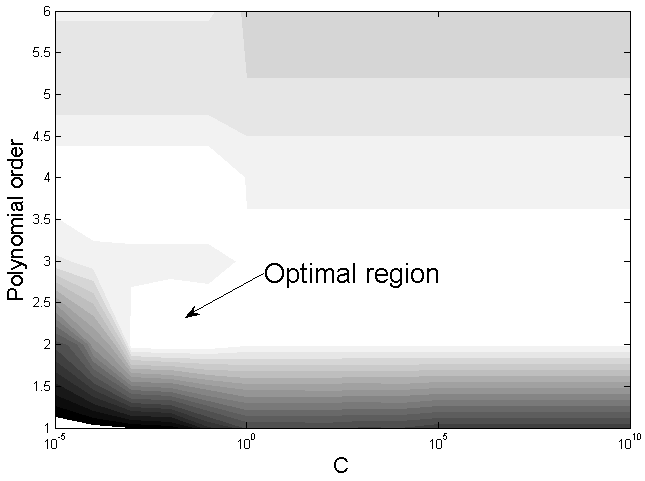


Figure 7.9: Result of grid search for the polynomial kernel parameters.

7.2.1.2 Gaussian kernel

For the Gaussian kernel the parameters to be determined are the error bound C and the width σ of the radial basis function. A grid search was performed as for the polynomial kernel and the results are shown in Figure 7.10. Based on these tests the optimal values were determined to be at $C = 10^{2.5}$ and $\sigma = e^2$. The SVMs were trained using these values resulting in 4 SVMs consisting of 744, 728, 95 and 68 support vectors. It is important to note that the first two SVMs has a very high number of support vectors (744 and 728 out of 1461 observations in the dataset). This might indicate that the approach is unsuitable for larger datasets as SVMs do not handle a large number of support vectors very well due to the use of QP.

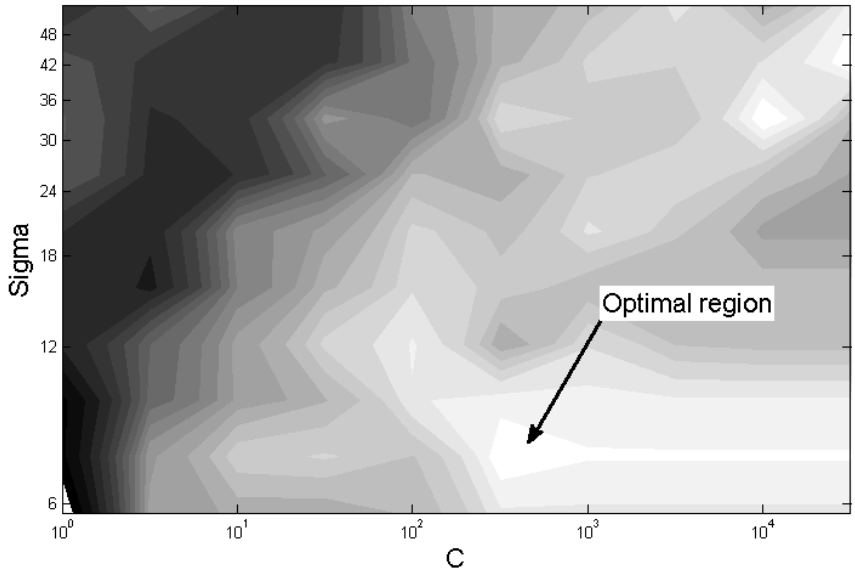


Figure 7.10: Result of grid search for the Gaussian kernel parameters.

7.2.1.3 IS results

The results of the trained SVMs are shown in Table 7.1. One can immediately observer that the Gaussian kernel SVM achieves perfect results for all three error types, correctly classifying all samples. The Polynomial kernel also achieves excellent results with a classification accuracy ranging from 99.45% for the TypeI error to 100% for the TypeII error.

By using the same performance criteria as in Section 6.2.1.1, we can compare the SVM results with that achieved using the various ANN approaches. The result of this comparison can be seen in Table 7.2. One can note that the SVMs are in general better classifiers than the ANNs. The PNN is closest in overall performance achieving results close to that of the polynomial kernel SVM.

Table 7.1: Results of the application of SVMs to the IS data

SVM kernel	Error Type	dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
Gaussian	Type I	Training	100.0	0	0	100.0	100.0	100.0	100.0
		Test	100.0	0	0	100.0	100.0	100.0	100.0
	Type II	Training	100.0	0	0	100.0	100.0	100.0	100.0
		Test	100.0	0	0	100.0	100.0	100.0	100.0
	Type III	Training	100.0	0	0	100.0	100.0	100.0	100.0
		Test	100.0	0	0	100.0	100.0	100.0	100.0
Polynomial	Type I	Training	100.0	0	0	100.0	100.0	100.0	100.0
		Test	99.45	1	3	99.57	97.22	92.11	99.86
	Type II	Training	100.0	0	0	100.0	100.0	100.0	100.0
		Test	100.0	0	0	100.0	100.0	100.0	100.0
	Type III	Training	100.0	0	0	100.0	100.0	100.0	100.0
		Test	99.73	0	2	99.72	100.0	90.48	100.0

Table 7.2: Comparison of performance of ANN and SVM models

Network type	TypeI	TypeII	TypeIII
MLP	86.67	48.63	66.87
TDNN	88.49	43.65	93.49
RBF	79.34	94.97	50.54
PNN	95.29	94.74	82.64
Gaussian SVM	**100.0**	**100.0**	**100.0**
Polynomial SVM	88.55	**100.0**	89.98

7.2.2 SVM using the ERS data

As for the IS data two different kernels were utilised; the Gaussian and the polynomial. The parameters were again determined using a grid-search approach. The ERS data consist of just two error classes (TypeI or no error) and hence it is enough to train a single SVM.

7.2.2.1 Polynomial kernel

There are two parameters to be set for the polynomial kernel, namely the error bound C and the polynomial order σ. The same approach was taken as in Section 7.2.1.1 which identified $C = 10^{-1}$ and $\sigma = 1.5$ as optimal values. These values were then used to train the final SVM resulting in a SVM with 259 support vectors.

7.2.2.2 Gaussian kernel

For the Gaussian kernel the parameters to be determined are the error bound C and the width σ of the radial basis function. A grid search was performed as for the polynomial kernel identifying the optimal values as $C = 1$ and $\sigma = e^{-}1$. The final SVM was then trained using these values and the final SVM uses 281 support vectors.

7.2.3 ERS results

The results of the Gaussian and polynomial kernel SVMs are summarised in Table 7.3, while the comparison to the ANN results are shown in Table 7.4. One can note that the Gaussian kernel SVM again achieves the best results with a test set classification accuracy of 92.64% as compared to 92.35% for the polynomial kernel SVM. Compared to the ANNs the two SVM achieves good results performing better than the MLP and PNN, but slightly worse than that of the TDNN and RBFN approaches.

Table 7.3: Results of the application of SVMs to the ERS data

SVM kernel	Error Type	Dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
Gaussian	Type I	Training	94.32	5	24	88.84	98.31	92.38	97.45
		Test	92.64	5	20	84.62	97.62	91.11	95.65
Polynomial	Type I	Training	93.93	7	24	88.84	97.64	92.33	96.46
		Test	92.35	4	22	83.08	98.10	90.35	96.43

Table 7.4: Comparison of performance of ANN and SVM models for the ERS data

Network type	TypeI
MLP	65.32
TDNN	74.08
RBF	**74.66**
PNN	37.15
Gaussian SVM	66.69
Polynomial SVM	65.58

7.3 Advantages and disadvantages of SVM

Following is an overview of the disadvantages and advantages of SVMs in general and in particular in comparison to ANNs as presented in the previous chapter.

Advantages:

- The SVM problem is convex and therefore contains no local minima. Hence the same solution will be derived regardless of 'starting point'. This is, for example, not true for ANNs where a different starting point might yield a different solution.
- The SVM automatically determines their model size (support vectors) while solving the QP, while for ANNs the model size must be determined by the user.
- Good generalisation can be achieved quite easily, especially with the use of soft margins.
- SVMs contain very few parameters to be determined after the kernel has been chosen.

Disadvantages: Burges (Burges, 1998) presents 4 main limitations to SVMs and are summarised below:

- According to Burges trying to find the best choice of kernel for a given problem is still very much a research issue.
- The training and testing of a SVM can be slow, especially for large datasets, though the same is true in general for ANNs.
- Discrete/categorical data present problems while training the SVM as the algorithm operates on continuous data. However this can be largely solved using suitable re-scaling.
- Optimal design for multi class problems is still a matter of research. Though it should be mentioned that for problems where a limited number of classes are present, the OAA method obtains very good results. Still as the more error types are taken into consideration this could become a real problem.

7.4 Discussions and conclusions

Due to the excellent results and generalisation ability of the SVMs, they are a promising avenue for further tests and research. The results match that of ANNs and even surpasses them in the case of the IS data where the Gaussian SVM achieves a perfect classification accuracy of 100% for both test and training set. The Polynomial kernel also achieves very good results with a classification accuracy of 99.45% to 100% matching the performance of the best ANN method, the PNN.

For the ERS data the performance of the SVMs is slightly worse than those of the ANNs, but does provide good generalisation results nonetheless achieving a accuracy for the test set of 92.64% using the Gaussian kernel and 92.35% using the polynomial kernel.

In the following chapter an evolutionary approach will be explored utilising CGP in order to automatically develop the error detection program. Unlike the ANN and SVM approaches, which are to a great extent 'black-box' approaches, the CGP programs can be easily studied and modified by human experts. Such an approach is therefore of great interest for use within the ATLAS TDAQ system. The results achieved using the CGP approach are compared to that of the ANNs and SVMs and the advantages and disadvantages of the different approaches are discussed.

References

M A Aizerman, E M Braverman, and L I. Rozonor, theoretical foundations of the potential function method in pattern recognition learning. *Autom. Remote Contr*, (25):821–837, 1964.

Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy. Svm and kernel meth-

ods matlab toolbox. Perception Systmes et Information, INSA de Rouen, Rouen, France, 2005.

O. Chapelle, P. Haffner, and V.N. Vapnik. Support vector machines for histogram-based image classification. *Neural Networks, IEEE Transactions on*, 10(5):1055–1064, Sep 1999. ISSN 1045-9227. doi: 10.1109/72.788646.

Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3): 131–159, 2002. doi: 10.1023/A:1012450327387.

Holger Frölich, Bernhard Schölkopf, and Olivier Chapelle. Feature selection for support vector machines using genetic algorithms. *International Journal on Artificial Intelligence Tools*, pages 142–148, 2003.

A. Ganapathiraju, J. Hamaker, and J. Picone. Hybrid svm/hmm architectures for speech recognition. In *in Speech Transcription Workshop*, pages 504–507, 2000.

Wei hao Lin. News video classification using svm-based multimodal classifiers and combination strategies. In *In ACM Multimedia, Juan-les-Pins*, pages 1–6. ACM Press, 2002.

Shun ichi Amari and Si Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12:783–789, 1999.

Huan-Jun Liu, Yao-Nan Wang, and Xiao-Fen Lu. A method to choose kernel function and its parameters for support vector machines. In *Proc. International Conference on Machine Learning and Cybernetics*, volume 7, pages 4277–4280 Vol. 7, 2005. doi: 10.1109/ICMLC.2005.1527688.

F. Melgani and L. Bruzzone. Classification of hyperspectral remote sensing images with support vector machines. *Geoscience and Remote Sensing, IEEE Transactions on*, 42(8):1778–1790, Aug. 2004. ISSN 0196-2892. doi: 10.1109/TGRS.2004.831865.

E. Osuna, R. Freund, and F. Girosit. Training support vector machines: an application to face detection. In *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 130 –136, 17-19 1997. doi: 10.1109/CVPR.1997.609310.

Raphael Pelossof, Andrew Miller, Peter Allen, and Tony Jebara. An svm learning approach to robotic grasping. In *In IEEE International Conference on Robotics and Automation*, pages 3215–3218. 2004.

Huzefa Rangwala and George Karypis. Profile based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, page bti687, 2005. doi: 10.1093/bioinformatics/bti687.

S.A. Rojas and D. Fernandez-Reyes. Adapting multiple kernel parameters for support vector machines using genetic algorithms. In *Proc. IEEE Congress on Evolutionary Computation*, volume 1, pages 626–631 Vol.1, 2005. doi: 10.1109/CEC.2005.1554741.

Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. volume 12, 2000.

Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.

Kuo-Ping Wu and Sheng-De Wang. Choosing the kernel parameters for support vector machines by the inter-cluster distance in the feature space. *Pattern Recogn.*, 42(5):710–717, 2009. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2008.08.030.

Chapter 8

Classification of errors using Cartesian Genetic Programming

This chapter presents a description of Cartesian Genetic Programming (CGP) and describes how it can be applied to evolve programs capable of performing error detection within the ATLAS TDAQ system. Section 8.1 gives an introduction to CGP and describes how it compares to Standard Genetic Programming (SGP). It also introduces a new new type of crossover and compares it to the approach commonly used in the literature. Section 8.2 applies CGP in order to evolve programs capable of detecting errors in the TDAQ system. The approach is compared to the results found in previous chapters and the relative merits of the different approaches are discussed.

An important benefit of the CGP approach is that a program can be automatically developed to perform the error management. The developed solution is also easily modifiable by a human developer, something that is not always the case with ANNs and SVMs.

8.1 Introduction to Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) was first introduced by Julian F. Miller in 1998 and subsequently presented in (Miller, 1999) and later in (Miller and Thomson, 2000). Although it is based on the same concepts as evolutionary programming and

Standard Genetic Programming (SGP) (Koza, 1998), it does differ in some fundamental ways. The most important difference is the representation of the programs which is realised using a directed indexed graph, as opposed to the tree representation normally used in SGP. As initially presented by Miller, the nodes in CGP are organised in a grid structure of x rows and y columns, hence the name Cartesian. A parameter '*levels-back*' is an integer ranging from 1 to the total number of columns. If *levels-back* is set to 1, a node may only connect to nodes in the previous layer and so on. If the number of rows are set to 1 and levels-back equals the number of nodes the program becomes a general directed graph where a node may be connected to any of the previous nodes.

As in the case of SGP evolutionary operators such as mutation and crossover are applied to the individuals. CGP uses a fixed number of nodes, but due to the connection structure used in CGP the actual program developed will usually consist of a sub-set of these nodes. This mapping between genotype and phenotype allows for *neutrality* where a number of individuals have the same program representation (phenotype) and fitness, but may contain differences which are currently not contributing to the output. It has been argued that this kind of neutrality may be beneficial in evolutionary computing, but results are somewhat unclear (Galvanes-Lopez and Poli, 2006). CGP has been shown so far to perform at least as well as SGP in some applications (Miller, 1999; Walker and Miller, 2006; Miller and Thomson, 2003; Harding and Miller, 2005). Figure 8.1 shows a CGP individual and the corresponding program (phenotype).

Another significant difference in using CGP as opposed to SGP is that it offers the possibility to have *multiple* outputs from a single program. This is achieved by retrieving the output from the last n nodes in the last column where n is the number of required outputs. Also, the problem of 'bloat' normally seen in SGP (Blickle and Thiele, 1994) has been shown not to be a problem when using CGP (Miller, 2001). The main differences between SGP and CGP are summarised in Table 8.1.

Note that extensions to CGP have been presented by Miller, in particular the automatic acquisition of modules (Walker and Miller, 2007; Miller and Harding,

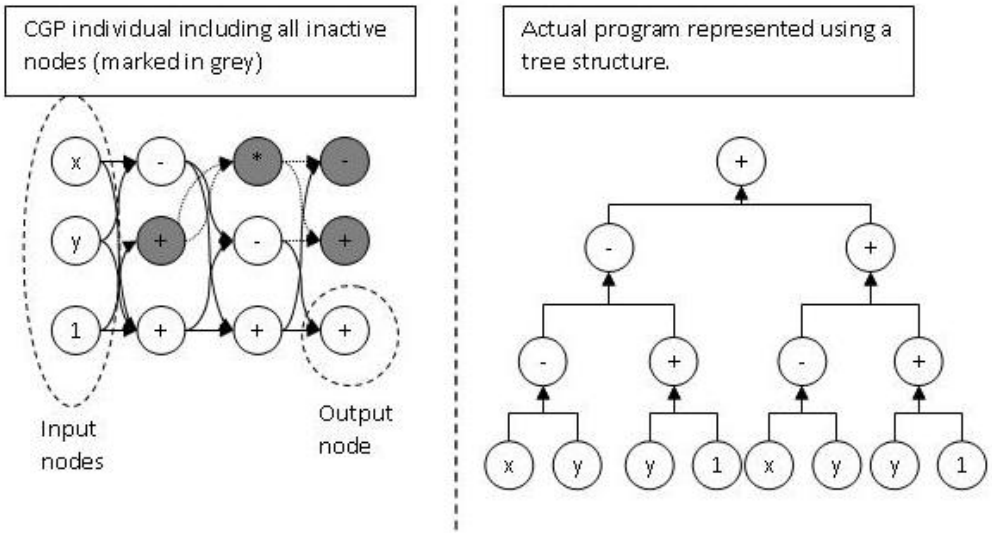


Figure 8.1: Shows a CGP individual (on the left) and the corresponding program tree on the right. Some nodes do not contribute to the actual program.

2008) a technique similar to that of automatically defined functions in SGP. However, the use of modules was not available in the implementation used for this work. This should not be of great importance as the main advantage of modules is that good solutions are found quicker, not that better solutions are found. Computational efficiency, while an important aspect, was not one of the key considerations in this work.

Table 8.1: The main differences between SGP and CGP

	SGP	CGP
Representation	Tree structure.	Directed graph organised in a grid structure (thereof the name Cartesian).
Size	Individuals may change size during evolution. This might in some cases lead to 'bloat'.	Individuals have a predefined size which does not change during evolution.
Crossover	Identical parents may yield new genetic material.	Identical parents will always yield identical children.
Neutrality	All nodes contribute to the output.	May contain 'inactive' nodes that do not currently contribute to the output. These may later be activated through evolutionary operations.
Outputs	A single output; the root node.	Any number of output nodes.

8.1.1 Structure of a CGP individual

Each individual consists of N nodes. N is, in standard CGP, chosen at the start of the evolution and is never altered by any operators. Each node consists of a function F and C inputs which each represents a connection to another node with the corresponding number. Each node can therefore be coded as a string consisting of node number, inputs and function. Figure 8.2 shows a possible encoding of a CGP program consisting of 6 nodes and two inputs organised in a grid layout with 3 nodes in each layer. As can be seen the numbering of the nodes and their inputs take into account the number of values in the input vector, though this is really a matter of implementation.

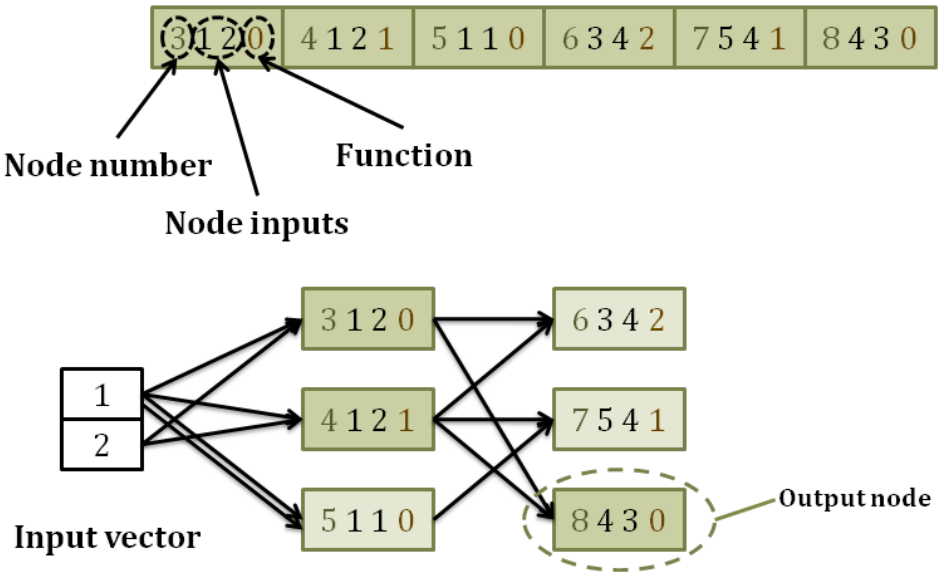


Figure 8.2: A possible encoding of a CGP individual and the corresponding graphical representation.

8.1.2 Evolutionary operators

8.1.2.1 Mutation

Mutation is an evolutionary operator where the structure of a single individual is altered. A mutation of a node is realised in one of two ways:

(i) by changing one of the inputs, i.e. connections to other nodes or input vector, so that it points to another randomly chosen and *legal* node.

(ii) by changing the function of the node by replacing it with a randomly chosen function taken from the available list of functions.

An example of mutation is shown in Figure 8.3 where the input of a node and the function of another node are mutated. Mutation is normally applied to each node in the individual with a probability set at the start of the experiment. A high mutation rate means that each node has a large chance of changing between each generation and the average fitness of the population will therefore fluctuate more. The advantage of having a large mutation rate is that potential solutions have less chance of being stuck at local minima in the fitness landscape. A small rate of mutation on the other hand is better when converging on the optimal solution and small changes are needed in order to achieve a better fitness. Some approaches will therefore start with a high mutation rate and lower it as the population converges on a solution.

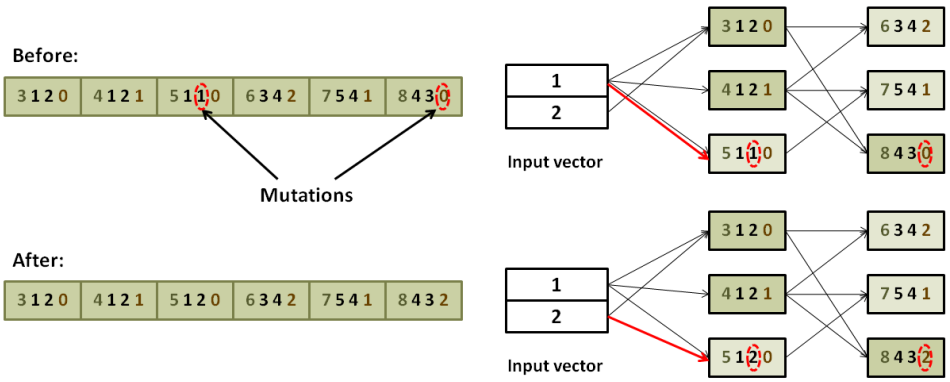


Figure 8.3: An example of mutation in CGP.

8.1.2.2 Crossover

Crossover is an evolutionary operator where two individuals are combined to produce one or more 'offspring'. First two individuals referred to as parents are chosen from the population using some selection strategy (see Section 8.1.2.4). The selection strategy is in most cases based on the fitness of the individuals. Crossover is then performed and the resulting individual(s) can be used in the next generation. The traditional crossover for CGP is described below:

Traditional crossover: Crossover in CGP is traditionally realised by choosing two points representing crossover limits in the chromosome of the individuals. These crossover points are the same for both individuals. All nodes between these limits are then swapped creating two new individuals. This crossover process is illustrated in Figure 8.4.

It is worth noting that crossover in CGP differs from SGP in that two identical parents will lead to identical offspring; this is generally considered to be a weakness of CGP.

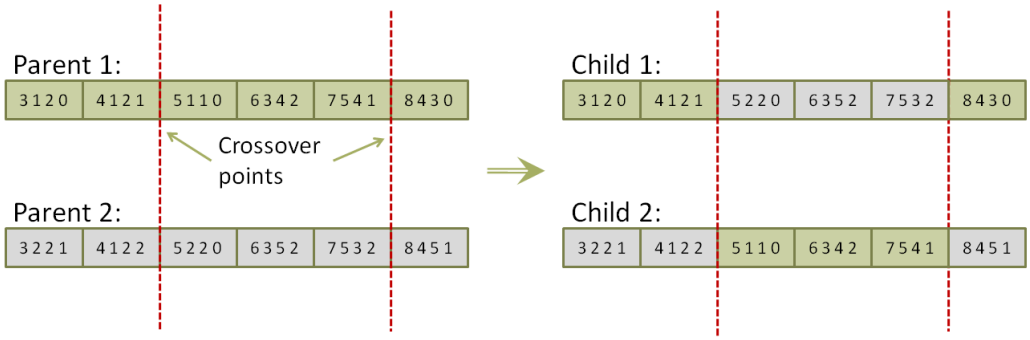


Figure 8.4: Traditional crossover in CGP.

In an attempt to improve upon the traditional crossover techniques, a new type of crossover has been developed by this author, called Graph crossover. Following is a description of this new crossover method before an empirical comparison with the traditional approach is performed.

Graph crossover: The idea behind the new crossover method is that functionality is coded in the 'sub-graphs' of the program. The traditional way of performing crossover will often copy nodes that have no connection with each other and are therefore not likely to contain any useful functionality. Instead, the aim is to copy sub-graphs of the program when performing crossover. The graph crossover is described in Algorithm 8.1. Using the algorithm a sub-graph of the individual is copied and thus it guarantees that the copied nodes are all connected, something which is not the case with the standard crossover approach. By always copying connected nodes this should increase the chance that some useful functionality is transferred when performing crossover. The full crossover process is illustrated in Figure 8.5.

Algorithm 8.1 Graph crossover

1: Given two Parents; P_1 and P_2.
2: Choose a node N_s at random in the individual P_1 and mark this for crossover.
3: Randomly choose a number j representing number of 'recursive' steps.
4: Traverse the sub graph down to a depth of j starting at N_s by following the inputs of each node recursively. Mark all nodes traversed in this way for crossover.
5: Copy all nodes marked for crossover into P_2 in the corresponding position (overwriting the existing nodes). This constitutes the new individual.

An experimental comparison between the new crossover approach and the standard one will be performed in the following section.

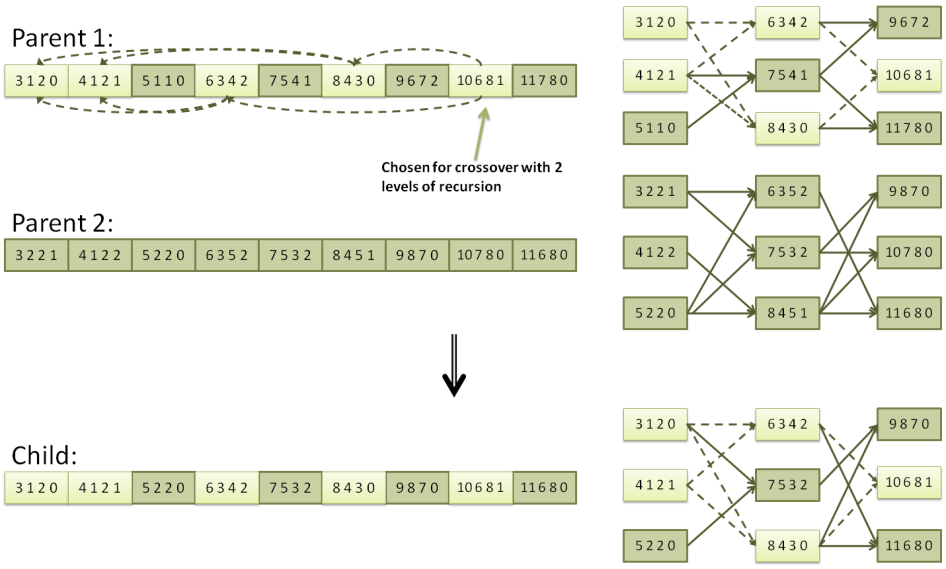


Figure 8.5: New crossover method called Graph crossover

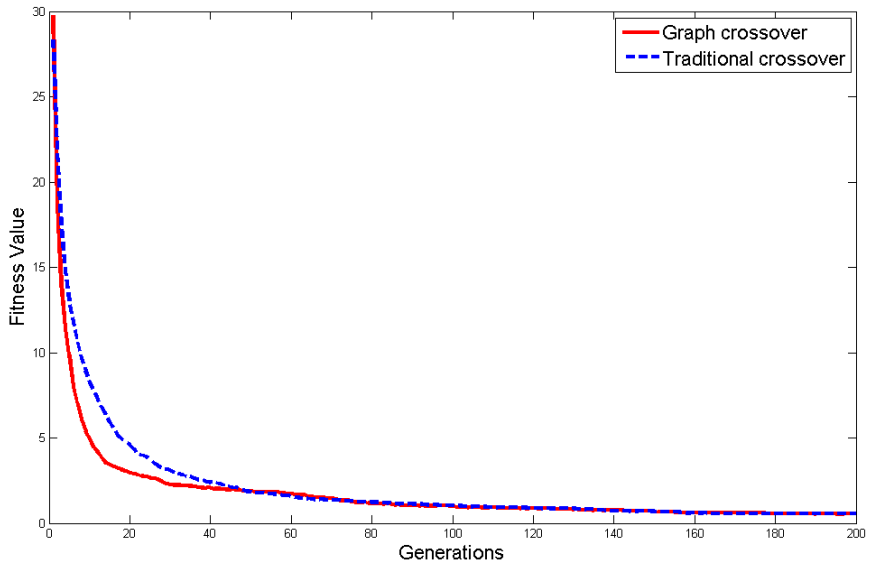
Comparison of crossover approaches: Both crossover approaches were used on 4 different problems, two symbolic regression problems and two even parity problems. These types of problems were chosen as they are relatively commonly investigated in literature. First, we will note the effect on the overall fitness of the population and the best individual for both the crossover approaches to see if there is an overall difference between the two. In addition we will look at the statistics of every crossover made throughout the experiment and note how many crossover operations lead to a better, a worse or an equally fit offspring. The CGP parameters used for the experimental runs are shown in Table 8.2.

Table 8.2: Parameters for the crossover tests

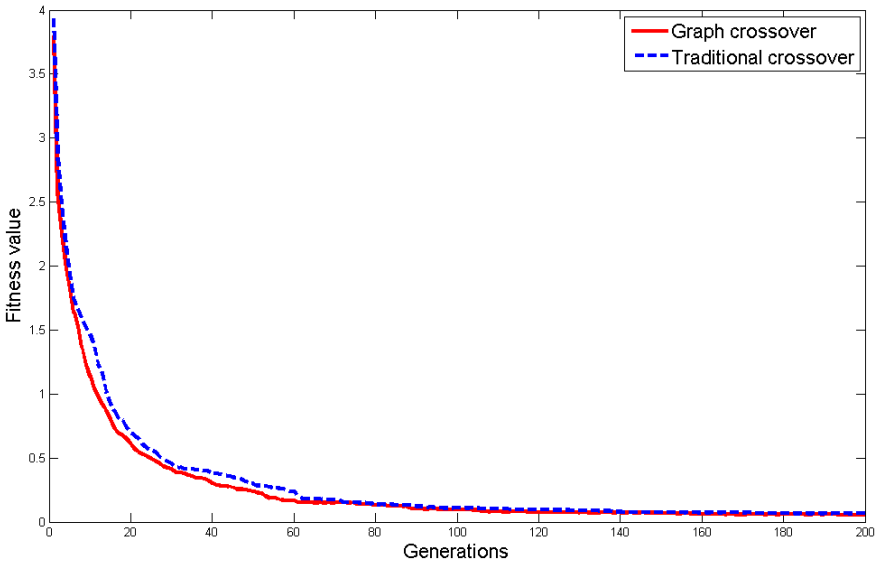
Parameters	Symbolic regression	Even parity
Individuals	50	50
Generations	200	500
Mutation probability	10%	10%
Crossover probability	80%	80%
Nodes	12	20
Backwards connections	8	12
Functions	$+, -, *, /$	AND, OR, NAND, NOR, NOT

Results: Figure 8.6 shows the result of CGP using the new crossover method versus that of the traditional one for two symbolic regression problems ($y = x^2 - 2x + 1$ and $y = x^6 - 2x^4 + x^2$) while in Figure 8.7 the performance for two even parity problems (3 and 4 bits) are compared. The graphs show the fitness value of the best individual for each generation averaged over 200 runs for the symbolic regression problems and 100 runs for the even parity problems. From the results it is evident that the new crossover method leads to better results compared to that of the traditional approach, in particular for the even parity problems. One can in general observe a faster convergence using the graph crossover and for the even parity problems it also achieves significantly better fitness of the best individual. The fact that graph crossover achieves so much better results for the even parity problems is likely to be due to the fact that such problems rely more on developed sub-graphs which are more likely to be transferred using the new crossover approach.

Table 8.3 shows the percentage of crossovers which lead to a better, worse or equally fit offspring for each of the test problems. One can see that the new graph crossover more often leads to a better individual, however for the symbolic regression problems the difference is not very large; e.g. 3.48% vs 3.16% for the $y = x^2 - 2x + 1$ problem. Still, if one considers the number of crossovers leading to worse fitness the difference is greater with 2.54% vs 8.23%. However one should note that the lower rate of crossovers leading to worse individuals may not always be beneficial. A very low value could mean that the crossover is not leading to significantly different individuals, meaning that it could more easily become stuck at a local minimum. Thus, while the standard crossover technique may lead to less fit individuals, it may indeed be exploring a broader area of the fitness landscape as compared to the graph crossover and may therefore perform better for other problems. This is however only a theory and a full investigation is not covered in this thesis, but is recommended for future research.

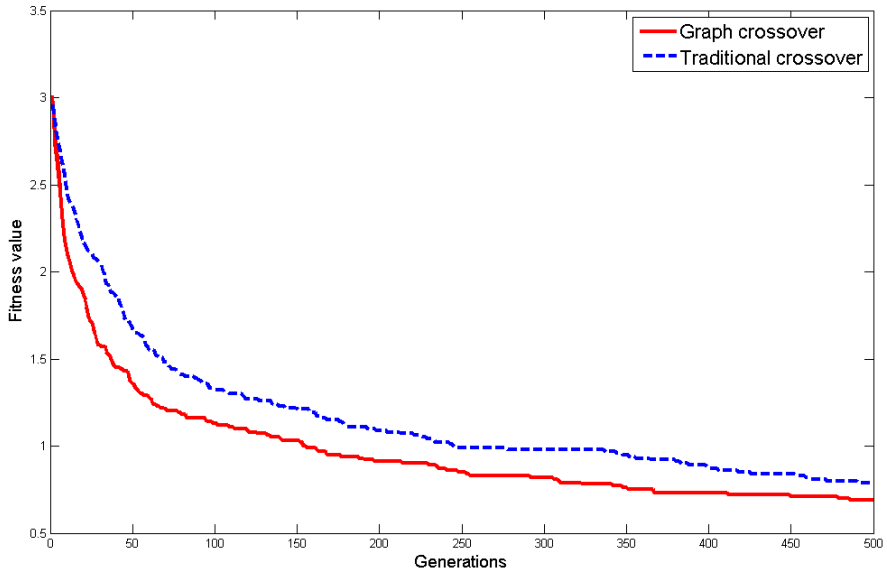


(a) $x^2 - 2x + 1$

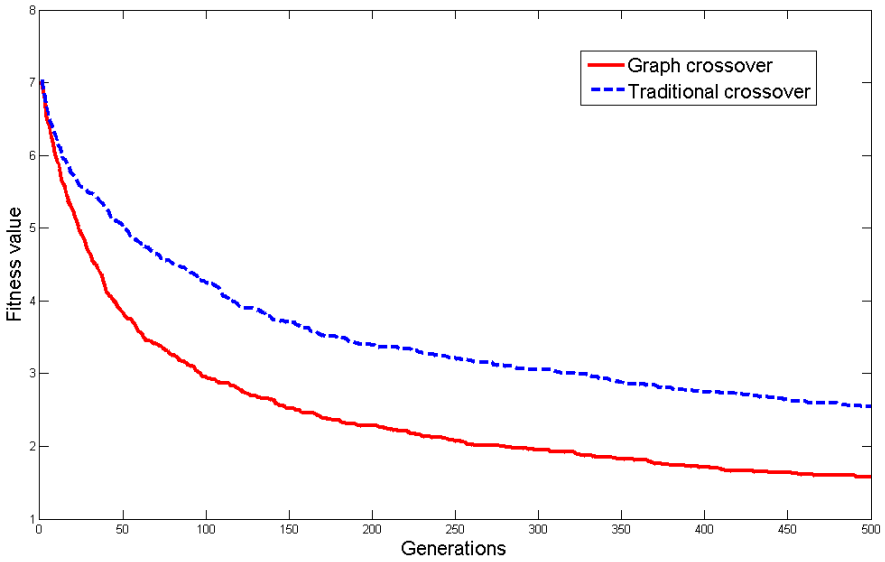


(b) $x^6 - 2x^4 + x^2$

Figure 8.6: The performance of the new crossover type. The graphs show the best individual for each generation averaged over 200 runs. a) shows the results for the symbolic regression problem of $x^2 - 2x + 1$ and in b) the results for the problem $x^6 - 2x^4 + x^2$ are shown.



(a) Even parity 3 bits



(b) Even parity 4 bits

Figure 8.7: The performance of the new crossover type. The graphs show the best individual for each generation averaged over 100 runs. Figures a) shows the results for the 3 bit even parity problem and in b) the results for the 4 bit even parity problem are shown.

Table 8.3: Effect of crossover operations on the offspring fitness

Problem	Crossover type	Better fitness	Worse fitness	Equal fitness
$y = x^2 - 2x + 1$	Graph	3.48%	2.54%	93.98%
	Regular	3.16%	8.23%	88.61%
$y = x^6 - 2x^4 + x^2$	Graph	9.95%	1.38%	88.68%
	Regular	7.58%	15.40%	77.02%
Even parity: 3 bit	Graph	13.92%	2.51%	83.57%
	Regular	5.62%	10.90%	83.48%
Even parity: 4 bit	Graph	24.36%	2.58%	73.16%
	Regular	10.11%	22.12%	67.77%

8.1.2.3 Reproduction and elitism

An important form of evolutionary operator is reproduction. In this operation an individual is selected from the existing population and copied into the new population. Depending on the selection strategy used (see Section 8.1.2.4) the individuals are chosen for reproduction based on their fitness value. This means that 'good' genes are more likely to be transferred to subsequent generations.

If a very high reproduction rate is used the evolution is often referred to as a 'steady-state' evolution as there will only be small fluctuations in the average fitness of the population.

A special case of the reproduction approach is called *elitism* where the fittest individual is *always* transferred without changes to the next generation. By doing this one guarantees that the best solution thus far is never lost or altered by the other evolutionary operators. However, elitism can cause the population to be dominated by a single very fit individual and may therefore converge prematurely.

8.1.2.4 Selection strategies

In order to select individuals for the evolutionary operations some strategy must be employed. There exists a number of different strategies with the commonality

that they are (almost) all dependent on the fitness of the individuals. Following is a description of the selection strategies available in the implementation of CGP used here. These strategies have been implemented as they are some of the most common forms of selection strategies used in the literature.

Tournament selection: In tournament selection a number of individuals are selected at random and they then 'compete' in a tournament where the best fit individual is considered the winner or has a greater probability of winning. This continues until a single individual is left as the winner. The winner of the tournament is then used for evolutionary operations. The overall selection pressure can then easily be adjusted by changing the size of the tournament. A higher number of individuals taking part in the tournament means that a less fit individual has a smaller chance of being selected. If a tournament size of 1 is used the selection procedure is equivalent to random selection.

Roulette selection: In roulette selection the individuals are selected based on their relative fitness. That is each individual is assigned a piece of a 'cake/roulettewheel' corresponding to their fitness. An individual is then selected randomly with a probability corresponding to its share of the wheel. As an example consider a population of 13 individuals with fitness given by

$$Fitness = [10, 4, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2]$$

where a higher fitness is better. The corresponding probabilities using roulette selection is shown in Figure 8.8. One can observe that a highly fit individual will potentially have a very high chance of being selected. Care must therefore be taken when assigning fitness, such that the values are properly scaled to avoid the problem where a single or a few individuals dominate the selection procedure.

Rank selection: Rank selection is similar to roulette selection, but instead of assigning probabilities based on the actual fitness of the individuals the probabilities are assigned based on the relative rank of an individual as compared to the rest of the

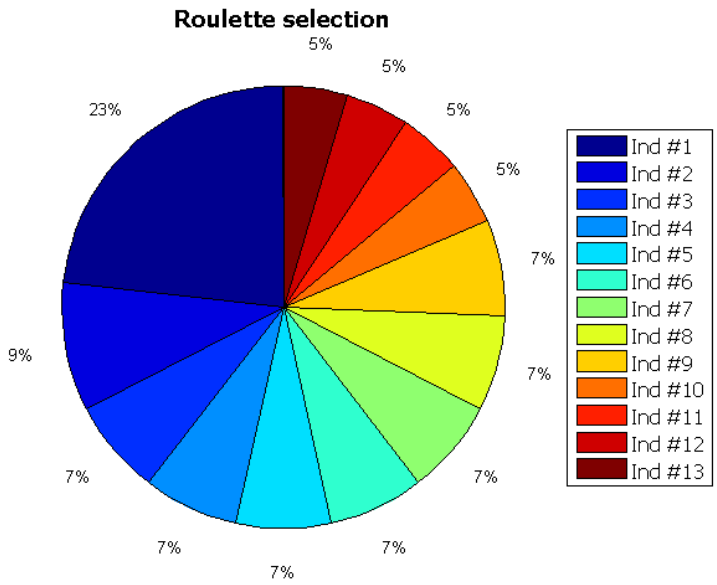


Figure 8.8: Selection probabilities for an example population using roulette selection.

population. The ranking is determined based on the fitness of the individuals and each individual is therefore assigned a rank ranging from 1 to n where n is the number of individuals in the population. If one ranks the individuals in ascending order (i.e. worst first) the probability of being selected is then given by $P_{select} = \frac{i}{n*(n+1)/2}$ where i is the rank of the individual. Using the same example population as in the previous section one gets selection probabilities as shown in Figure 8.9. Rank selection ensures that a single very fit individual does not dominate the selection procedure as might be the case when using roulette selection. Indeed the selection probability is independent of the absolute fitness value, but rather depends on the relative fitness value of an individual as compared to the rest of the population. However the least fit individuals have an increasingly small chance of being selected as the population size increases, even though their fitness may be very close to better ranked individuals.

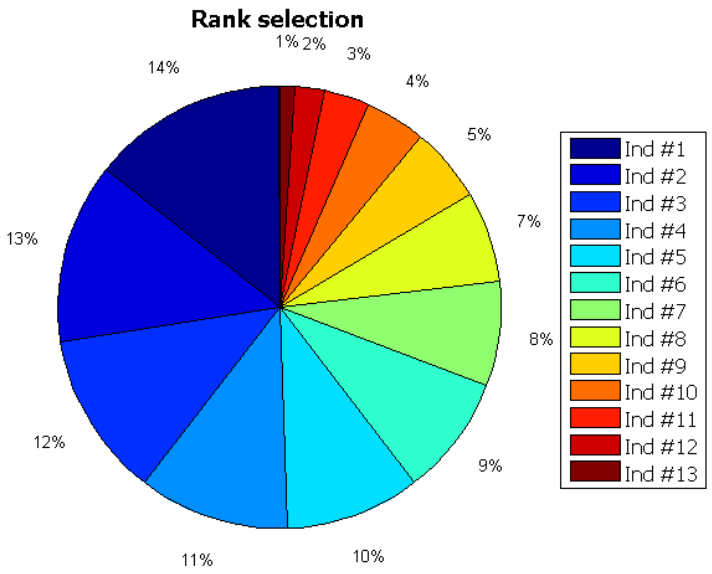


Figure 8.9: Selection probabilities for an example population using rank selection.

8.1.3 Evolution

CGP evolution is realised in the same way as for GA (see Section 6.2.3.1) or SGP. There are many variations to the basic algorithm which may be used though the core concepts are usually kept identical. The algorithm used in this work is outlined in Algorithm 8.2. This variant of the algorithm is commonly used in the literature and was therefore chosen for this work. Further investigation of the impact of the evolution algorithm is left for future work.

8.1.4 Hierarchical fair competition

Hierarchical fair competition (HFC) is a relatively new model for evolutionary computation introduced by Hu and Goodman in 2002 (Hu and Goodman, 12-17 May 2002; Hu et al., 2005). The idea behind the HFC model is that the traditional evolutionary model is susceptible to premature convergence, and will often become stuck (or at least delayed) at a local minimum. Convergence is of course key to evolutionary models in general, but should be controlled in order to avoid the possibility

Algorithm 8.2 CGP evolution

```
1: Create initial population
2: Evaluate fitness of each individual
3: while End criteria is reached (predefined fitness or max generations) do
4:    while New population smaller than old do
5:       Choose individual according to selection strategy
6:       Apply evolutionary operators
7:       Insert into new population
8:    end while
9:    New population replaces the old one
10:   Evaluate fitness of new population
11: end while
```

that solutions get stuck at a local minima. In the traditional approach a high fitness individuals may to a large degree take the place of lower ranked individuals and may dominate the evolutionary process. The idea behind HFC is to make the competition *fairer* in the sense that similarly ranked individuals should compete only with each other. This is achieved by sub-dividing the population into several classes, usually referred to as *demes*. The individuals are assigned to a deme according to their fitness ranking and each deme has an associated admission threshold. The selection and application of evolutionary operators are then constrained to a single deme so that individuals only compete with individuals within the same deme.

If an individual through evolution achieves a higher fitness it is moved to a corresponding deme if it passes that deme's admission threshold. The HFC concept is illustrated in Figure 8.10. Note that individuals are never moved to a lower level deme, hence the migration between demes are *unidirectional*. It is therefore clear that the entry level deme (lowest fitness) can send individuals to any other deme, while the highest ranked deme can only receive individuals. If a deme receives more individuals than it exports, the lowest ranking individuals are replaced. Similarly, if a deme is lacking individuals after a migration, new ones are created randomly to replace them. New individuals are therefore continuously created and supply the

population with new genetic material.

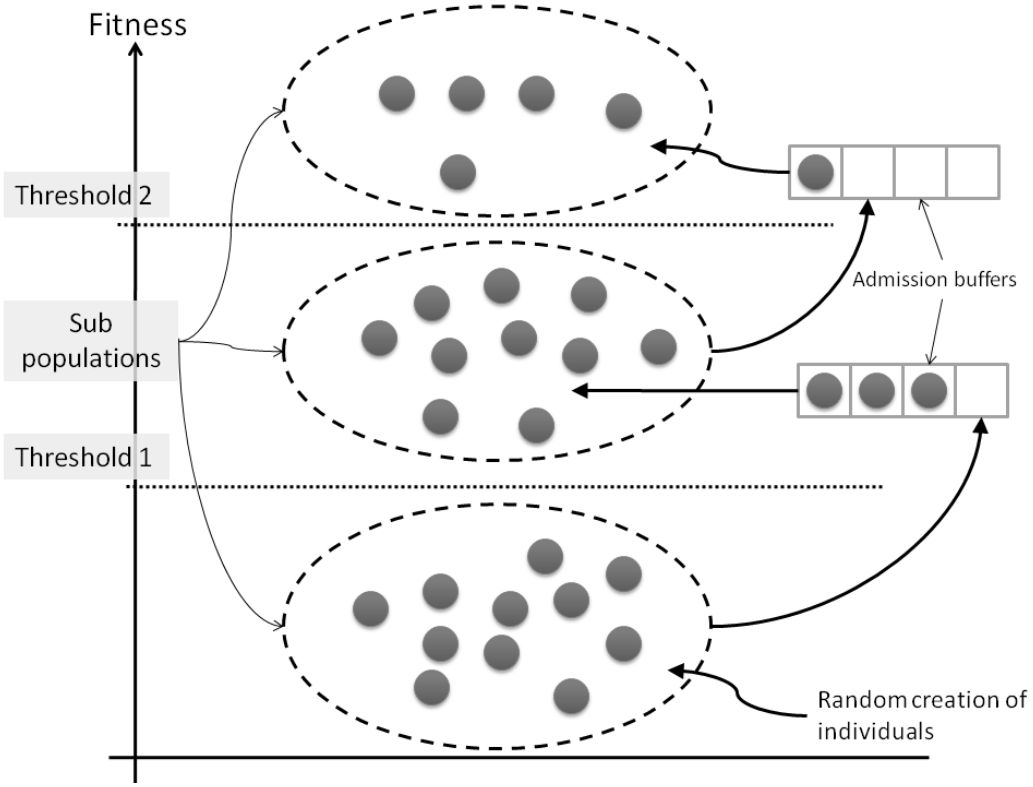


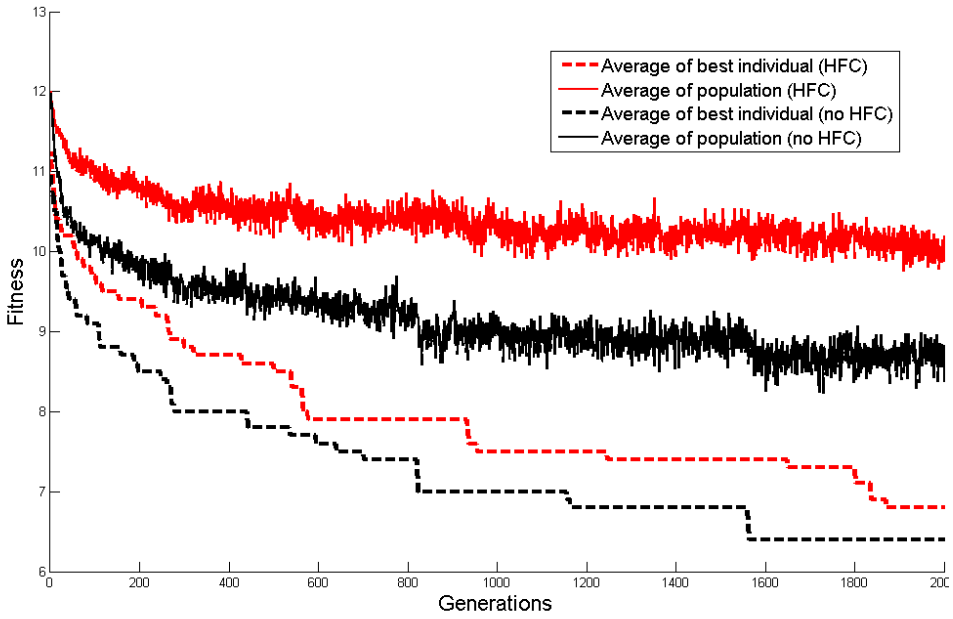
Figure 8.10: An illustration of the HFC evolutionary model. Adapted from (Hu et al., 2005).

Note that it is not a requirement that migration is performed at every generation step. Indeed, in the implementation used in this work (see Appendix B) it is a configurable parameters which can be tuned by the operator. This provides a tradeoff in terms of computational costs and may indeed be beneficial as each deme is allowed to 'mature' and spread new genetic material within the deme before passing on new individuals to higher demes.

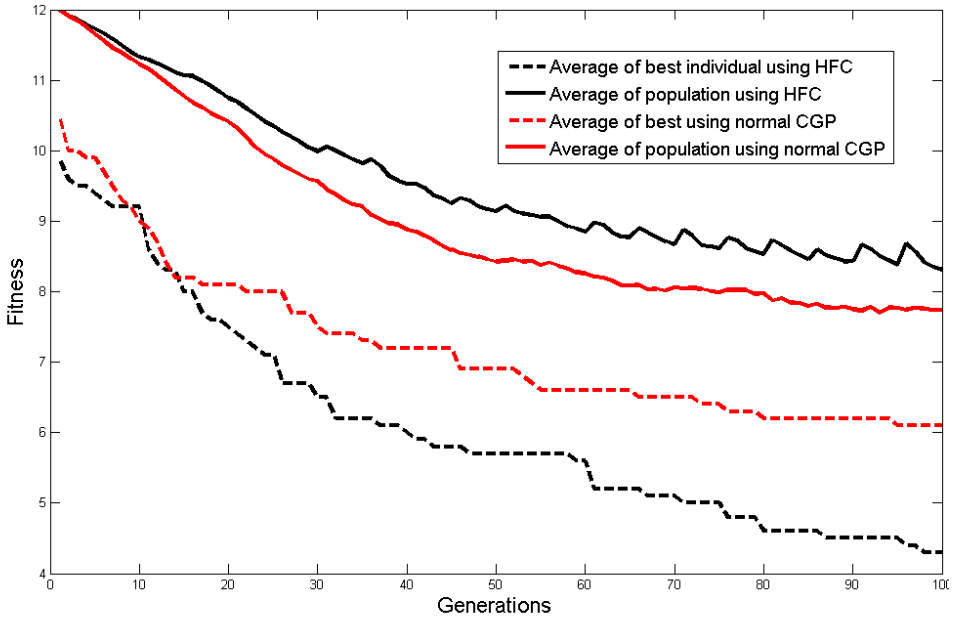
8.1.4.1 Empirical tests of HFC in conjunction with CGP

A number of tests were performed in order to evaluate the effectiveness of HFC in conjunction with CGP. HFC was first tested on an even parity problem using 5 bits/inputs. The initial tests were performed using 32 individuals over 2000 generations and the results where averaged over all runs. The population size was set to a low value in order to conduct a large number of tests. The generation value was chosen to be relatively large in order to allow the population to converge, though the exact value is not likely to impact such a direct comparison.

The initial tests showed that HFC seemingly has a negative impact on the results. The reason for this discrepancy is likely to be due to the fact that the *deme* sizes are too small when using only 32 individuals and therefore does not contain enough genetic material to work properly. The tests were therefore re-run using a larger population size of 1024 individuals for 100 generations. In this case HFC had a positive effect on the overall results leading to a faster convergence to the best individual. However, the population average converges somewhat slower which is due to the fact that new individuals are continually inserted into the population (to replace individuals leaving the lowest deme). This shows that one must take care to choose the parameters when using HFC in order to get the best possible effect. The result of the two tests are shown in Figure 8.11 while the parameters used are shown in Table 8.4. Further study on the impact of HFC for different types of problems is left for future work, though one can conclude that it has the potential of improving upon the regular evolutionary model at least for some types of problems, though the impact has at least some correlation to other parameters such as population size.



(a) Population size: 32



(b) Population size: 1024

Figure 8.11: The performance obtained using HFC vs the regular CGP model. The graph shows the fitness of the best individual and of the population averaged over all runs.

Table 8.4: Parameters for HFC tests

Parameters	Even parity 5 bits - I	Even parity 5 bits - II
Individuals	32	1024
Generations	2000	100
Mutation probability	5%	5%
Crossover probability	80%	80%
Nodes	20	20
Backwards connections	12	12
Functions	AND, OR, NAND, NOR, NOT	AND, OR, NAND, NOR, NOT

8.1.5 Determining CGP parameters

As for other evolutionary approach the determination of the available parameters is important and can greatly affect how effective the CGP is. In Section 6.2.3.2 this problem was investigated in the context of GAs, though most of the arguments are general for all evolutionary algorithms. Importantly, it has been shown that finding a 'window' of good parameters also applies for genetic programming approaches (Piszcz and Soule, 2007). Since the parameters are dependent and interact in a non-linear manner an exhaustive search of parameters may be too computationally expensive in many cases.

It is also likely that changes to the test set used to evolve the programs will affect the optimal set of parameters. In the case of the TDAQ data the work described in this thesis is a first attempt to apply evolutionary algorithms in order to automatically evolve error detection programs using this *type* of data. However, as the techniques are further explored the data used to evolve these programs may change. This can be either due to a better understanding of the system and its interactions allowing one to remove or add different parts of the dataset, or it could be due to the TDAQ system itself changing and thus producing different datasets. Based on this, and due to the computational cost, the optimisation of CGP parameters were therefore left for future work. Instead values similar to those found in the literature have been used throughout this chapter unless stated otherwise. Table 8.5 shows the range of parameters used based on (Miller and Thomson, 2003; Walker and Miller, 2005; Miller and Thomson, 2000; Clegg et al., 2007).

Table 8.5: Typical CGP parameters as found in the literature.

Parameter	Value
Population size	$[5-]$
Crossover probability	$[0.6 - 0.95]$
Mutation probability	$[0.02 - 0.10]$
Number of generations	$[1000 - 100000]$

8.2 Applying CGP for error detection in the TDAQ system

In this section CGP is applied to automatically evolve programs to detect and classify the different errors present in the data. First the fitness function used to evaluate the different solutions is presented, before CGP is applied using IS and ERS data in turn.

8.2.1 Fitness function

A crucial part of utilising CGP or any related evolutionary model is to determine the fitness function. An inappropriate choice of fitness function may negatively affect the evolution or indeed cause the process to fail. For the error detection problem the initial approach was to use the number of misclassified samples (i.e. the error rate) as shown in (8.1). However, if the datasets contain a large majority of positive or negative samples, such a classification can be misleading. For example, if 95% of the data points are negative samples, then a function which is always returning 0 would classify 95% correct values and will be considered a highly fit individual, something that is not the case. It might even outperform other classifiers that are actually correctly classifying some of the positive samples as well.

In order to avoid these problems a fitness function based on the measures introduced in Chapter 4, namely Sensitivity, Specificity, PPV and NPV was chosen. A similar function was used in Chapter 6 and 7 in order to compare the different ANNs and SVMs, though some small modifications were made. As the problem is defined as a binary classification problem all the outputs were, for the purpose of calculating fitness, modified so that any output ≥ 0.5 was considered a positive output while any output < 0.5 was considered negative. TP, TN, FP and FN can then be calculated and the fitness function is given using the product of Sensitivity, Specificity, PPV and NPV shown in (8.2). This is achieved by modifying PPV and

NPV such that:

$$PPV = \begin{cases} 0 & , TP = 0 \\ \frac{TP}{(TP+FP)} & , TP > 0 \end{cases}$$

and

$$NPV = \begin{cases} 0 & , TN = 0 \\ \frac{TN}{(TN+FN)} & , TN > 0 \end{cases}$$

Hence the fitness will always be in the range $[0, 1]$ with an optimal fitness of 0; i.e. the goal is to minimise the fitness function. This ensures that the evolved program must correctly classifies at least one negative and one positive sample to achieve anything but the worst possible fitness.

$$Fit = (FP + FN) \tag{8.1}$$

$$Fit = 1 - Sensitivity * Specificity * PPV * NPV \tag{8.2}$$

8.2.2 IS data

In this section CGP is applied in order to evolve an error detection program based on the IS dataset. Two different approaches were explored. First, separate programs were evolved for each of the error types. Hence, three different programs were evolved each predicting the different error types based on the same input data. As CGP supports multiple outputs (as compared to SGP which supports only a single output) the evolution of a *single* program in order to detect all types of errors in the system was then explored. This program has 3 output values each representing the prediction of an error type.

8.2.2.1 Data preparation

The IS dataset as used consists of 974 observations with 25 variables/dimensions. The high number of dimensions in the input data could make the evolution of a program more difficult. PCA was therefore applied in order to reduce the number

of dimensions. However, as can be seen from Figure 8.12 the first 10 principal components explain only approximately 50% of the variance in the data. Even the first 20 principal components explain less than 90% of the variance. It is therefore likely that any attempt of reducing the number of dimensions could lead to a significant loss of variance and potentially a loss information that could be important to the error detection. Based on this the PCA results were not used and the dataset was left as is using all 25 dimensions. One therefore has to rely on the evolutionary process to select the appropriate inputs and discover any relationships between them.

White noise was not added to the data in these tests in order to keep the computational costs within reason. Hence, the data set is not identical to the one used in Chapters 6 and 7. However, in Table 6.3 in Chapter 6 the results of applications of ANN to the exact same dataset is presented and will therefore provide a more accurate reference when comparing the results.

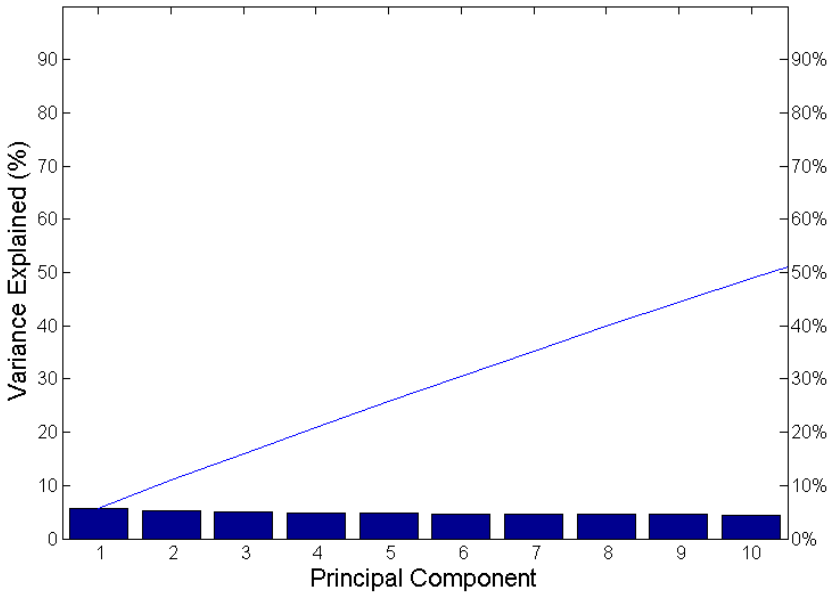


Figure 8.12: Variance explained by the first 10 principal components of the IS dataset.

8.2.2.2 Choosing functions

All functions defined in Table 8.6 were available during the evolution of the programs. Hence, any node in the potential solutions might represent any one of these functions. Note that some of the functions accept a different number of parameters, so that if a function changes due to mutation this might also affect the inputs to that particular node.

No selection of functions was made prior to the tests as there may be different optimal sets of functions for each of the error types (TypeI, TypeII and TypeIII). Such a selection therefore becomes complex and time consuming, and is left for future research; see Section 9.2.

Table 8.6: Functions available while evolving the CGP individuals.

Terminal	Inputs	Description/output
+	a,b	Addition $y = (a + b)$
-	a,b	Subtraction $y = (a - b)$
*	a,b	Multiplication $y = (a * b)$
/	a,b	Division $y = \frac{a}{b}$
OR	a,b	logical or $y = (a \vee b)$
AND	a,b	logical and $y = (a \wedge b)$
NAND	a,b	logical not-and $y = \neg(a \wedge b)$
NOR	a,b	logical not-or $y = \neg(a \vee b)$
XOR	a,b	logical exclusive or $y = (a \oplus b)$
NOT	a	logical not $y = \neg a$
gt	a,b	Greater than: returns 1 if $(a > b)$, otherwise 0.
lt	a,b	Less than: returns 1 if $(a < b)$, otherwise 0.
lif	a	Logical if: returns 1 if $a > 0$, otherwise 0.
sif	a,b	Simple if: returns b if $a > 0$, otherwise returns 0.
sign	a	Sign: returns $sign(a)$.
min	a,b	minimum of a and b.
max	a,b	maximum of a and b.
double	a	double of a $(a * 2)$
Constant		Returns a constant number between 0 and 1.

8.2.2.3 Multiple program approach

In this approach a separate program was evolved for each of the three error types; TypeI, TypeII and TypeII. 20 runs were conducted for each of the three error types. The normal evolutionary process was chosen due to two main factors:

1. The result of using HFC for the ERS dataset did not conclusively show a positive effect.

2. The normal evolutionary process requires less computational effort.

None of the CGP parameters were changed between any of the runs. The full list of parameters that were used are listed in Table 8.7.

Table 8.7: Parameters for the multiple program approach using IS data

Parameters	Value
Evolutionary model	Normal
Individuals	20
Generations	1000
Mutation probability	10%
Crossover probability	90%
Layout	Directed graph
Rows * columns	50
Backwards connections	49
Selection strategy	Rank

The overall results for the three different error types are all summarised in Table 8.8. The specific results for each of the error types, including the best solutions achieved for each type, are discussed in more detail in the next sections.

Table 8.8: Results of CGP using multiple program approach using the IS data

Network type	Error Type	Solution	Dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
CGP (Normal)	Type I	Best Ind	Training	98.29	7	3	99.45	80.00	90.32	98.74
			Test	97.94	3	5	98.65	84.21	76.19	99.18
		Average	Training	97.97	4.3	7.6	98.62	87.71	82.18	99.22
			Test	96.61	3.2	10	97.30	83.16	63.84	99.12
	Type II	Best Ind	Training	99.15	4	1	99.83	69.23	90.00	99.30
			Test	98.46	2	4	98.95	71.43	55.56	99.47
		Average	Training	98.17	6	4.7	99.18	53.85	75.07	98.95
			Test	97.22	5.1	5.7	98.51	27.14	26.84	98.66
	Type III	Best Ind	Training	98.29	10	0	100	47.37	100	98.26
			Test	99.20	2	5	98.69	66.67	44.44	99.47
		Average	Training	97.30	7.6	8.2	98.56	59.77	72.32	98.66
			Test	96.06	4	11.4	97.03	33.93	18.84	98.95

TypeI error

The average best fitness at each generational step is shown in Figure 8.13 together with ± 1 standard deviation. This shows that CGP is able to consistently produce good solutions relative to the particular problem. The programs are also relatively simple with an average length of 8.8 nodes. From Table 8.8 one can see that the best individual achieves a classification accuracy of 98.29% and 97.94% for the training and test sets respectively. The CGP evolved program achieves a performance rating of 61.48. If one compares this to the results listed in Table 6.3 one can see that the results match the best performances of ANNs for the same dataset (without noise).

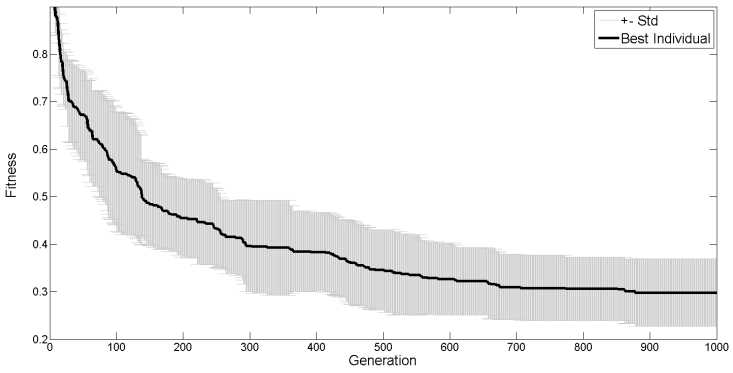


Figure 8.13: Fitness of the best individual at each generational step averaged over all runs for the TypeI error type. The gray area indicates ± 1 standard deviation as a measure of how consistently CGP finds a good solution.

The best solution is visualised in Figure 8.14. The program is very simple consisting of only 11 nodes out of 50 and utilising only 8 of the 25 available inputs. This indicates that there are enough nodes to effectively solve the problem at hand. Indeed one could experiment with a smaller number of nodes to possibly reduce the time it takes to converge on a good solution though this is left for future work. Another observation is that there are 6 'levels' of nodes in the program, here meaning that the longest path of nodes from output to input consists of 6 nodes. This information could be useful if using the grid model for the CGP individuals in order to ensure that sufficient columns are available in the grid.

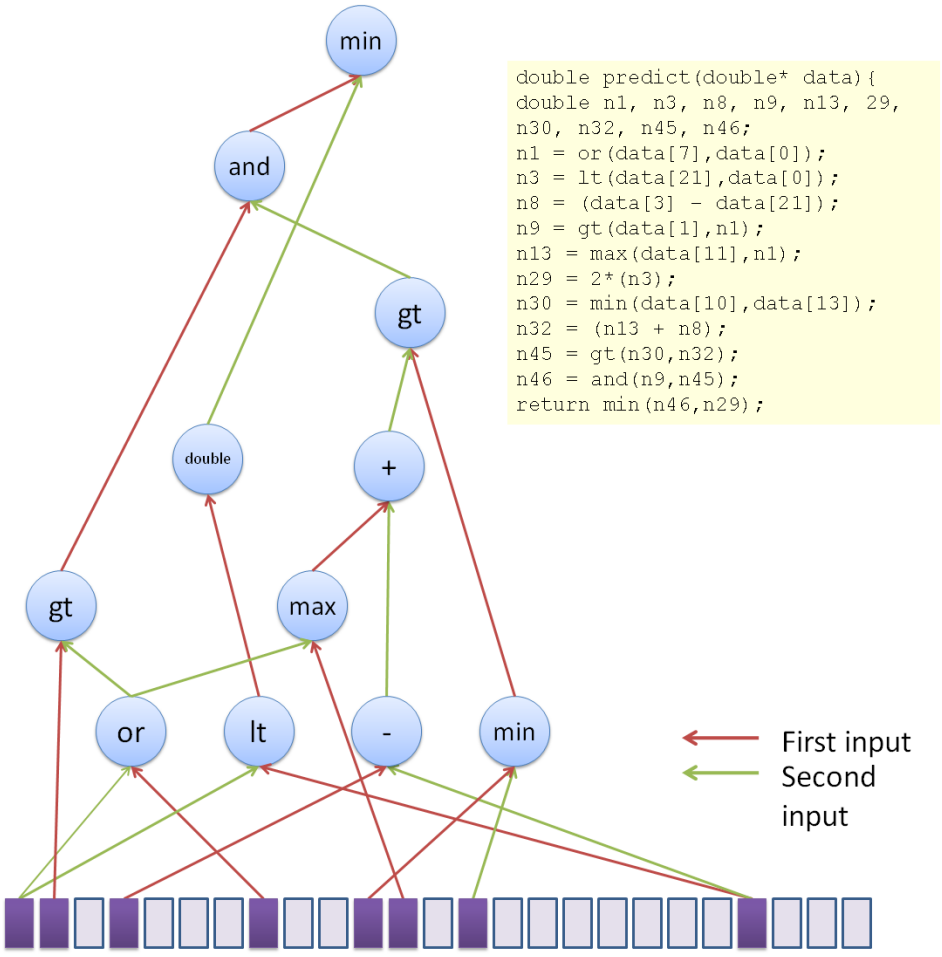


Figure 8.14: Best program found for the TypeI error. The program utilises only 8 of the 25 available inputs and consists of 11 nodes.

TypeII error

Figure 8.15 shows the fitness of the best individual averaged over all runs together with ± 1 standard deviation. One can observe that the results are less consistent than for the TypeI error. The best individual achieved a classification accuracy of 99.15% and 98.46% for the training and test set respectively. The overall performance rating is at 38.46 also matching the performance of ANNs for that same dataset.

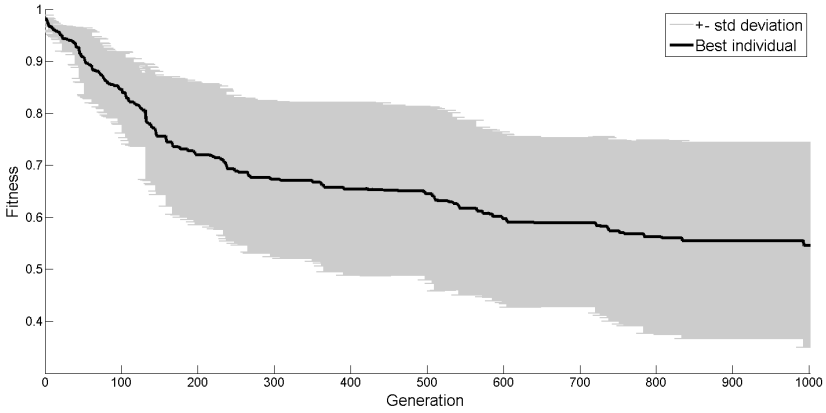


Figure 8.15: Fitness of the best individual at each generational step averaged over all runs for the TypeII error. The gray area indicates ± 1 standard deviation as a measure of how consistently CGP finds a good solution.

The best program is visualised in Figure 8.16 and one can observe that only 9 of the 25 inputs are being utilised. The program consists of 12 nodes out of the maximum 50, hence 38 nodes are unused in the program. This again indicates that a sufficient number of nodes were available during the evolution of the program. The longest path is between output and input is again 6.

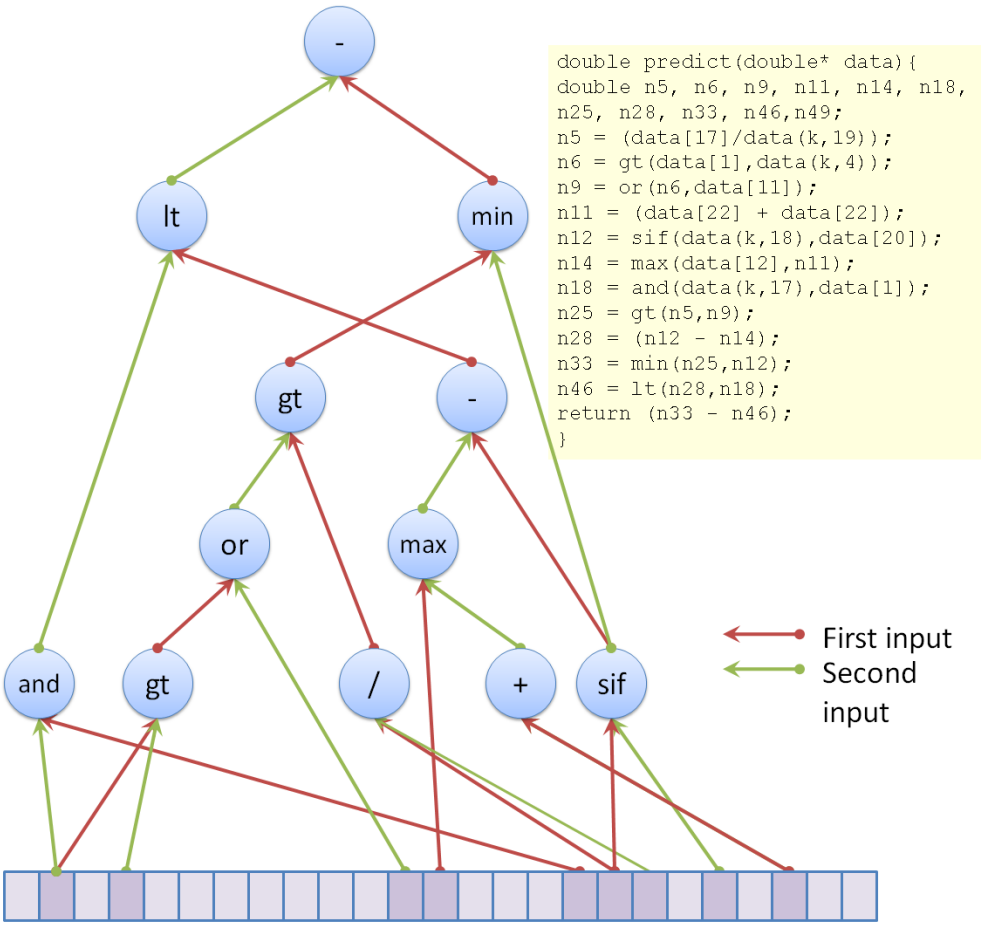


Figure 8.16: Best program found for the TypeII error.

TypeIII error

Figure 8.17 shows the fitness of the best individual averaged over all runs together with ± 1 standard deviation. The results are more consistent than for the TypeII error, though still not as good as for TypeI. This is to be expected, especially considering the fact that the clusters found in Chapter 5 were more prominent for the TypeI error than for the other two error classes. The best individual achieved a classification accuracy of 98.29% and 99.20% for the training and test set respectively. Though it is worth noting that the Sensitivity is quite low for both training and test sets with values of 47.37% and 66.67% respectively. This means that many positive cases may go unnoticed. This is reflected in the overall performance for the test set which has a score of 28.85. This is lower than all the ANNs for the same dataset.

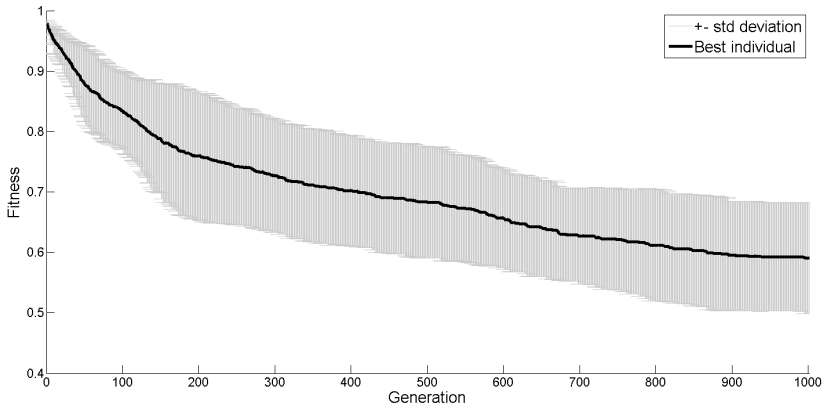


Figure 8.17: Fitness of the best individual at each generational step averaged over all runs for the TypeIII error. The gray area indicates ± 1 standard deviation as a measure of how consistently CGP finds a good solution.

The best program is visualised in Figure 8.18 and one can observe that in this case 12 of the 25 inputs are being utilised and the program consists of 17 nodes, still well within the maximum of 50.

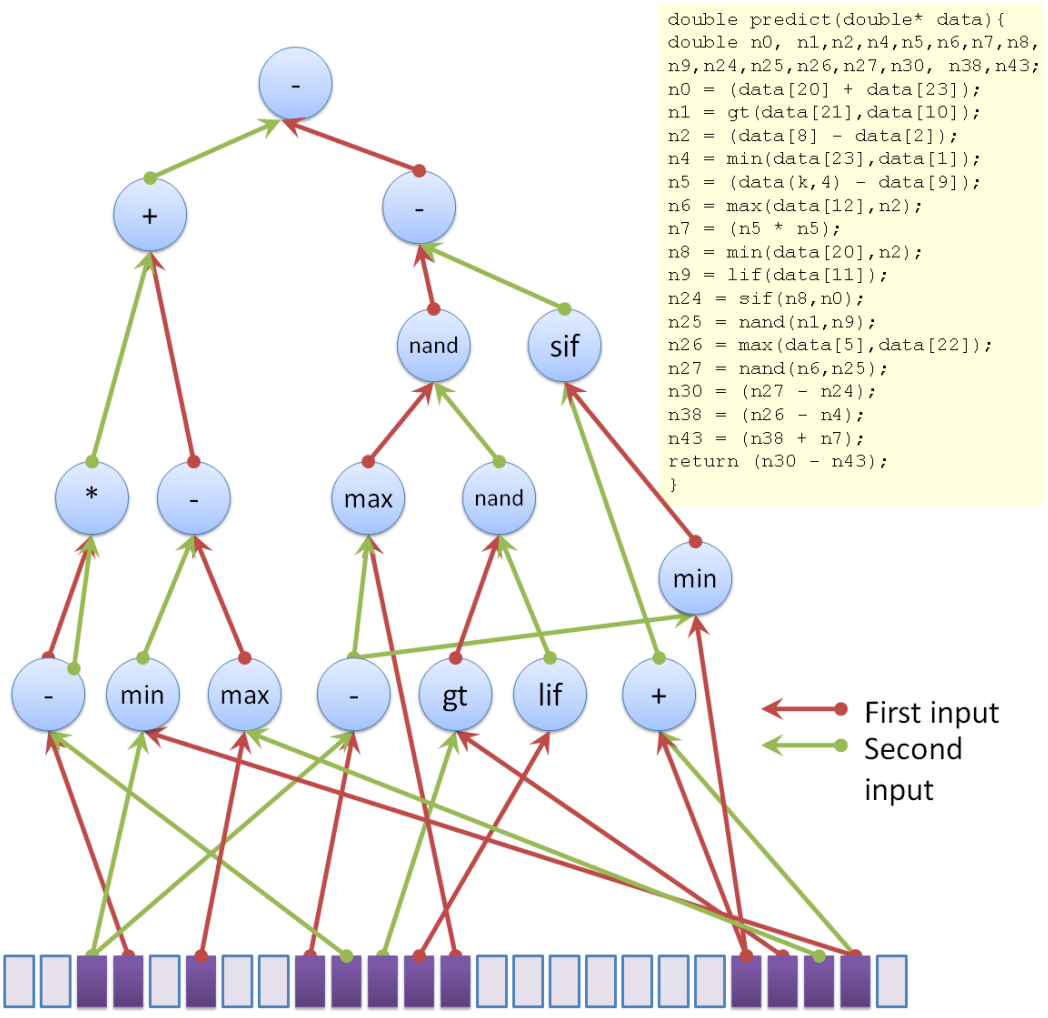


Figure 8.18: Best program found for the TypeIII error.

8.2.2.4 Single program approach

Clearly evolving a single program to predict all types of errors is a more difficult problem. The fitness function was modified so that Sensitivity, Specificity, PPV and NPV was calculated for each error type and then multiplied. Hence, the fitness function was:

$$Fit = 1 - Sensitivity_{TypeI} * Sensitivity_{TypeII} * Sensitivity_{TypeII} * \cdots * NPV_{TypeIII}$$

As was discovered using the multiple program approach in the previous section 50 nodes were more than sufficient to evolve good solutions to the problem. Indeed most solutions used around 10 nodes for each of the problems. The single program approach was therefore attempted using 60 nodes which should provide a healthy margin and allow for neutrality to exist within an individual. The parameters used in this experiment are summarised in Table 8.9.

Table 8.9: Parameters for the single program approach using IS data

Parameters	Value
Evolutionary model	HFC
Individuals	100
Generations	5000
Mutation probability	10%
Crossover probability	90%
Layout	Directed graph
Rows * columns	60
Backwards connections	59
Selection strategy	Rank

Results: 3 runs were made using the parameters shown in Table 8.9. The results of the best individual is summarised in Table 8.10. Overall the results are similar to those achieved using a single program per error type. The results range from 95.12% for the TypeIII test set to 97.69% for the TypeII test set, showing that a good generalisation performance was achieved. However, the solution is not really

satisfactory because for example in the case of the TypeIII error the PPV is only 15.79% due to a relatively large numbers of false positives. Even so, in overall terms the results show that it is indeed possible to evolve a single program to detect all three types of errors based on the IS data.

Table 8.10: Results of CGP using the single program approach.

Network type	Error Type	Dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
CGP (Normal)	Type I	Training	97.94	7	5	99.09	80.00	84.85	98.73
		Test	97.94	3	5	98.65	84.21	76.19	99.18
	Type II	Training	99.48	3	0	100	76.92	100	99.48
		Test	97.69	5	4	98.95	28.57	33.33	98.69
	Type III	Training	97.61	3	11	98.06	84.21	59.26	99.46
		Test	95.12	3	16	95.82	50.00	15.79	99.19

Looking more closely at the final solution as illustrated in Figure 8.19, several observations can be made. In total 22 nodes are used out of the 60 available. This was to be expected taking into account the results achieved in the multiple program approach. However an important observation is that only a single intermediate node (circled in the figure) is used in more than one of the output results. All other nodes contribute only to a single output. The reason for this is likely to be related to the fact that the errors are not correlated. The patterns in the data corresponding to each error type is likely quite different and the CGP fails to evolve any sub-program that can be reused for all the errors. Thus, the individual evolves three separate sub-programs each detecting one of each type of error. The input data however does contain several variables that contribute to several or indeed all of the outputs.

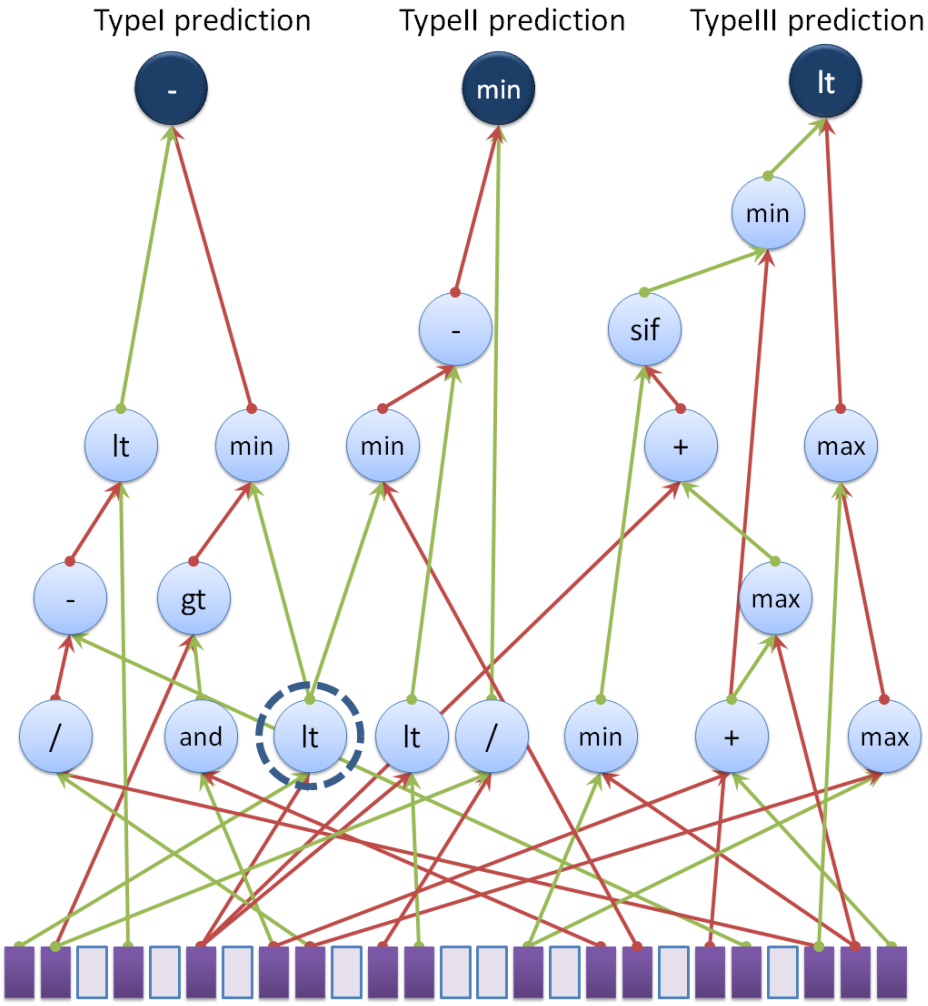


Figure 8.19: Best program found using the single program approach.

8.2.3 ERS data

In this section we will apply our CGP to develop a program to detect errors based on the ERS data. The goal is to establish whether CGP can be used to evolve error detection programs using this type of data.

8.2.3.1 Data preparation

One of the initial questions that must be answered is how to prepare the data in order to use CGP effectively. In Chapter 6 the messages were grouped into blocks of 5 leading to 30 dimensions in the input data. Using so many parameters might be difficult when developing CGP programs as one would need a very large program and hence the evolutionary process might become very time consuming.

In order to avoid this problem the data was further processed. PCA, as described in Chapter 5, was chosen in order to reduce the number of dimensions in the dataset. PCA was applied to the ERS data, thus reducing the number of dimensions in the data to 3. As can be seen in Figure 8.20 this still retains 85% of the variance in the dataset. The resulting dataset was then used for subsequent evolution of error detection programs.

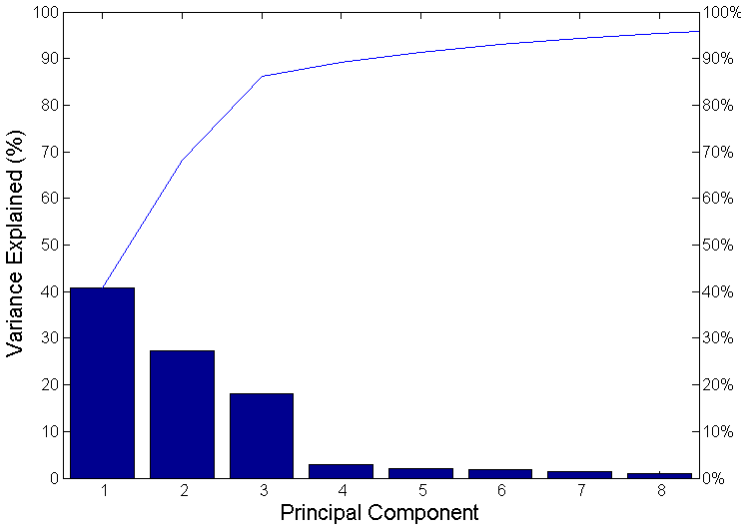


Figure 8.20: Variance explained by the first 8 principal components of the ERS dataset.

8.2.3.2 Initial test

In order to evaluate the feasibility of the CGP approach a number of relatively short runs with a limited population size were attempted. A population size of 96 individuals was used and evolved for 200 generations. This was repeated 100 times. All the CGP parameters are presented in Table 8.11. The different functions available during program evaluation are shown in Table 8.6.

Table 8.11: Parameters for initial testing using the ERS data

Parameters	Value
Evolutionary model	HFC
Individuals	96
Generations	200
Mutation probability	5%
Crossover probability	80%
Layout	Grid
Rows * columns	4*7
Backwards connections	3
Selection strategy	Rank

Figure 8.21 shows the results of the 100 runs. It shows the mean of the best individuals averaged over all populations including the standard deviation at each generation step. The results of these initial tests were quite promising. The optimal program achieved a classification accuracy of 80% on the training data and 61% on the test data. While this is quite a bit worse than that of ANN or SVM, a number of factors should be considered:

- The final solution uses just 12 nodes which is far less than that of the other approaches. The best solution is shown graphically in Figure 8.22. The figure also includes a program listing for a simple C++ routine corresponding to the program.
- As the nodes are all simple functions and thus the program can be evaluated practically 'instantly'.

- A third and very important factor is that the program can be easily read and understood by a human operator. This can give further insight into the data and possibly it can be improved upon by human operators. This also means that it is straightforward to transform the program into rules that can be used in the existing expert system and can be effected with minimal effort.

Taking these factors into consideration the initial results very promising. It should be possible to use CGP in order to develop fast and effective error detection programs.

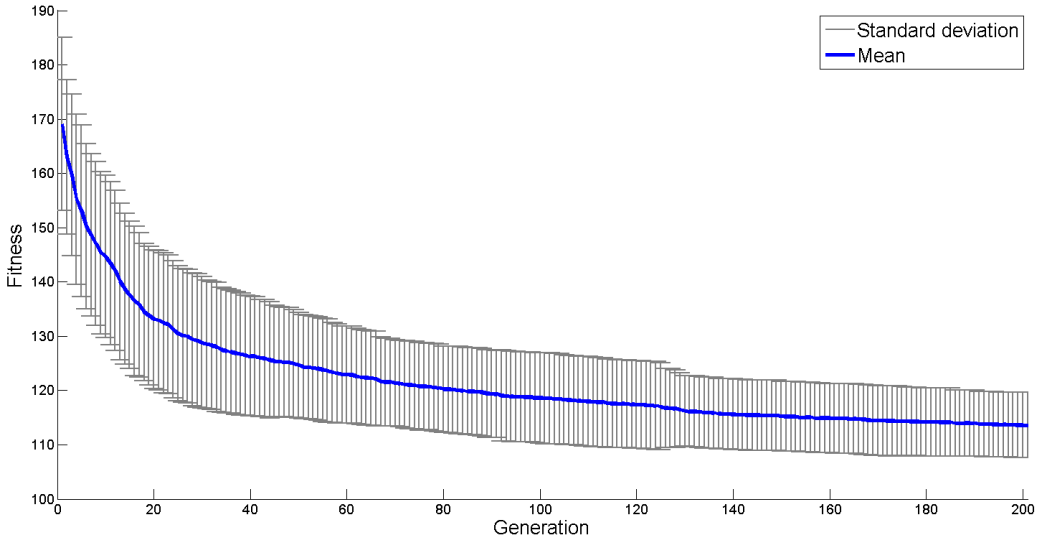


Figure 8.21: Mean and standard deviation for the best individual at each generation averaged over 200 runs.

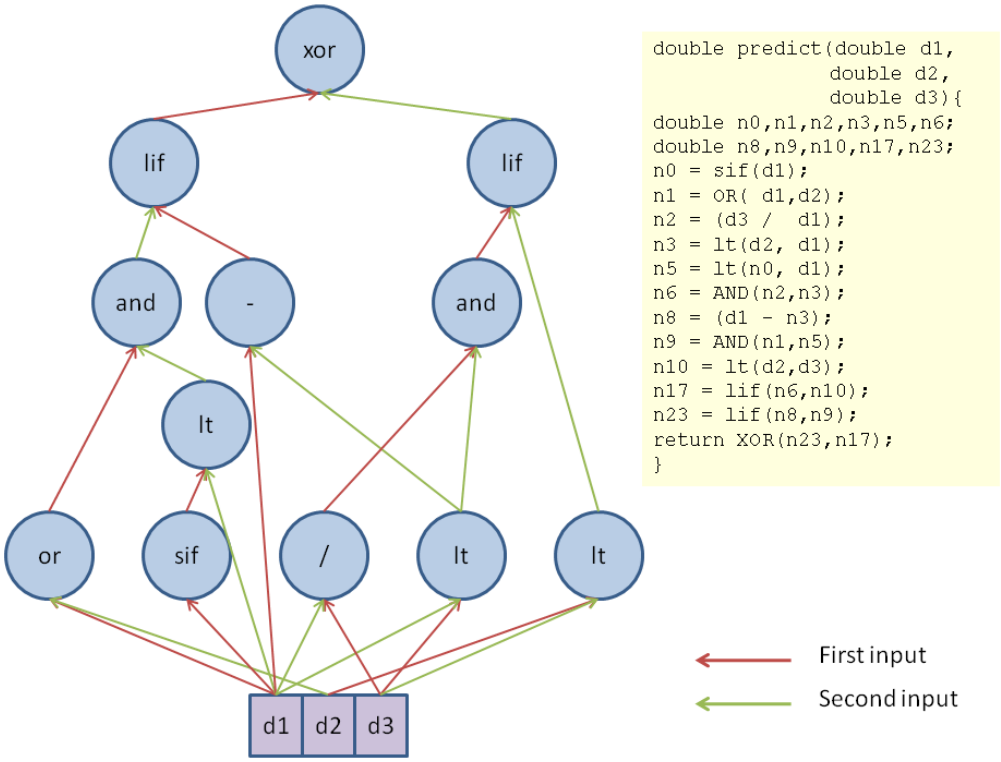


Figure 8.22: Best program found through initial test runs.

8.2.3.3 Selecting functions

In order to choose the optimal set of functions for this particular problem the concept of appearance percentage is 'borrowed' from GNMM as described in Section 6.3. 250 runs were conducted using a small population size of 100 individuals for 200 generations. All the functions listed in Table 8.6 were available during the runs. The functions used in the best individual for each run was recorded. Note that, as we are using CGP, only *active* nodes in the program were considered, i.e. those functions that contribute to the final output of the individual. The appearance percentage of each function was then calculated in order to determine if some functions were significantly more often used in the final solution. The results of the appearance percentages are shown in Figure 8.23. As can be seen from the figure there are some functions that seem to be more useful for the solution, in particular *gt* and *lt*. This comes as no surprise as the problem is basically a classification problem and one would expect such functions to be very useful.

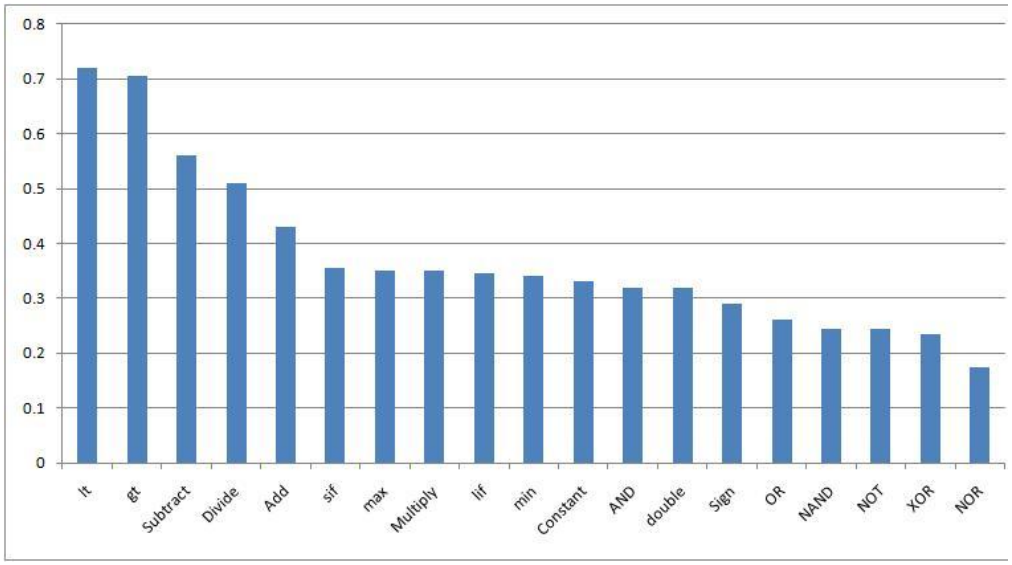


Figure 8.23: Appearance percentage of all functions

In order to verify the results the tests were repeated for 100 runs using only the top 6 functions. The results were then averaged over all runs and compared to those achieved while using all the available functions. The results can be seen in

Figure 8.24. As expected it is clear that using only a subset of the functions enables the population to converge faster and to a better solution on average. However, care should be taken not to select too few functions as this might lead to fast convergence and a good solution, but it might be missing a 'crucial tool' to achieve even better performance.

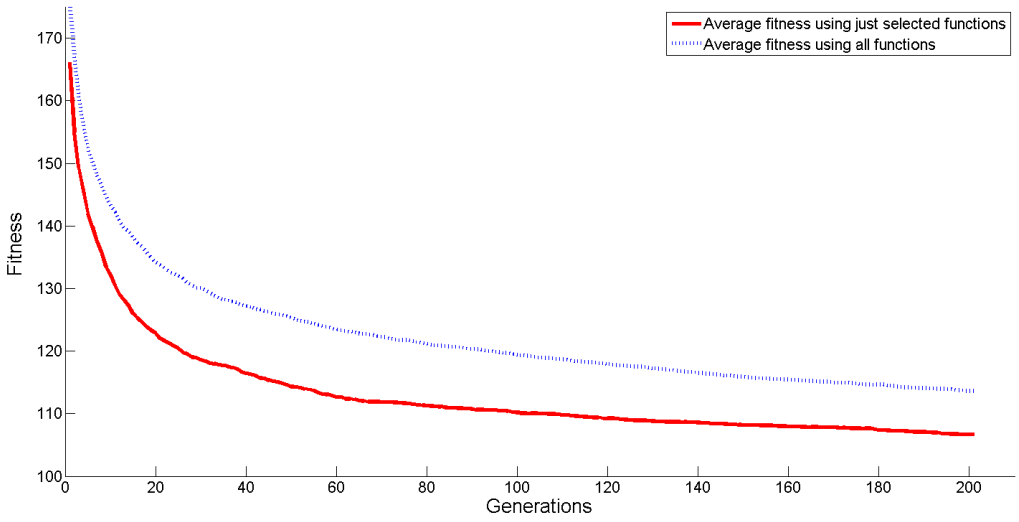


Figure 8.24: Average fitness at each generation when using just selected functions vs the use of all functions.

8.2.3.4 Evolving an error detection program

After determining the best set of functions, CGP was used to evolve a program to do error detection. The parameters for the CGP are shown in Table 8.12 together with the functions which were available during the evolution. In order to evaluate further the overall usefulness of HFC for this problem 30 runs using the normal evolutionary model were performed; 20 individuals were used over 1000 generations. Then 30 runs using HFC were performed, with 100 individuals used over 200 generations. The difference in population size was based on the results in Section 8.1.4.1 where we found that HFC works best when the population is larger. The number of generations is therefore less in order to have approximately the same amount of total computational effort for the two approaches.

Table 8.12: CGP parameters for the ERS runs

Parameters	Batch #1	Batch #2
Evolutionary model	Normal	HFC
Individuals	20	100
Generations	1000	200
Mutation probability	5%	5%
Crossover probability	90%	90%
Layout	Directed graph	Directed Graph
Nodes	30	30
Backwards connections	29	29
Selection strategy	Rank	Rank
Available functions	lt, gt, +, −, /, ∗, min, max, double, sif, lif, constant, and	

Table 8.13 summarises the results of the two different evolutionary approaches. Both approaches produce a best individual which performs similarly to the best individual found using the normal evolutionary model achieving 77.06% classification accuracy and the best found using HFC achieves 77.65%. On average HFC achieves slightly better solutions for the test cases with 72.17% accuracy versus 68.49% for the normal model. However this might be due to the probabilistic nature of evolutionary approaches and further studies should be conducted before drawing a final conclusion.

Table 8.13: CGP results using the ERS data.

Network type	Solution	Dataset	Accuracy	FN	FP	Specificity	Sensitivity	PPV	NPV
CGP (Normal)	Best Ind	Training	79.84	67	36	82.09	78.39	87.10	71.12
		Test	77.06	42	36	75.00	78.57	81.05	72.00
	Average	Training	79.75	70.9	32.6	83.78	77.13	88.17	70.43
		Test	68.49	45.3	61.9	57.05	76.89	73.07	N/A
CGP (HFC)	Best Ind	Training	81.21	63	33	83.58	79.68	88.21	72.73
		Test	77.65	46	30	79.17	76.53	83.33	71.25
	Average	Training	79.30	69.2	36.5	81.82	77.67	86.96	70.47
		Test	72.17	59.7	34.9	75.76	69.52	80.19	66.80

Graphical representations of the best individual found using the normal and the HFC model are shown in Figure 8.25 and Figure 8.26 respectively. One can see that the HFC solution contains less nodes and has a simpler structure. It is of course possible that the individual found using the normal approach could be further simplified taking into account the constraints on the input data (i.e. some of the nodes might always evaluate to the same value), hence this should not be given too much importance at this point.

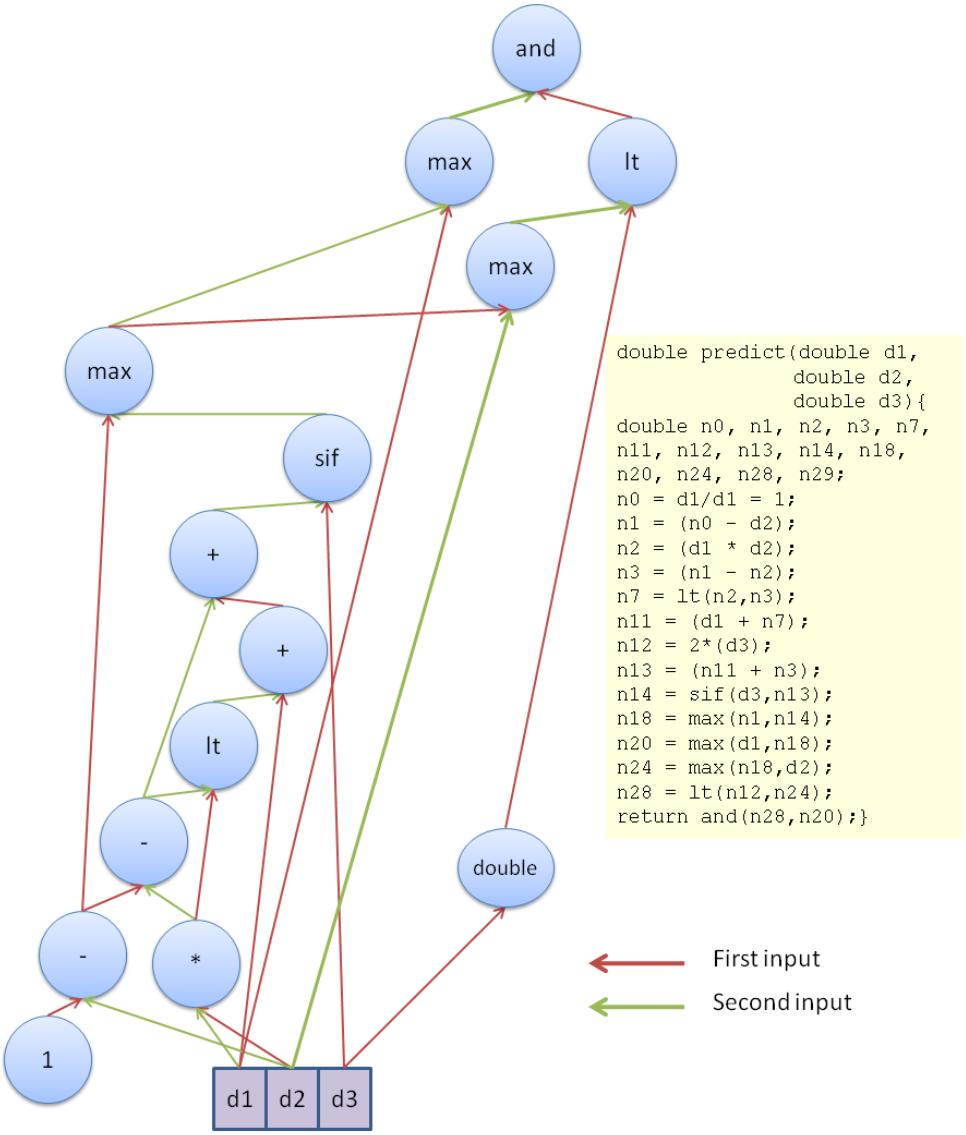


Figure 8.25: Best program found using the normal evolutionary model.

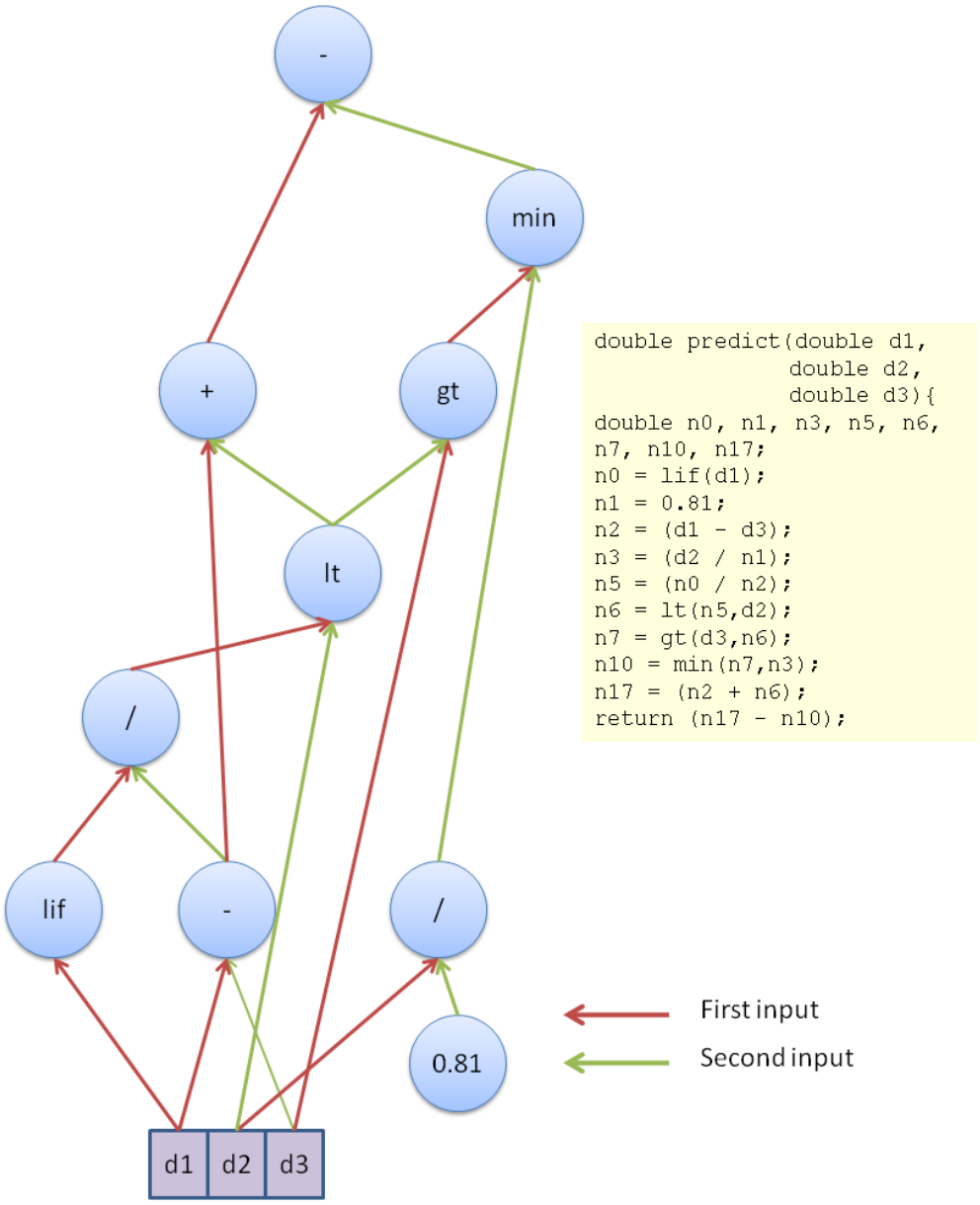


Figure 8.26: Best program found using the HFC evolutionary model.

Overall the results show that it is possible to develop an error detection program based on the ERS data achieving good results, though it is unable to match the performance of the ANNs and SVM approaches presented in Chapter 6 and Chapter 7. Though one should take into account that the test data were used after first reducing the dimensionality of the dataset using PCA. This means that

some information is likely to have been lost. However, the approach was taken in order to keep the computational effort within a reasonable limit. For future tests it would be of interest to perform evolution using all the available data in order to see whether better results can be achieved.

8.3 Discussions and conclusions

The results for the ERS data were encouraging and show promise for the use of CGP to evolve error detection programs achieving a classification accuracy of 81.21% and 77.65% for the training and test set respectively. While it did not match the results achieved using ANNs and SVMs it is still a promising avenue for further research. The CGP used a simplified input set and the resulting programs are small, and therefore easy to analyse and incorporate into the existing system. It should be noted that while the CGP is relatively easy to analyse it is not trivial to infer any general observations about the system from the CGP programs. As the input data has been preprocessed including reducing the dimensionality of the data one cannot directly see which values of the initial datasets are included and how they are utilized in the CGP programs. This reduces the transparency of the CGP evolved programs and makes rule extractions (at least at a general level) very hard.

For the IS dataset excellent results are achieved which are in most cases comparable to the results achieved using ANNs and SVMs in Chapter 6 and 7 respectively. The best individuals achieved a classification rate for the test set of 97.94%, 98.46% and 99.20% for the TypeI, TypeII and TypeIII problems respectively. The programs were able to perform an automatic input variable selection in each case using only a subset of the available parameters. The size of each program was also automatically determined through evolution and stayed well within the maximum number of nodes available in an individual. These excellent results make CGP a good choice for quickly developing error detection programs based on the available IS data. Especially taking into account the fact that the programs can very easily be ported into the existing EMS framework.

The overall results show that CGP can be used to evolve error detection programs in the context of the ATLAS TDAQ system. The simplicity and portability of the evolved rules make this approach a promising one and, importantly, one which can easily be tested within the actual system.

References

T Blickle and L. Thiele. Genetic programming and redundancy. pages 33–38. Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94, Saarbrucken), 1994.

Janet Clegg, James Alfred Walker, and Julian Frances Miller. A new crossover technique for cartesian genetic programming. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1580–1587, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: 10.1145/1276958.1277276.

Edgar Galvanes-Lopez and Riccardo Poli. An empirical investigation of how and why neutrality affects evolutionary search. Seattle,, 2006. GECCO'06.

S. Harding and J. Miller. Evolution of robot controller using cartesian genetic programming. In *6th European Conference on Genetic Programming (EuroGP 2005)*, pages 62–72. Springer, 2005.

Jian Jun Hu and E.D. Goodman. The hierarchical fair competition (hfc) model for parallel evolutionary algorithms. *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, 1:49–54, 12-17 May 2002. doi: 10.1109/CEC.2002.1006208.

Jianjun Hu, Erik Goodman, Kisung Seo, Zhun Fan, and Rondal Rosenberg. The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(2):241–277, 2005. doi: 10.1162/1063656054088530.

John R. Koza. Genetic programming. In James G. Williams and Allen Kent, editors, *Encyclopedia of Computer Science and Technology*, volume 39, pages 29–43. Marcel-Dekker, 1998.

Julian Miller. What bloat? cartesian genetic programming on boolean problems. In Erik D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 295–302, San Francisco, California, USA, 9-11 2001.

Julian F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann. ISBN 1-55860-611-4.

Julian F. Miller and Peter Thomson. Cartesian genetic programming. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802, pages 121–132, Edinburgh, 15-16 2000. Springer-Verlag. ISBN 3-540-67339-3.

Julian F. Miller and Peter Thomson. A developmental method for growing graphs and circuits. In *5th Int. Conf. on Evolvable Systems*, pages 93–104. Springer, 2003.

Julian Francis Miller and Simon L. Harding. Cartesian genetic programming. In *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 2701–2726, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-131-6. doi: http://doi.acm.org/10.1145/1388969.1389075.

Alan Piszcz and Terence Soule. Genetic programming: profiling reasonable param-

eter value windows with varying problem difficulty. *Int. J. Innov. Comput. Appl.*, 1:108–120, January 2007. ISSN 1751-648X. doi: 10.1504/IJICA.2007.016792.

James Walker and Julian Miller. Embedded cartesian genetic programming and the lawnmower and hierarchical-if-and-only-of problems. GECCO'06, 2006.

James Walker and Julian Miller. Changing the genospace: Solving ga problems with cartesian genetic programming, 2007.

James Alfred Walker and Julian Francis Miller. Investigating the performance of module acquisition in cartesian genetic programming. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1649–1656, New York, NY, USA, 2005. ACM. ISBN 1-59593-010-8. doi: http://doi.acm.org/10.1145/1068009.1068287.

Chapter 9

Conclusions and further work

This chapter summarises the results and presents the main findings of the research presented in this thesis. Final conclusions are drawn before some possible avenues of future research are outlined.

9.1 Summary of main findings

This thesis focuses on the development of new techniques, using a ITS approach, within the context of the Error Management system of the ATLAS TDAQ system at CERN. Chapter 1 provides an introduction to the problem and presents the thesis objectives. Chapter 2 then provides the background of the ATLAS TDAQ system and outlines some of the key components in the system in the context of error management. In Chapter 3 the existing ES approach to error management is described. The main limitations are identified in Section 3.2.4 which include:

(i) The current system lacks adequate error detection capabilities. More specifically only explicit errors such as applications crashing or stopping are dealt with. There is no detection of errors that occur 'silently'.

(ii) Due to the rules based nature of the system adaptability to new error scenarios is difficult, often requiring significant time and effort from experts.

(iii) The generalisation ability of the system is also not sufficiently good.

This thesis has attempted to overcome these limitations in the following ways:

(i) An ITS approach has been taken where error messages from the applications (ERS data) and other information published by the applications themselves (IS data) are first analysed and then used in order to detect errors in the system. Hence if there are any patterns in the ERS or IS data it would be possible, using the described ITS techniques, to train/evolve a system that is able to detect and classify those patterns. This would greatly increase the error detection and classification capabilities of the system.

(ii) The ITS approach taken, from analysis through pre-processing and training/evolution of error classifcation systems, can be fully automated. This means that in order to detect new problems one only needs to gather data corresponding to those scenarios. While this step would require both insight into the system and significant human intervention it does not necessarily involve system experts. It should also in most cases be easier than defining new rules covering all the different scenarios.

(iii) The ITS approaches are all robust to noise or minor variations in the input data. All the approaches search for overall patterns in the data as opposed to reacting to a specific set of conditions. Hence, the system should be able to deal with small variations in the input data (such as some applications sending different ERS messages) while still correctly detecting and classifying errors.

Note that some limitations remain. As all the input data are heavily pre-processed it is not easy to extract any general rules from the ITS systems developed, whereas ideally one would like to understand the root causes of the errors and potentially prevent them from occurring in the first place.

The overall approach was presented in two main parts:

9.1.1 Part one

Chapter 4 describes error detection from a theoretical point of view. Different measures of the effectiveness of error detection systems are presented before the specifics of the TDAQ system in the context of error detection are described.

The experimental setup is then described in Chapter 5 together with a description of how error situations were induced in order to simulate real errors in the system. It then describes the analysis and visualisation techniques that were subsequently used. These techniques were then applied and it was shown that there are some clusters in the data corresponding to the times the errors were induced.

9.1.2 Part two

The second part of the thesis then investigates different techniques for developing error detection systems based on the two datasets gathered in the second part.

Chapter 6 applied ANNs in order to learn from the two datasets. 4 different types of ANNs were trained using the datasets and the results were compared. For the IS dataset the best results were achieved using a PNN which achieved a generalisation performance ranging from 99.59% to 99.86% on the test set for the three error types, though all the different ANN types achieved relatively good performance. For the ERS dataset the best performance was obtained using a constructive RBFN which achieved a classification rate of 94.51% on the test set.

An ensemble approach was then attempted using the output of the four networks as input to the ensemble. The ensemble output was the weighted sum of the 4 ANN outputs. The weights were found using a genetic algorithm as the search space is too large to perform a brute force search and too complex to use any hill-climbing algorithms. The ensemble achieved better results than that of any of the single ANN for the ERS dataset with a generalisation performance of 94.90%. The ensemble approach thus improved upon the result, but at the cost of higher complexity and computational effort. For the IS data the ensemble was not able to improve upon the results of the PNN. This is however to be expected as the PNN already achieved near optimal results and an ensemble approach might therefore be a useful approach also for the IS data if the classification task is harder and thus not solvable by a single ANN.

In Chapter 7 a support vector machine approach was investigated. SVMs were chosen as they are especially efficient when the data is sparse and/or imbalanced

making them a good fit for the TDAQ data. The theory of SVMs was described including some possible approaches for dealing with multi-class data. SVMs were then applied to the ERS and IS datasets using a brute force algorithm in order to determine the parameters. Two different kernels, Gaussian and polynomial, were tested. While both kernels achieved very good results, the Gaussian kernel obtained the best results for both problems achieving a generalisation accuracy of 100% for the IS data thus showing that a perfect classifier is possible based on that particular dataset. The SVM also achieved very good results for the ERS dataset with a classification accuracy of 92.64%.

Finally, in Chapter 8 an evolutionary approach was taken. Cartesian Genetic Programming was introduced and compared to the standard genetic programming approach. A new form of crossover was introduced and compared to the standard one, achieving better results for several types of problems. The evolutionary model of Hierarchical Fair Competition (HFC) was also tested with CGP and compared to the regular approach showing better results for some types of problems while being detrimental for others. CGP was then used to evolved error detection programs using the same datasets as was used for the ANNs and SVMs as input values. The CGP results were slightly worse than that of the ANN and SVM approaches achieving a generalisation accuracy ranging from 97.94% to 99.20% using the IS dataset and 77.65% for the ERS dataset. The IS performance is only slightly worse than that of the ANN and SVM approaches while for the ERS data the performance is significantly worse.

It should be noted that the classification rates achieved by the different techniques in Part two are very high and might indicate that the initial datasets was too 'easy' to classify. It is likely that simpler approaches could have achieved similarly good results for the dataset. However, the main goal of the thesis was to establish a framework for extending the existing TDAQ EMS something that has successfully been achieved.

9.2 Future research

There are a number of ways forward in order to build upon the work presented in this thesis. The following is a consideration of some of the main points before specific research is suggested grouped by the relevance to sections of the thesis.

9.2.1 General points

At some point the techniques investigated in this thesis must be implemented in the actual ATLAS TDAQ system in order to evaluate how well they will be able to perform and of how much use they actually are to the operators. This includes the integration with existing components and how to best incorporate the developed techniques is still a question to be addressed. Integration with the existing EMS system and the expert system core is a natural way forward.

Further tests should be performed using different configurations of the TDAQ system both software and hardware-wise, ideally using a larger experimental setup. More and other types of errors could be simulated providing a larger dataset on which to test the techniques. Furthermore, only a small number of techniques have been investigated in this thesis, though an attempt was made to choose the techniques that are most relevant and commonly used for such problems. Other techniques could be investigated and compared to the ones presented in this thesis and the results could be compared. Two promising approaches include ANFIS (Jang, 1993) and Fuzzy ARTMAP (Carpenter et al., 1992).

9.2.2 Analysis

Some new promising methods of clustering exist which could give further insight into the data or used to verify the results presented in this thesis. One particularly interesting approach is clustering using affinity propagation (Frey and Dueck, 2007) which claims to be better and faster than many other clustering methods.

As methods are being developed and further understanding of both the techniques involved and the TDAQ system itself is gained, it will be possible to use this

knowledge to improve the techniques. For example, it would be possible to filter out irrelevant information from the dataset as part of the preprocessing of the data. Efficient methods for automatically filtering the data could therefore be of great interest.

9.2.3 The ANN approach

Further ANN topologies should be tested in order to find the optimal choices. This includes further parameter tuning of each ANN. Potentially one could use different ANNs to detect and classify different types of errors in the system.

Rule extraction techniques such as described in (Towell and Shavlik, 1993) provides a way of extracting rules from the trained ANN. This can both provide further insight into the reasons and implications of the errors, and could be integrated into the existing expert system approach described in Chapter 3 which remains at the core of the ER system.

9.2.4 The SVM approach

While the initial results are indeed very good, there are still a number of issues to address. As mentioned the determination of kernel parameters automatically is currently performed using a grid search algorithm and could be improved upon considerably. This together with the automatic determination of model size inherent to SVMs means the approach is one of the most promising attempted so far.

Also other types of kernels should be compared in order to find the optimal choice for this particular type of data. This might of course be different for the IS and ERS datasets, but further research must be done in order to determine this.

9.2.5 The CGP approach

There are a number of potential ways forward using CGP of which just some are mentioned below:

Studies of the impact of evolutionary parameters such as mutation rate, crossover rate, population size, individual size, evolutionary model, available func-

tions, etc could be made in order to find an optimal solution for each problem. As one attempts to solve different problems, or possibly using different data sources, it might be prudent to test different fitness functions as well.

Furthermore, one could use the existing developed programs and combine them in an ensemble approach in order to achieve better results at the cost of further complexity of the solution.

A more ambitious approach would be to directly evolve rules to be used in the expert system core of the EMS. One would naturally need to ensure that syntax restraints are kept and that the rules operate on parameters available in the expert system working memory.

Further research on the use of HFC in conjunction with CGP should be conducted. In particular, its impact on different problems could be explored to better understand when it is and when it is not an advantage to use the model. Population size also seems to be of importance when using HFC and a detailed study of the impact of population and deme sizes could be conducted.

9.3 A final word

As stated in Chapter 1, developing an effective EMS system is of great importance to the ATLAS experiment. This thesis has investigated some potential techniques for improving the error management system that is in place today and has shown that many of the techniques presented in this thesis could very well be included in the TDAQ system in the future. It has been shown that clustering and analysis techniques can provide insight into datasets gathered from the system providing both a means of gaining better understanding of the system and to visualise key aspects of the different datasets.

Much of the work will also translate to similar large, distributed software systems and could therefore be of use to researchers who are working on similar problems. It is recommended that error management is given a serious role early in the development of such systems and that the development of an EMS is facili-

tated throughout the system. This is especially important in large systems where development of the individual sub-systems is often done by different groups and the integration of the system as a whole only happens at a later stage. This was not the case for the TDAQ system where an overall EMS was only developed very late in the project and with some severe restrictions (e.g. with a minimal of change to the existing code) hence making it less effective than it could have been. Given the importance of error detection and management in such large scale system the author would strongly advice that designers of future systems to include the facilities needed to develop a proper EMS from the start.

References

G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen. Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. *Neural Networks, IEEE Transactions on*, 3(5): 698–713, Sep 1992. ISSN 1045-9227. doi: 10.1109/72.159059.

Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, page 1136800, 2007. doi: 10.1126/science.1136800.

J.-S.R. Jang. Anfis: adaptive-network-based fuzzy inference system. *Systems, Man and Cybernetics, IEEE Transactions on*, 23(3):665–685, 1993.

Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

Appendix A

Publications

Detecting errors in the ATLAS TDAQ system: A neural networks and support vector machines approach

John Erik Sloper*†, Evor Hines*
*School of Engineering, University of Warwick, Coventry CV4 7AL, UK
†European Organization for Nuclear Research - CERN
Geneva, 1211 Switzerland

***Abstract*—This paper describes how neural networks and support vector machines can be used to detect errors in a large scale distributed system, specifically the ATLAS Trigger and Data AcQuisition (TDAQ) system. By collecting, analysing and preprocessing some of the data available in the system it is possible to recognize and/or predict error situations arising in the system. This can be done without detailed knowledge of the system, nor of the data available. Hence the presented methods could be used in similar system without significant changes.**

The TDAQ system, and in particular the main components related to this work, is described together with the test setup used. We simulate a number of error situations in the system and simultaneously gather both performance measures and error messages from the system. The data are then preprocessed and neural networks and support vector machines are applied to try to detect the error situations, achieving classification accuracy ranging from 88% to 100% for the neural networks and 90.8% to a 100% for the support vector machines approach.

I. Introduction

The ATLAS experiment[1] is at the very forefront of particle physics research and incorporates a large number of custom hardware and software modules. It consists of a large cylinder (43 m length x 25 m diameter) of detecting devices, placed around one of the beam collision points of the Large Hadron Collider [2]. Its scope is to determine which particles are produced during collisions of high energy protons every 25 nano seconds. The High-Level Trigger and DataAcQuisition (TDAQ) system of the ATLAS experiment is a large heterogeneous system consisting of a wide variety of software and hardware components. The final production system will consist of approximately 3000 nodes running more than 50000 processes. The system is divided into a number of sub-systems each performing a well defined task. A detailed description of the ATLAS TDAQ system can be found in [3].

II. Background and Motivation

The ATLAS TDAQ system is a vast and complex distributed system, a characteristic that makes it very difficult for any single human user to gain a complete understanding of the system, or indeed even of a sub-system. The sheer number of applications and the interconnectivity between them, makes any analysis or prediction of the system behaviour difficult using classical techniques and monitoring approaches. For example, the applications running in the ATLAS TDAQ system may produce a large number of error messages of various levels of severity. After a typical week of running the system, several million of these messages may have been issued and subsequently gathered and stored in the 'log database'. Analysing all the messages, or even browsing through them, is non-trivial for a human user, especially as the number of messages grows. Thus while the database is indeed very useful when investigating a particular problem, the only overall analysis of the data currently being performed are simply statistical measures. Introducing different techniques for automating the analysis process, including different visualization techniques, would be very useful and would allow us to benefit further from the gathered messages.

In addition it is of great value to be able to detect errors occurring in the system. The existing methods of error detection largely focuses on discrete events such as processes exiting/dying, machines no longer responding, etc. The TDAQ system already has methods and components to detect those cases and is able to deal with, and recover from, most errors of that kind. However, scenarios where the system, or parts of it, is still partly or periodically operational are more complicated. These scenarios are often difficult to detect and it is often not trivial to understand exactly why they occur or what is the cause. There exist however a great deal of 'meta-data' in the system, that is data such as performance measure, error messages etc. This data is produced by all the applications in the system and is readily available. However, as these datasets may contain thousands or even millions of observations and due to the complexity of the system, they are not easily analysed nor understood. In this paper we will investigate how some of this data can be automatically processed and it can be determined if patterns exist in the data corresponding to different error scenarios. We will then try to detect these error situations using neural networks (NNs) and support vector machines (SVMs). In [4] the application of NNs to detect errors in the TDAQ system based on error messages alone was investigated achieving promising results.

III. The TDAQ system

In this section a brief overview of the TDAQ system is given and the data which have been gathered from the system during our experiments are described.

The TDAQ system consists of a number of sub-systems performing different tasks ranging from reading data from the detector, to filtering and transporting the data to mass storage. Figure 1 shows a schematic view of the main components in the TDAQ system. In addition to the components shown in the schematic view, there also exists a number of services providing functionality such as inter process communication, process control, error reporting and information sharing.

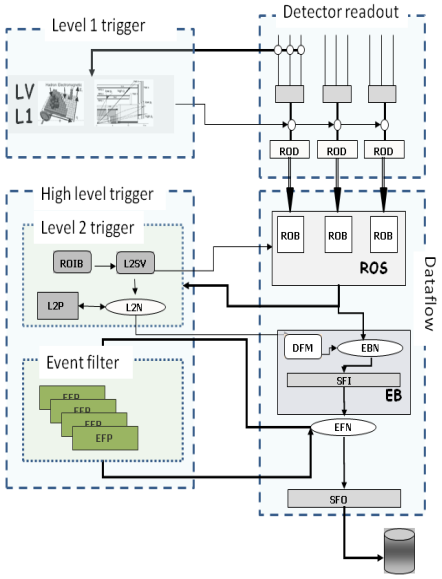


Fig. 1. Schematic view of the TDAQ system.

A. Message production

The TDAQ system is a vast and heterogeneous system and will ultimately consist of more than 50000 processes running simultaneously. The development has been realized by a large group of software developers developing close to 200 software packages. However, in order to have a uniform means of both creating and distributing errors in the system a package called the Error Reporting Service (ERS) has been created and is used by all applications. The ERS provides several features including a common format for the errors, a fixed range of severity levels and a uniform way of reporting and transporting the errors. This allows for easy configuration of both the amount of information contained in each error message and how they are distributed in the system.

In order to pass ERS messages between applications, the ERS relies on the Message Reporting Service (MRS). The MRS is a service for passing messages between applications using a subscription-notification model. The ERS marshals the errors, transports them using the MRS and can recreate them on the receiving end. The underlying means of transportation is completely transparently to the applications issuing the errors.

B. Message logging

To store the ERS messages produced in the TDAQ system a 'logger' application is used. This application receives **all** error messages issued by all applications in the system (or rather all messages that are passed using the MRS) and stores them in an Oracle database. The standard format of all error messages simplifies this task significantly. As all messages are stored in a common location, this allows us to easily retrieve the messages for later analysis. We can also take advantage of the typical database features such as indexing and queries in order to make these operations efficient. Details of the log service mechanism can be found in [5].

C. Performance measures

The TDAQ system provides an Information Service (IS) which allows any application to both publish any kind of information and to subscribe to changes in information published in the service. Any type of information can be published and most applications provide some information about itself, be it rate of data processing, CPU utilization, number of threads, amount of used memory, etc. All this data can easily be gathered while running the system. Figure 2 shows an example of performance data produced for one of the applications in our test setup.

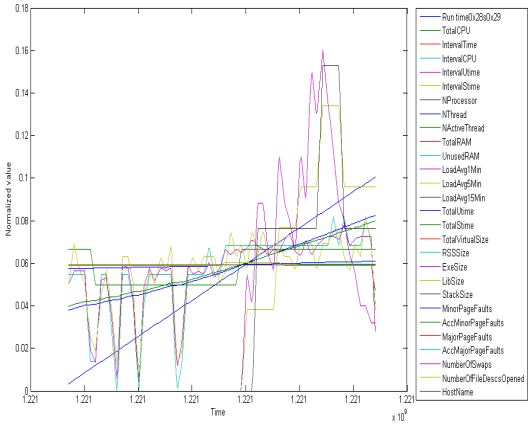


Fig. 2. Example information published by a single application. The graphs shows the values in time as our test setup is being run and errors are introduced.

IV. Experimental setup

Naturally the new error detection methods cannot be developed on the actual ATLAS TDAQ system, as this is now being phased into an operational stage and is not readily available for development of new functionality. In order to perform our tests we utilized a test lab available at CERN specifically designed to perform tests of the TDAQ software. The software itself is also designed and developed so that it is possible to test it in a number of environments and it is not dependent on the actual ATLAS detector. The test setup used in this paper includes the 'full chain' of the TDAQ system including the majority of applications types that will be used for the experiment. Of course all physics data and other variables that would normally originate from the detector were simulated. The setup consists

of 30 machines running approximately 50 processes, and to a large extent follows the layout shown in Figure 1. Though the size of our test setup is much smaller than that of the actual TDAQ system, it nevertheless exhibits similar behavior and does therefore provide a good base for testing and performing 'proof-of-concept' experiments while keeping the system to a manageable size.

A. Simulating errors

As with any system of its size, a wide range of problems may occur in the TDAQ system. In the scope of this paper we will focus on a well defined problem. We have chosen to consider the case in which an application has silently stopped working, though it is still running. This is a fairly likely scenario, and could happen due to a number of reasons such as an internal failure or some system related problems. We will simulate this situation by using POSIX signals. POSIX is an IEEE standard (ISO/IEC 9945) for user and software application interface to the operating system and is supported by the UNIX systems we use for our experiments. Among other things, it defines an interface that allows us to send signals/commands to any running process. There are a number of signals available, but we used the SIGSTOP signal, which according to the standard should stop a computer program for later resumption. Thus by sending this signal to an application it will silently stop doing its task and we can then monitor how the system responds. The applications are resumed after a time-out varying from 10-15 seconds. We do this in order to see if a transparent error can be detected using NNs and SVMs. Being able to detect transient errors is important as such errors might be an indication of more severe problems in the system.

The procedure is performed for 8 different applications of two different types. We will refer to errors related to applications of the first type as Type-1 errors, and errors induced in the second type of applications as Type-2 errors. Throughout the tests all ERS messages and all IS information from processes taking part in the chain of data from readout to storage has been gathered.

B. Gathering and preprocessing the datasets

As described in III-B, all ERS messages are stored together in a single Oracle database. Messages corresponding to our tests were therefore retrieved from the log database a posteriori; eliminating the need for 'online' gathering of the messages which, while feasible, is slightly more complicated. The data include timestamps, making it easy to correlate the messages to the time of the induced errors. A total of 1645 messages are retrieved from the database. In addition we gather the information published in the IS. This is done in real time and the information are stored as time series in a number of files corresponding to each application.

Naturally the datasets gathered must be pre-processed to make it possible to apply NNs and SVM techniques. It is important to note that the preprocessing is done without using any knowledge of the logical meaning of the data, but it is simply transformed into a numerical form suited to the application of the techniques in question. The reasoning for this is as follows:

While it is possible and in many cases useful to understand the logical meaning of the data (for example a message could correspond to the fact that a connection has timed out), we have not attempted to perform such analysis in this work. Instead we rely on the fact that certain patterns in the datasets may correspond to certain error conditions in the system. This has the advantage that we do not need to perform any parsing of the data nor build up a knowledge base of the 'actual' reason for errors or conditions, something that would be extremely time consuming to do given the size and complexity of the TDAQ project.

The disadvantage is that one relies totally on the fact that errors will cause a distinct pattern in the data. Also it might be very difficult to distinguish some error types as they may lead to similar patterns in the processed data. The difference between such errors may however be obvious if one had parsed all the messages and had access to a knowledge base as described above.

V. Data mining and visualization techniques

A. Self organized maps

Self organized map (SOM) is one of the techniques we will use for clustering and data visualization. It has widespread use in data mining, classification and data visualization. The SOM was first introduced by Kohonen [6]. SOM is based on unsupervised learning, meaning it does not use any a priori knowledge of the data to be clustered. It is capable of finding classes/clusters inherent in the data (though depending on the dataset this will not always be true).

B. Principal Component Analysis

Principal component analysis (PCA) is a mathematical method that reduces the number of dimensions in a dataset while retaining as much of the variance as possible. The method has a wide range of uses including data mining, dimensionality reduction and de-correlation, pattern recognition and (lossy) data compression [7][8]. We will use it to simplify our dataset and to eliminate redundant information and filter out noise. PCA is realized by projecting the original M dimension data into D dimensional space where $D \leq M$. This D dimensional space consists of orthogonal vectors called principal components. The principal components are chosen in such a way that a maximum of variance is achieved and are ordered by the amount of variance explained by them. In this way the first principal component explains more variance than the second and so on. By choosing $D < M$ one can therefore project the data into a lower dimensional space while retaining as much variance as possible. There exist a number of variants of PCA [9], though we will use the regular form in this paper.

C. Fuzzy C-means clustering

Fuzzy C-means (FCM) clustering was first proposed in 1973 by J.C. Dunn[10] and was further developed by Bezdek[11].

It is a method where a data point is allowed to belong to two or more clusters according to a fuzzy membership function for each cluster. For each point x_i, there exists a degree of membership to a cluster j represented as U_{ij}. Each cluster centre C_j is calculated using:

$$C_j = \frac{\sum_i^n U_{ij}^m x_i}{\sum_i^n U_{ij}^m}$$

The algorithm then iteratively updates these memberships until a predefined error tolerance is reached.

VI. Preliminary observations

When analysing the datasets it immediately becomes clear that for the Type-2 errors there are no, or very few, ERS messages being sent in the system. There may be various reasons for, and they may even be related to bugs in the software and should therefore be subject to further analysis. However this will not be discussed here, but it does mean that it will not be possible to detect errors using our techniques based on the ERS data for the Type-2 errors.

Also it should be mentioned that the IS information is only updated approximately every 5 seconds, while ERS messages can be continuously sent and received in the system. This means that an approach based on ERS messages will potentially be able to detect the errors earlier. However this is not necessarily the case. Consider a situation where an application is waiting for some data and will not report any error until a given time-out is reached. In a such a case the IS information might immediately, or at least within the 5 seconds delay, show a degradation in the applications throughput or efficiency while the error message(s) is dependent on the given time-out. There is therefore no obvious rule as to which approach would be the best, and it is likely that a combination of the two could yield even better results.

VII. Clustering results

Both the ERS and IS datasets were processed using both PCA and SOMs. The result of the PCA of the ERS data can be seen in Figure 3. As can be seen from the figure there are some distinct clusters in the data. This can be verified by looking at the result of the SOM applied to the same data. The U-matrix and the individual component planes of the trained SOM are shown in Figure 4 and Figure 5. Also for the IS dataset there were some immediately obvious clusters, though not as distinct as for the ERS data. The U-matrix for the SOM trained with the IS data is shown in Figure 6.

By applying FCM clustering of the results of PCA and SOM we find 6 main clusters in the ERS data and 7 clusters in the IS data. The number of clusters were automatically determined using the Davies-Bouldin index [12]. By further investigating the messages contained in each cluster we find that two particular clusters correspond well to the time when the error situations where simulated. This leads us to believe there exists a pattern of messages corresponding to the error situations, which can consequently form the basis for the development of an error detection system/technique. NNs and

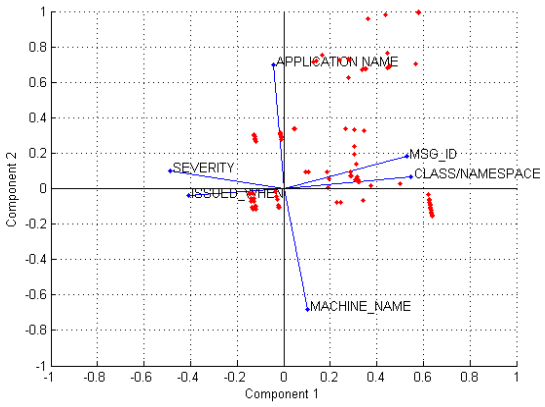


Fig. 3. ERS data projected onto the first two principal components.

then SVMs will be applied to the two datasets in the two following sections; VIII and IX and the results of the two approaches will then be compared.

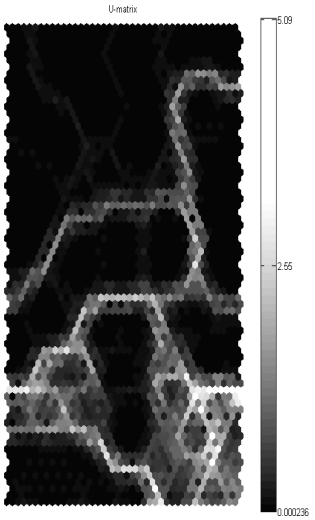


Fig. 4. SOM U-matrix for the ERS data.

VIII. Neural network approach

This section describes the types of networks used, and how they were applied to classify the datasets gathered during our tests. Two main types of neural networks were tested in this work:

- Feed forwards back propagation (FFBP) - this is the 'classic' multi layer perceptron (MLP) network consisting of an input layer, a hidden layer and an output layer. The network is trained by feeding input through the network and propagating errors back in order to train the weights.
- Time Delay Neural Network (TDNN) - TDNNs were first introduced by Waibel in 1987 for use in speech recognition [13]. By utilizing a number of delayed inputs, this method allows the network to learn temporal patterns

Fig. 5. Component planes of the trained SOM for the ERS data.

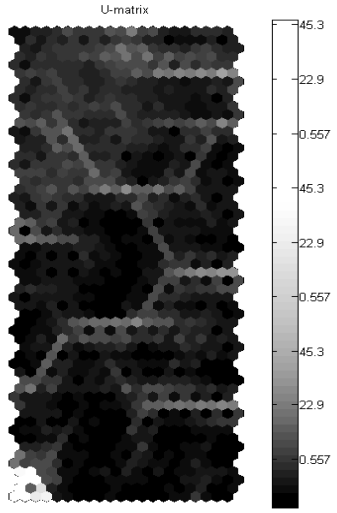


Fig. 6. SOM U-matrix for the IS data.

in the data. Compared to a regular MLP each unit in the first hidden layer is modified to be presented with DN delayed input vectors in addition to the un-delayed input. If we consider an input vector to be of size I=5 and a delay of N=2 the hidden units will have 15 weighted inputs. As each input vector is presented to the hidden units at N+1 different points in time, it allows the TDNN to correlate temporal changes in the input data. It is possible to introduce delays at subsequent hidden layers, but we will use the simple form of delays only at the input layer in the following work. TDNNs normally use the back propagation algorithm for learning, modified to take into account the delayed inputs.

We use the same approach for both the ERS data and the IS data using 70% of the dataset as training data and the remaining for test. The FFBP network achieved 96.9% and 90.8% accuracy for the training and test set respectively. The TDNN achieved 92.9% and 82.7% classification accuracy. The overall results from classifying the ERS and IS data are shown in Table I and Table II respectively.

Note that *all* the introduced error scenarios were detected at some point before the process was resumed (as described in section IV-A).

IX. SVM APPROACH

SVMs are generalized linear discrimination machines that take advantage from operating in a transformed, typically higher dimensional, space in order to best discriminate classes of data[14]. The basic idea behind SVMs, which was originally developed for classification, is to select the optimal class separating hyper plane having the maximum distance/margin from the classes' convex hulls. One can find motivation for this choice in the framework of statistical learning theory and intuitively lead to enhanced generalization properties. Subsequently, SVMs have been generalized for regression and functional approximation problems and in this formulation are also known as Support Vector Regressors (SVRs).

We use a regular SVM approach to identify the error conditions using the ERS data achieving 94.7% and 90.8% classification rate for the training and test data respectively. This is slightly worse than that of the MLP, but better than the TDNN.

For the IS data we use the 'one-against-all' algorithm for classifying the data as there are more than one class to be identified. The 'one-against-all' algorithm creates a SVM classifier for each class, attempting to separate data of that class from all the others. The IS data classification rates of 100% for the training data and 92.9% for the test data is best of all approches. The overall results are shown in Table I and Table II.

As for the NN approaches, the SVM is able to detect all error scenarios at some point before the error was cleared from the system.

X. CONCLUSIONS

In this paper we have shown how NNs and SVMs can be successfully applied to error detection problems using ERS and IS data gathered from the TDAQ system. We have performance ranging from 88% classification accuracy in the worst case to 100% accuracy in the best case, while detecting all introduced errors at some point in time. This detection was based on data which currently is largely unused and promises new ways of adding to or complementing existing methods of error detection.

Both SVMs and NNs are relatively easy to (re)train, which is of great importance to us as the system is still under development and can therefore change behaviour relatively often, something that might affect existing classifiers.

TABLE I
PERFORMANCE OF THE DIFFERENT NETWORKS AND OF SVM USING ERS DATA (ONLY TYPE-1 ERRORS)

	Dataset	Overall	Error type 1	No error	False Negative	False Positive
FFBP	Training	**96.9%**	100%	94.8%	0	7
	Test	**90.8%**	100%	84.5%	0	9
TDNN	Training	**92.9%**	97.8%	91.0%	2	12
	Test	**82.7%**	97.5%	72.4%	1	16
SVM	Training	**94.7%**	94.6%	94.8%	5	7
	Test	**90.8%**	95%	87.9%	2	7

TABLE II
PERFORMANCE OF NETWORKS AND SVM USING IS DATA.

	Dataset	Overall	Error type 1	Error type 2	No error	False Negative	False Positive
FFBP	Training	**99.5%**	100%	92.7%	100%	1	0
	Test	**89.4%**	100%	50.0%	92.1%	3	6
TDNN	Training	**100%**	100%	100%	100%	0	0
	Test	**88.2%**	100%	100%	86.8%	0	10
SVM	Training	**100%**	100%	100%	100%	0	0
	Test	**92.9%**	50.0%	66.7%	94.7%	4	2

XI. FURTHER WORK

Further tests will be conducted to ensure the best possible choice in data preprocessing and network architectures. Other types of NNs and SVM approaches will be investigated and their effectiveness evaluated. Especially Fuzzy ARTMAP [15] is of interest as it may be particularly useful in on-line learning, where the developed system does not have to re-learn data already used for training.

Ultimately the techniques and approaches used in our test setup must be validated and tested on the full scale system and be made available to users and developers through simple to use libraries and tools so as to be incorporated in the ATLAS TDAQ system.

ACKNOWLEDGMENT

The authors would like to thank their colleagues for their continuing input, ideas and feedback throughout the design and development of the system.

REFERENCES

[1] *ATLAS: technical proposal for a general-purpose pp experiment at the Large Hadron Collider at CERN*, ser. LHC Tech. Proposal. Geneva: CERN, 1994.

[2] L. Evans, "The large hadron collider," *New Journal of Physics*, vol. 9, no. 9, p. 335, 2007. [Online]. Available: http://stacks.iop.org/1367-2630/9/335

[3] ATLAS Collaboration, "Atlas high level trigger, data acquisition and controls technical design report," 2003.

[4] E. Hines, M. Leeson, M. M.Ramn, M. Pardo, E. Llobet, D. Iliescu, and J. Yang, *Intelligent Systems: Techniques and Applications*. Shaker publishing, 2008, ch. 8, pp. 241–268.

[5] R. Murillo-Garcia and G. Lehmann Miotto, "A log service package for the atlas tdaq/dcs group," *IEEE Transaction on Nuclear Science*, vol. 54, no. 1, pp. 202–207, 2007.

[6] T. Kohonen, "Self-organised formation of topologically correct feature map," *Biological Cybernetics*, vol. 43, pp. 56–69, 1982.

[7] N. Liu, H. Wang, and W.-Y. Yau, "Face recognition with weighted kernel principal component analysis," in *Proc. 9th International Conference on Control, Automation, Robotics and Vision ICARCV '06*, 5–8 Dec. 2006, pp. 1–5.

[8] D. Tien, K.-W. Lim, and L. Jun, "Comparative study of pca approaches in process monitoring and fault detection," *Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE*, vol. 3, pp. 2594–2599 Vol. 3, 2-6 Nov. 2004.

[9] L. Wei-min and C. Chein-I, "Variants of principal components analysis," *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*, pp. 1083–1086, 23-28 July 2007.

[10] J. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," vol. 3, 1973.

[11] J. C. Bezdek, *Pattern recognition with fuzzy objective function algorithms*. New York,: Plenum, 1981.

[12] D. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Trans. Pattern Anal. Machine Intelligence*, vol. 1, pp. 224–227, 1979.

[13] A. Waibel, A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," vol. 37, no. 3, pp. 328–339, 1989.

[14] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[15] G. Carpenter, S. Grossberg, N. Markuzon, J. Reynolds, and D. Rosen, "Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps," *Neural Networks, IEEE Transactions on*, vol. 3, no. 5, pp. 698–713, Sep 1992.

Dynamic Error Recovery in The ATLAS TDAQ System

John Erik Sloper, *Student, IEEE,* Giovanna Lehmann Miotto and Evor Hines

***Abstract*— This paper describes the new dynamic recovery mechanisms in the ATLAS Trigger and DataAcQuisition (TDAQ) system. The purpose of the new recovery mechanisms is to minimize the impact certain errors and failures have on the system. The new recovery mechanisms are capable of analyzing and recovering from a variety of errors, both software and hardware, without stopping the data gathering operations. An expert system is incorporated to perform the analysis of the errors and to decide what measures are needed. Due to the wide array of sub-systems there is also a need to optimize the way similar errors are handled for the different sub-systems.**

The main focus of the paper is to consider the design and implementation of the new recovery mechanisms and how expert knowledge is gathered from the different sub-systems and implemented in the recovery procedures.

***Index Terms*— ATLAS,TDAQ, Error Recovery, Expert System**

I. Introduction

THE ATLAS High-Level Trigger and DataAcQuisition (TDAQ) system is a large heterogeneous system consisting of a wide variety of software and hardware components. The final production system will consist of approximately 3000 nodes running more than 20000 processes. The system is divided into a number of sub-systems each performing a well defined task. A detailed description of the TDAQ system can be found in [1].

II. Background and Motivation

The TDAQ system has been under development for more than a decade. Even so the development of an advanced Error Recovery (ER) system has not been a priority. Previous error recovery systems have been simple and thus were only able to handle trivial cases such as restarting or ignoring a dead process, or putting the system into an error state. As we approach the start-up of the experiment, larger and larger system configurations are being used. Ideally the system should gather data from the detector over long periods of time (several days) without interruption. Due to the very large time frames and volumes of data to be collected, errors will most certainly occur and it is important that it is possible to recover from these errors without the need to restart the particular stage of the experiment. It should also be possible to perform this recovery with the need for as little user interaction as possible. The various sub-systems will have their specific requirements for handling errors and errors might have different impact depending on the state of the sub-system or the system as a whole.

Manuscript received May 10, 2007; Revised November 14, 2007.

J.E. Sloper is with CERN, Geneva, Switzerland on leave from the University of Warwick, Coventry (e-mail: john.erik.sloper@cern.ch)

E. Hines is with the University of Warwick, Coventry

G.L. Miotto is with CERN, Geneva, Switzerland

III. Requirements

There are a number of main requirements that have guided the general design of the ER system:

- **Error handling** - The ER should be able to react to errors reported in the system, analyze them and take appropriate actions to ensure that the system returns to or stays in an operational state.
- **Customizable** - It must be possible to ensure that different behavior will take place in response to similar, or indeed identical, problems arising in different sub-systems. For example a dying application within one of the computing farms might have no overall effect on the system other than a slight reduction in computing power. However, if an application in the ReadOut System (ROS) dies, this will have considerable impact on several other sub-systems.
- **Configurable** - It must be possible to easily change the behavior of the recovery system. The TDAQ system is under development and it must therefore, when completed, be able to accommodate for new components. Even in the final system there will be a relatively large number of different configurations to run. It must therefore be possible to configure the ER system to accommodate any needs that may arise.
- **Abstraction** - The ER should be such that any parts can be replaced without the need to change any other components. For example applications should be able to report errors through a fixed interface without knowing anything about the mechanisms of the recovery system. This will also help to minimize the need for change in user implementation. This is very important because the ER system is being introduced relatively late in the development of the TDAQ system.
- **Performance** - The ER should be able to analyze the errors and reach a decision within reasonable time span, usually in the order of seconds. The longer these decisions take, the longer the TDAQ system will remain in an error state causing a loss off data taking capability.

IV. Design and Implementation

The ER system is closely related to the TDAQ RunControl system and naturally design and implementation choices are

guided by this. The following sections will first give a brief overview of the TDAQ system, including some of the most important components. We will then move on to consider the RunControl system in detail before discussing specific design and implementation choices for the ER system.

A. The TDAQ System Structure

Applications in the TDAQ system are organized in a tree structured manner. In addition there are a number of services which are provided using a classic client-server model. Communication is realized using a dedicated Inter Process Communication (IPC) package based on CORBA [2]. The most important services in the context of the recovery system are the Message Reporting System (MRS) and the Error Reporting Service (ERS). The MRS is a service for passing messages between different applications using a subscription-notification model. The ERS provides several services, including a common format and a fixed range of severity levels for all errors reported in the TDAQ system. The ERS relies on the MRS in order to pass error messages between different applications.

B. Configuration Database

A common database stores the configuration of the TDAQ system. It contains everything from command line parameters to the overall organization of both hardware and software connections. The database defines a set of segment objects, typically representing a subsystem or a collection of applications with similar functionality, e.g. a set of Readout modules. Each segment contains a set of applications, resources and other segments. The configuration database is available to all applications through a database server.

C. TDAQ RunControl

The TDAQ RunControl system is responsible for distributing commands from the operator(s) throughout the system. It starts, stops and monitors all applications in the TDAQ system and ensures that the system is in a coherent state. To synchronize operations Finite State Machine (FSM) principles are used. Fig. 1 shows the FSM used for the TDAQ system. In addition to the states shown it can also go into an error state indicating that the application cannot continue its function.

The RunControl is constructed using the configuration database with controllers arranged in a tree structure where each controller is responsible for a segment. Normally commands are only sent to the topmost controller and are then propagated throughout the control tree. Interaction with the RunControl is performed through a graphical interface which among other things displays the RunControl tree, including the current state and any errors. Fig. 2 shows the logical layout of the RunControl.

D. Design

There are two main parts of the dynamic error recovery system; a *local* unit and a *global* unit. The local unit is integrated with each controller in the control tree. It has

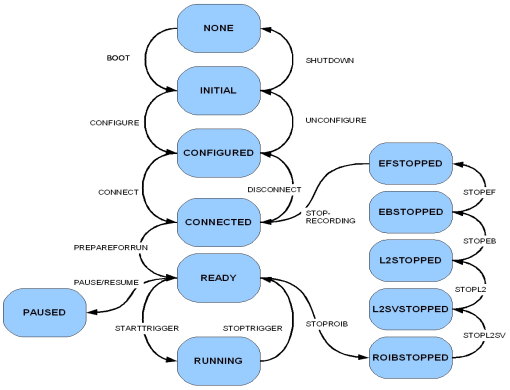


Fig. 1. The TDAQ FSM

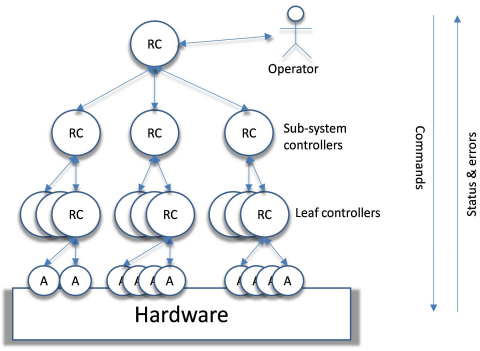


Fig. 2. A TDAQ control tree

a complete 'map' of the applications within its controller's segment. Any changes in the applications status are reported to the ER system by the controller. The main goal of the local recovery unit is to handle errors that can be dealt with at a segment level. That is, errors that don't have an immediate effect on the rest of the system. The local recovery unit receives information directly from the controller it is integrated with. It will then analyze any problems taking into account information such as the configuration, system state, other errors and so on. It can set the error state of the controller, but will also be able to perform more advanced actions such as restarting applications or notifying other applications. In addition to receiving errors directly from the controller it can also receive errors directly from applications. All errors gathered by the local unit will be reported to the global unit, including information such as whether the action has been taken and if the problem has been solved or not.

The global unit handles errors that have a system-wide impact, where applications in different segments will be affected by the recovery actions. It should also be able to take actions based on information received from the local units. The server keeps track of all errors in the system, including the ones reported from the local units. This allows for more general decisions to be made, based on parameters such as frequency of errors within a segment and/or the system as a whole.

Both the global and local unit has access to the configuration

database and uses it to build maps of the connections in the system, read default recovery actions, which tests are associated with the different applications, etc.

In addition to handling errors from the TDAQ system, the ER must also be able to handle errors from systems outside of TDAQ such as the Detector Control System (DCS), networking, farm monitoring tools and so on. This is realized by integrating a proxy application into the control tree and passing error messages through the proxy. A detailed description of this interaction in the DCS case can be found in [3].

The ER system is designed to interface with related components such as the Diagnostics and Verification System (DVS) [4]. This allows it to actively test components in the system. This is especially useful in cases where the actual fault is not immediately apparent. For example if an application is reported not to be responding the recovery system can test the network connections or the host of that application to see if the error is in fact a hardware problem. There are well defined tests for different hardware objects and applications which can be used by the recovery system to properly identify the problem.

Due to the distributed nature of the system it is in most cases not practical to perform a synchronous recovery. Even though the system does support synchronous communication between applications, the ER is designed to perform the recovery in an asynchronous manner. There are several reasons for this. First of all to ensure the abstraction requirement is fulfilled, it is better not to have a direct connection between applications which are reporting errors and the ER system. Also due to time constraints it might not be practical for some applications to wait for an answer from the ER system before continuing its operation. For example, an application might report that another application is not responding, but it may still be able to continue its own operation.

E. Implementation

The two parts of the system are implemented in a similar way. Fig. 3 shows a diagram describing the building blocks of both the local and the global ER units. The main difference between them is that the local unit is directly integrated with a controller and can communicate directly with it. The global unit on the other hand is a standalone server completely outside the control tree. The global ER server also has a simple interface both for reporting errors to it (used by the local recovery systems) and for retrieving information about errors that have been reported and actions taken. This is mainly used for monitoring purposes, but can be a useful tool for a human expert as a help to identify problems in the system.

Both the local and the global ER units rely on a rule-based expert system, also known as a knowledge based system, to analyze errors and decide on appropriate recovery actions. The main advantages of a rule-based expert system are that it is simple to implement in the first place and that changes and additions can be easily made as the need arises. It is very difficult to predict a priori all the different errors that might occur and what appropriate actions should be taken. It is therefore very important that the expert system can be easily changed and customized as more data is gathered and a better

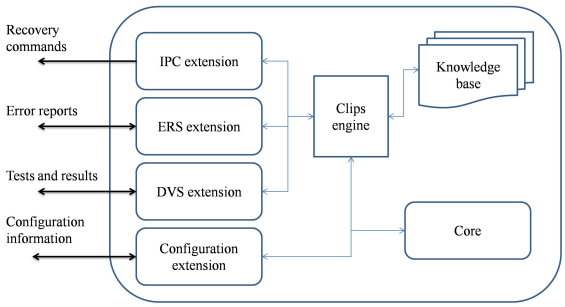


Fig. 3. General structure of both local and global ER units

understanding of the system is gained through experience. The main disadvantage with using a rule-based system is that it can be difficult to manage such systems as the size of the knowledge base increases. Experience suggests that when the number of rules grow it becomes increasingly difficult to keep track of the impact each change will have on the system. This not withstanding we find that with careful implementation this problem can be avoided and the flexibility of the approach makes it a good choice for our system.

The expert system is being implemented using CLIPS [5]. The CLIPS technology was chosen for several reasons. First of all it has been used for many years in the ATLAS experiments, both in the controllers [6] and in the DVS framework, and there exists therefore considerable experience among the developers. Another reason is the ease with which the system can be extended and interfaced with C++ which is the main language used in TDAQ development. C++ extensions are used to interface the CLIPS environment with other parts of the system such as IPC, ERS, DVS, etc.

The CLIPS system consists of a knowledge base and an inference engine. The knowledge base consists of a number of text files where expert knowledge can be encoded as IF-THEN rules. CLIPS also supports procedural and object oriented programming and this is used to build a class hierarchy representing proxies of the applications and hardware in the system. These proxy objects are created dynamically using the information from the configuration database and the information about the objects can then be used in the recovery rules.

CLIPS parses the knowledge base at runtime. This allows the behavior of the ER unit to be easily customized by supplying different sets of files ,describing the knowledge base and rules, as arguments to the application. Fig. 4 shows an example of a simple knowledge base rule. This simple rule detects the case where an application has died and it notifies the associated supervisor and the controller to ignore the application from now on.

We will now look in detail at the different steps of the recovery procedure and how each step is handled and implemented in the ER system:

```
if
system state is running, and
application App1 status is absent, and
application App1 has supervisor S1, and
application App1 membership in
then
notify S1 ignore App1
notify controller ignore App1
set membership App1 out
```

Fig. 4. Sample rule

1) Error Detection: The recovery system itself does not do any direct error detection. It relies on the applications themselves and the controllers to notify whenever an error occurs. However the recovery system might perform tests as a consequence of reported errors. These tests may then discover problems that have not been reported directly.

2) Error reporting: Applications report errors to the recovery system using the ERS. This system is being used for all error reporting in the TDAQ system, and there is therefore no specific need for changes to be made for these applications. However, to ensure that all errors are sent using the same format a set of error classes is defined to be used for specific types of errors. The recovery system (both local and global) subscribes to a set of these classes, depending on its configuration. The applications reporting the error do not need to have any direct knowledge of the recovery system, but merely reports the error using one of the predefined classes. The error can then be picked up anywhere in the system and handled appropriately. Note that the recovery units in different segments might subscribe to different classes of problems and some classes are only handled by the global unit and will never be picked up by the local units and vise versa. One example is a faulty front-end driver. This is known to have impact on several parts of the system and is therefore only picked up by the global server. Errors can also be reported from the local units directly to the global one using the IPC framework. Fig. 5 shows how the global recovery server interacts with the system as a whole.

In addition to the messages reported through ERS, the local unit receives error notifications directly from the controller. This includes notifications such as dead applications (in that segment), errors, timeouts, etc. It also receives updates about the state of the applications and the controller and can use this information to decide on what action to take.

3) Error Analysis: There are two main types of errors that can occur in the TDAQ system. The first type is transient, meaning that the error will not necessarily persist in the system if a full restart is performed. Most software errors/failures are of this type. The other type consists of non-transient errors which will be present even if the system is restarted. Most hardware errors are non-transient and therefore need to be dealt with differently. In many cases it is likely that hardware errors will not be reported, but might cause a software error instead (e.g. an application cannot contact a machine due to a malfunctioning cable). It is therefore important that the recovery system is able to recognize as many of these cases as possible. The DVS is very useful in this respect allowing the recovery system to automatically test a piece of hardware as a consequence of certain error reports. Also, the configuration database provides some information about how to deal with different errors. The configuration database defines a default behavior in case of an application dying, going into error, etc. However this is clearly limited as decisions need to be taken based on dynamic parameters such as system state and other errors.

One of the most difficult challenges is handling situations where one error leads to an error in a different part of the system; and so on, creating an 'avalanche' of errors. To solve this it is important to classify which applications are depended upon by others and are likely to cause avalanche errors. One should then try to identify which specific applications will send errors in this case. In this regard the configuration database is extremely important as it must be possible to automatically retrieve information about how the applications are interconnected, what hardware is being used by which applications and so on. Though a human expert might have the knowledge and experience to track down the real error, it is not trivial for a computer program to do the same. It is therefore important to gather knowledge about the consequence of different errors so as to be able to build a system which will be as effective as possible.

The ER system is able to recognize cases where the system is likely to be non-functional and no recovery is possible. In this case the run should be stopped, but the general policy is that there should not be a means whereby the execution of the system will be stopped automatically. Hence the recovery system will therefore need to notify a human operator who can then make the final decision.

4) Recovery Actions: Both the local and the global recovery unit can send commands to all applications in the system through IPC. All controllers and applications share the same command interface which helps simplifying the recovery procedures. The most important commands are *enable* and *disable*. These commands are used to notify affected components whether an application is operational and should be a part of the system or not. For example if a processing application in the trigger system is malfunctioning the recovery system can tell its supervisor and its controller to ignore it using the disable command, effectively removing the application from the system. It can then try to restart the application and bring it back to the correct state. If the recovery actions are successful the ER can notify the supervisor and controller again, this time to enable it.

In the case of a non-transient error the recovery system can, if appropriate, make changes to the configuration database. For example if a connection is known to be malfunctioning it can be disabled in the database so at the next reconfiguration of the system applications will not use it. This type of action is available in addition to the standard recovery procedures.

V. Gathering Knowledge

Gathering knowledge is a crucial point for all expert systems. A large number of developers are involved in the TDAQ

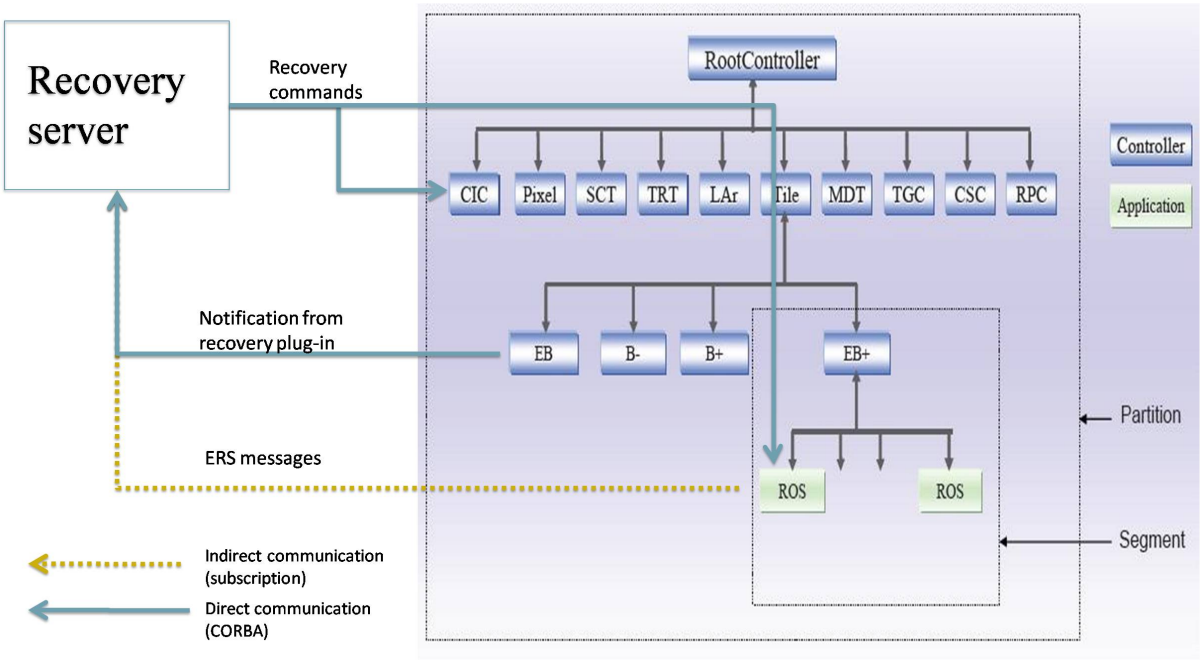


Fig. 5. Overview of interaction with the global recovery server

project making the task of gathering knowledge even more demanding. As a first step the ideas for the new recovery mechanism and especially the reporting mechanisms, have been presented at several ATLAS and TDAQ meetings and conferences in order to make sure the sub-system developers are aware of the system.

Due to the size and complexity of the system, recovery analysis and actions must be customized for each sub-system. The knowledge of what errors might occur and what actions to take are often held by the individual sub-system experts. A series of discussions have been held with some of the sub-system detector experts to try do define the most effective recovery behavior possible. Even though the knowledge base is written in a human-readable language it is still too complex for a regular user to modify it. All changes to the knowledge base should be done by a system expert.

As for any computer program it is very difficult to prove its correctness and this is extra complex with rule-based programs where the execution is done by the inference engine. Testing and feedback from users is therefore an invaluable means by which to both ensure that the system behaves as intended and to further develop the knowledge base. The system is regularly run on test-beds using some standard configurations. Additions to the knowledge base are tested using these setups before being made available for use in the experiment.

VI. Performance and Scalability

In a system of the size of TDAQ scalability can be a real issue. As the local recovery units are mapped to each controller they will scale with the control tree as the system grows. The global unit on the other hand is implmented using a single instance per partition. However, as the recovery unit deals with errors there is a natural limit to the number of messages that need to be processed. If error messages are too frequent it is unlikely that any recovery is possible. Performance is still an issue as the recovery mechanism should be able to analyze the errors and reach a decision within reasonable time span, that is within a few seconds after the error has been reported. The longer these decisions take, the longer the TDAQ system might be nonoperational. In addition the likelihood of an error causing an avalanche throughout the system increases as time goes by. This may be due to timeouts on connections or multiple requests to the component in error and so on. Also as the knowledge base grows the execution time of the CLIPS inference engine will increase. Regular performance tests will therefore be performed to ensure that an acceptable level of performance can be maintained.

If performance proves to be a problem the global unit can be divided into several units which perform dedicated tasks. This will both reduce the number of errors to be handled by each unit and also reduce the size of the knowledge base and hence increase performance. However, one must be careful to do this for well separated tasks or one would loose the benefit of having a server with a global view of the system.

VII. Conclusions

While the dynamic error-recovery system is still in the early stages of development the current experience of the system is positive. Several sub-systems have started defining recovery scenarios and customized recovery units have been designed. It is already possible to keep the system running in cases where it previously would have had to be stopped or manually fixed by an operator.

The extendible design and the use of a knowledge based system means we are able to add or change the behavior of the recovery system in a simple way without affecting other

components. This is very important, especially since many of the error scenarios are not yet identified or documented and as feedback is received additions, modifications and improvements must be made.

VIII. Further Work

The Error recovery system is a work in progress. Focus over the next year will be concerned with expanding functionality to include more recovery scenarios for dealing with failures in all the sub-systems. As feedback is received the system will be modified, improved and new rules will be added to the knowledge base. There is also an ongoing investigation to try to identify related technologies which may play a significant role in the development of the system. For example fuzzy logic is being investigated as a possible means via which to deal with unknown errors and other such situations.

Acknowledgment

The authors would like to thank their colleagues in ATLAS TDAQ for their continuing input, ideas and feedback throughout the course of the design and development of the system.

References

[1] ATLAS Collaboration, "Atlas high level trigger, data acquisition and controls technical design report," 2003.

[2] "OMG, CORBA." [Online]. Available: http://www.omg.org/corba

[3] V. K. A. Poy, J. Sloper and G. Miotto, "Operation of the atlas experiment: Organization of the detector controls and the data acquisition system," in *IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference of the IEEE Industrial Electronics Society*, Paris,France, nov 2006, pp. 190 – 194.

[4] A. Kazarov, A. Corso-Radu, G. Lehmann-Miotto, J. Sloper, and Y. Ryabov, "A rule-based verification and control framework in atlas trigger-daq," *Nuclear Science, IEEE Transactions on*, vol. 54, pp. 604 – 608, 2007.

[5] "CLIPS: A tool for building expert systems." [Online]. Available: http://www.ghg.net/clips/CLIPS.html

[6] D. Liko et al, "Control in the ATLAS TDAQ system," in *Computing in High Energy Physics and Nuclear Physics*, Interlaken, Switzerland, sept-oct 2004, p. 159.

A RULE-BASED VERIFICATION AND CONTROL FRAMEWORK IN ATLAS TRIGGER-DAQ

A. Kazarov, A. Corso-Radu, G.Lehmann Miotto, J.E. Sloper[1], CERN, Geneva, Switzerland
Yu. Ryabov, Petersburg NPI, Gatchina, Russian Federation

1 on leave from University of Warwick, Coventry, England

Abstract

In order to meet the requirements of ATLAS data taking, the ATLAS Trigger-DAQ system is composed of O(10000) of applications running on more than 2600 computers in a network. With such system size, s/w and h/w failures are quite frequent. To minimize system downtime, the Trigger-DAQ control system shall include advance verification and diagnostics facilities. The operator should use tests and expertise of the TDAQ and detectors developers in order to diagnose and recover from errors, if possible automatically.

The TDAQ control system is built as a distributed tree of controllers, where behaviour of each controller is defined in a rule-based language allowing easy customization. The control system also includes verification framework which allow users to develop and configure tests for any component in the system with different levels of complexity. It can be used as a stand-alone test facility for a small detector installation, as part of the general TDAQ initialization procedure, and for diagnosing the problems which may occur during the run time.

The system is currently being used in TDAQ commissioning at the ATLAS pit and by subdetectors for stand-alone verification of the hardware before it is finally installed.

The paper describes the architecture and implementation of TDAQ control system with more emphasis on the new features developed for the verification framework, features requested by users during it's exploitation in real environment.

INTRODUCTION

The paper describes components of the Run Control and Verification framework, the core of the ATLAS Trigger-DAQ (TDAQ) Control system, one of subsystems building the TDAQ system .The key feature of these components is that they are based on the expert-system technology and they should help the TDAQ operator to minimise system down time and to control it smoothly by using the expertise of the system developers.

In the first part we describe the motivation, the design principles, the architecture and some details of the implementation of DVS (Diagnostic and Verification System), Run Control and Setup components. In the second part, DVS component is described in more details including recent developments and examples of its usage for ATLAS commissioning.

DESIGN AND ARCHITECTURE OF RULE-BASED CONTROL COMPONENTS

Motivation and objectives

ATLAS (A Toroidal LHC ApparatuS) Trigger-DAQ system [1] transfers and filters data from the detector front-end electronics to the mass-storage for the offline analysis. It is composed of a big number of hardware and software components, as summarized below:

- 1800 read-out VME boards
- 1800 fiber links
- 150 ROS PCs each hosting 4 ROB-IN cards
- 500 LVL2 PCs
- 90 SFI PCs
- ~2000 EF PCs
- ~30 SFO PCs
- ~50 infrastructure PCs (file servers)
- ~200 Ethernet switches
- and O(10000) applications running on the listed above hardware

The T/DAQ control system shall be capable to smoothly control this number of various components, meeting some requirements [1]. The size of the system under control and the number of components make the probability of a failure very high. The typical failures we are facing every day are PC components failures and applications crashes. Keeping in mind the life-time of the experiment (around 10 years) and the high cost of its down-time and as well the fact that the system will be operated by a non-expert shift crew, it is very important to have system verification, failure diagnostics and recovery facilities embedded in the Control system of the TDAQ. These facilities should allow the Operator to

- Detect problems as early as possible by means of probing the system.
- Make use of the system's developers expertise (knowledge).
- Automate verification of a large system.
- Minimize system down-time using recovery procedures based on problem diagnosis.

Framework approach

Another important aspect of the TDAQ system which makes important contribution to the design, is the geographical decentralization of the experiment and a big number of users all over the world participating in the system development. In order to build the full-scale TDAQ system its software will be used by all ATLAS subdetectors each providing specific functionality, whereas the 'core' software should join all pieces together and guarantee their coherent functioning. This brings the idea of the 'core' framework components providing dedicated services and well-defined interfaces for users and developers.

Design principles

To fulfil the above requirements it was decided to design the system following these principles:

- *Framework* approach: a system shall be configurable and extensible by the experts and users also during the experiment life-time.
- *Expert system* approach: the system's behaviour is described in a rule-based language to allow accumulation of expert's knowledge and an easy adaptation to changing conditions.
- *Hierarchical distributed* architecture of the Run Control system reflecting the tree structure and the scale of the experiment.

Architecture

On the UML diagram below (Figure 1) the actual architecture of the ATLAS Control subsystem is presented.

It is composed of a number of modules or components, implementing particular functionality and interacting with other components via public interfaces. In this paper we

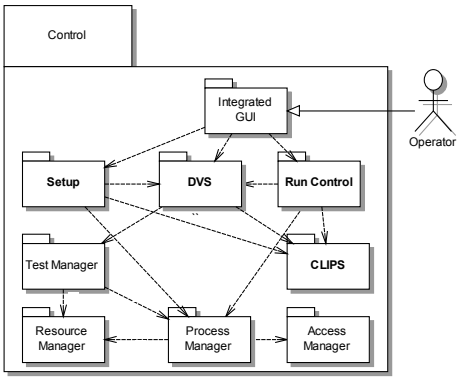


Figure 1. Architecture of the Control subsystem

briefly describe Run Control, DVS and Setup components, all sharing expert system technology provided by the CLIPS package.

Both Setup and Run Control components use verification functionality provided by DVS framework in order to verify the relevant subset of the system under control.

The TDAQ Operator interacts with the system via a human-friendly Integrated Graphical User Interface.

Control subsystem interacts with other T/DAQ subsystems, especially with Configuration and Monitoring ones (not shown on the figure for simplicity).

Run Control

Run Control is a distributed framework allowing each subsystem in TDAQ to define the behaviour of a particular controller and to join all controllers in a distributed tree structure, in order to guarantee synchronous and homogeneous execution of Operator commands through the whole system. The top-level Run Controller represents the status of the system to the Operator and accepts his commands, while leaf controllers deal with applications which directly control the TDAQ hardware. Intermediate controllers represent statuses of different subsets of the T/DAQ.

Each controller is responsible for:

- distributing commands to its children
- receiving children's statuses and analysing (diagnosing) errors
- determining its own state
- undertaking possible recovery actions
- passing status and error information to its parent

The general picture of the Run Control tree is presented in Figure 2.

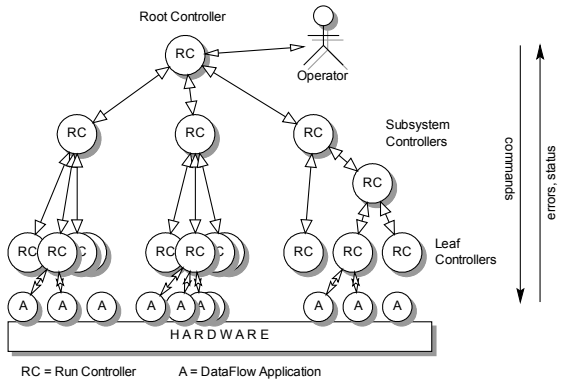


Figure 2. Tree of Run Controllers

Each controller is an application running on some machine in the system. The controller implements a pre-defined Finite State Machine skeleton (common for all controllers), defining its all possible states and transitions; and a small expert system (i.e. an expert system engine and a set of rules or a Knowledge Base, KB) defining his behaviour. The details of the implementation are described in [2]. The default rules included in the framework implementation are very simple, like: *"if all my children controllers are in state 'A', then change my state to 'A'"*, or *"if just one o my children is in 'error' state (e.g. timed-out in transition, reported an error or become unavailable), then change my state to 'error' "*

More sophisticated recovery rules should analyse all available data and make decisions like disabling a sub-tree, executing pre-defined recovery actions or reporting an unrecoverable error to the parent controller, so it could take over the problem.

The concrete recovery policies is to be defined when the final system will be put in operation so the framework should allow easy adaptation of controller's behaviour during experiment run-time.

DVS (Diagnostics and Verification System)

DVS is a framework which allows to:

- Configure a test for any component in the system
- Have a testable view on the particular configuration of a system in a user-friendly GUI
- Automate testing of the system
- Make diagnostics conclusion in case of a problem detected during testing (provided some knowledge put in the Knowledge Base)

More details on DVS are presented in the following section and in [3].

Setup component

Setup component is a 'boot-strap controller' for the initial infrastructure of a particular TDAQ configuration. It brings the system to a state where it can accept RC commands. It uses DVS to verify in depth system's h/w, used by the particular TDAQ configuration used for the current run. At this stage, specific tests developed by particular subsystems are executed in order to detect potential problems and to confirm the system's integrity before launching any process. If some tests fail, user is provided with some options, like *'ignore and continue'*, *'retry'* or *'abort'*. For the final system some automatic recovery rules should be defined.

Setup KB contains additional rules to start, restart and verify applications and diagnose related problems. Most of the infrastructure applications have back-up capabilities, so they are restarted automatically if they crash or even if the host machine goes down. In the latter case a back-up host is chosen and applications restarted there.

The functionality of applications under supervision is also confirmed by the execution of tests.

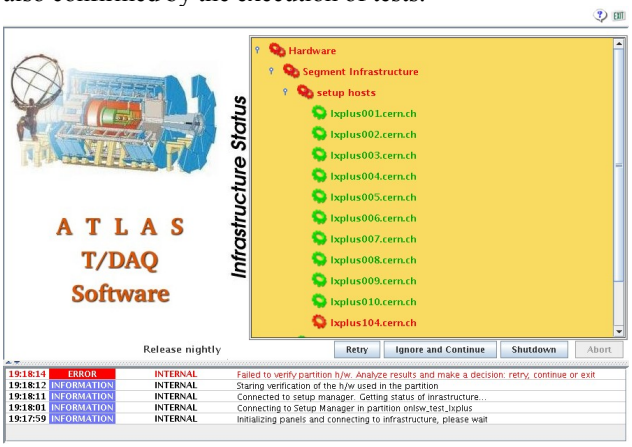


Figure 3. Setup GUI window with hardware infrastructure being tested.

On the Figure 3, the initial GUI window of T/DAQ software is shown. On the right side one can see setup panel with the tree of h/w components (a rack of PCs in this case) being tested, where one PC did not pass test.

CLIPS: an expert system shell

The behaviour of the Run Control components which are mostly data- and event-driven is described naturally in terms of rules, rather then in a procedural language. It is also necessary to have an easy way to update the knowledge about behaviour of the system during experiment lifetime, saving expert knowledge and detailed information about recovery procedures applied by the experts. So it was decided to use expert system technology as the base for all core run control components. Some evaluations ([2]) have been done, and CLIPS expert system shell [4] was selected. It can be shortly characterized in the following:

- CLIPS stands for '**C** '-**L**anguage **I**ntegrated **P**roduction **S**ystem
- Produced by NASA
- Free, open (written in 'C') and well- documented
- Embeddable in other s/w products as a library
- Programming capabilities: rule-base programming paradigm (rules and facts), OO language (classes and objects), conventional procedural constructs

DVS: AN OVERVIEW AND RECENT DEVELOPMENTS

Use Cases

On the Figure 4, the use cases for DVS framework are presented. First, a TDAQ Expert (typically a developer of the system) contributes to the framework by developing specific tests and providing some diagnostic knowledge. Afterwards, TDAQ Operator can reuse Expert's expertise by verifying any subset of the system and diagnosing detected errors. The verification functionality is also used by non-human bodies, like Controllers in order to check the functionality of the subsystem under control.

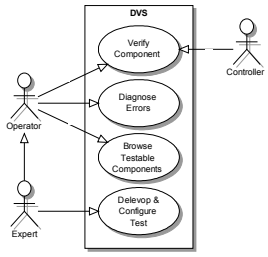


Figure 4 Use Cases for DVS

Architecture

Architecture of the DVS framework is shown on Figure 5. Its core components are the Test Repository database describing all test, and expert system filled with some knowledge by an expert. The functionality is available to end-users via C++ and Java API and via a user-friendly GUI.

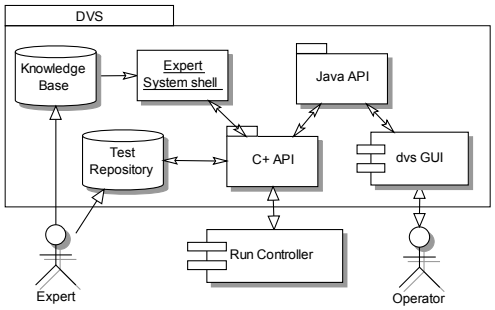


Figure 5 DVS architecture

Tests configuration

A test is a binary running on a particular host in a system. It verifies a particular functionality of a TDAQ component and returns a single result value: PASSED, FAILED, UNRESOLVED, TIMEOUT and some text output. For a single component a number of tests can be associated which can be organized in sequences, executed synchronously or asynchronously. Tests and their relationships are fully described in a database. The following attributes are available for test description:

parameters, host, dependencies, timeouts, scope, complexity and mask. Any test can be associated either to a particular object in the configuration database or to allobjects of a particular type. In the latter case test's parameters and host can be parametrized by attribute values of the concrete objects when the test is launched.

In Figure 6, right side, database editor windows are shown, one with the list of tests defined in the configuration and another one with the parameters of a particular test.

GUI

On the Figure 6 you one can see DVS GUI in action. In its left side the tree of testable components is displayed. User can select a single component or a group in the tree and run all defined tests by clicking a single button. As tests finish, components icons change colour reflecting the result. On the right panel the test output is dumped.

Recent developments

DVS was developed and made available for the end users in 2003 [3]. A number of improvement requests appeared as constant feedback from the users. Many of them were implemented in recent T/DAQ s/w releases:

- Tests *levels* and *masks* for more precise test selection which allow to promptly configure test repository without editing the database. User can select a subset of tests with a particular level of complexity or satisfying some regular expression.
- Asynchronous and synchronous mode for execution of tests for complex objects.
- Test *scope* to prevent conflicting tests from being executed when system is taking data.
- Tests *verbosity* can be defined globally in runtime.
- Test's *runtime output* for long-running tests: user can see output of a test as it executes in real time, even before it finishes.
- Test output produced by all children components can be combined in one panel and then saved in a file.
- New type of interactive tests, called '*actions*', were introduced to allow users to execute more complex test scenarios requiring some input. This also allows to reuse already existing console utilities in the framework. An action is configured as a test, but it is launched in a separate terminal window.

USE OF DVS FOR ATLAS COMMISSIONING

DVS is currently used by different ATLAS subdetectors and T/DAQ subsystems in the ATLAS commissioning activities. A number of tests were developed for different detector and DAQ components, so they can be extensively tested before being installed at the final position. One good example is 'MobiDAQ' system developed by the Tile calorimeter group [5]. Defferent tests for detector front-end electronics and for ROD (Read-Out Drivers) VME modules were developed. On Figure 6 you can see DVS GUI with a number of detector tests being executed.

Another set of tests was developed for ROB-IN PCI modules commissioning.

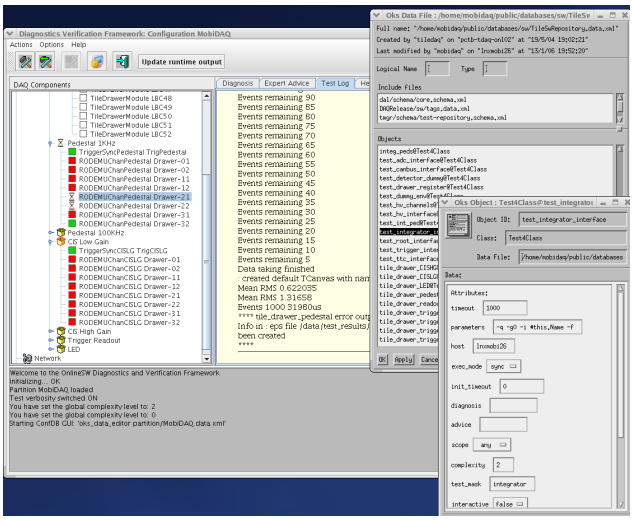


Figure 6. DVS GUI and MobiDAQ test repository in configuration DB.

LARGE-SCALE TESTS

All Control components are a subject for scalability tests, regularly performed on a cluster of PCs. Papers [6] and [7] describes results of the tests in details.

REFERENCES

[1] ATLAS High Level Trigger, Data Acquisition and Control TDR, CERN/LHCC/2003-022 (2003), http://cern.ch/atlas-proj-hltdaqdcs-tdr

[2] D. Liko et all, "Control in the ATLAS TDAQ system", CHEP 2004 Interlaken Switzerland, September 2004

[3] A. Kazarov et all, "Verification and Diagnostics Framework in ATLAS Trigger/DAQ", 2003 Computing in High Energy and Nuclear Physics, La Jolla, Ca, USA, March 2003

[4] CLIPS Expert System Shell, http://www.ghg.net/clips/CLIPS.htm

[5] MobiDAQ web page: http://atlas.web.cern.ch/Atlas/SUB_DETECTORS/TILE/Commissioning/mobidaq/HowTo.htm

[6] ATLAS HLT/DAQ Software Large Scale Tests March 2004, ATL-D-TR-0001, EDMS ID: 499884

[7] D. Burkhart et. all. "Testing on a large scale: Running the Atlas Data Acquisition and High Level Trigger software on 700 pc nodes", CHEP 2006, Feb 2006, Mumbai India.

Operation of the ATLAS Experiment: Organization of the Detector Controls and the Data Acquisition System

Alex Barriuso Poy, John Erik Sloper, Viatcheslav Khomutnikov and Giovanna Lehmann Miotto
European Organization for Nuclear Research (CERN)
Geneva 1211
SWITZERELAND
alex.barriuso.poy@cern.ch

***Abstract* — The overall control of the ATLAS experiment includes the supervision of the hardware in the experimental set-up and the common experimental infrastructure, as well as the supervision of all processes involved in the event readout. This functionality is provided by two independent, although complementary and interacting systems: control for Trigger and Data AcQuisition (TDAQ), and the Detector Control System (DCS). The size and complexity of both systems suggest a hierarchical structure as the natural way to organize the control of the detector. The hierarchical control of both systems is driven by two independent Finite State Machines. This paper mainly discusses the hierarchical control of ATLAS, how the use of FSMs is approached, and the coordination between the two systems to ensure a coherent operation. In addition, the interfaces to DCS and TDAQ that will be used in the future ATLAS control room are shown.**

I. Introduction

ATLAS [1] is a general-purpose particle detector designed to study p-p collisions at the Large Hadron Collider (LHC) at CERN, which will start operation in 2007. It will be the largest particle detector ever built, distributed over a cylindrical volume of 42 m length and 11 m radius. It is composed of three detector systems, the tracker, the calorimeter, and the muon system. These are divided into 12 different specialized sub-detectors that perform different tasks such as track reconstruction and particle identification.

The data from the detector is read out through the Trigger and Data AcQusition (TDAQ) system which is a large and heterogeneous system and contains a wide variety of components to be controlled. These items are typically clustered according to the detector topology. They range from readout modules connected to the detectors to nodes in the computer farms used for data selection. The Run control (RC) for TDAQ is in charge of controlling the hardware and software elements involved in the data taking process. The RC is built in a hierarchical and distributed manner.

On the other hand, the Detector Control System (DCS) [2] supervises the hardware in the experimental set-up including all the detector services (e.g. high voltage, cooling) and the common experimental infrastructure (e.g. racks, environmental conditions). DCS also serves as interface to external systems such as the CERN technical services (e.g. electricity, ventilation) and most notably to the LHC accelerator (e.g. for beam conditions and backgrounds). The DCS consists of a distributed Back-End (BE) running on PCs and of the Front-End (FE) instrumentation. The associated data volume to be treated by DCS is very large, in total, around 200.000 channels will be supervised. The magnitude of the ATLAS DCS, in terms of system complexity and collaboration effort, suggests also a hierarchical structure as the natural way to organize the detector control.

The interaction between TDAQ and DCS is handled by the DAQ-DCS Communication (DDC) software package [3].

This paper discusses the hierarchical control of ATLAS using finite state machines (FSMs) and the coordination during the operation between TDAQ and DCS. It presents also the top level user interface to monitor and control the detector conditions. Section II describes how the FSM concept is applied in the two major systems for governing ATLAS, DAQ and DCS. Section III discusses the coordination during operation of both systems with focus on the DDC package. The interfaces between the hierarchical control and the operators on shift are discussed in section IV.

II. The Finite State Machine Applications

The two major systems for governing the ATLAS detector, TDAQ RC and DCS, are based on the same concept of FSM. This allows for the sequencing and automation of operations. Even though both systems are based on the same concept, they have different requirements and use different technologies for implementing it. The TDAQ control is governed by a single global FSM for all components in its control tree. This guarantees that the same sequencing is followed by all its different components during the read out process. In contrast, the DCS is composed of many different systems with different behaviours that need to be automated, for example High Voltage (HV), Low Voltage (LV), cooling, etc. As a result, the DCS control is composed of many FSMs arranged in a hierarchy with rules of the parent – child interaction defined.

1-4244-0136-4/06/$20.00 ©2006 IEEE

Some parts of the detector should operate continuously since any interruption could be very costly in time and money or may even be detrimental to the performance of the detector. Hence supervision by the DCS is needed at all times. The TDAQ system in contrast is required only while physics data are being taken or during specific monitoring, calibration, or testing runs. Therefore the DCS system must be able to run completely independent of the TDAQ.

An essential requirement of both, DCS and TDAQ systems, which is particularly important in the commissioning and installation phases, is the ability to partition the system into several independent, but fully-functional subsets. It must be possible for several detectors and/or several parts of a given detector to be triggered and to take data in parallel and independently of each other. Thus, to allow partitioning while keeping interaction between the two systems, at a certain level, both hierarchies are a mirror of each other. This simplifies the communication between the systems as the TDAQ RC can always interact with a corresponding part of the DCS.

A. The TDAQ Run control hierarchy

The TDAQ RC system is responsible for initialization and supervision of the full TDAQ system. This encompasses starting/stopping of processes, the distribution of commands given by the operator (human user) and monitoring and handling of any errors and faults that may occur. The RC system implementation is based on the CLIPS programming language [4] with heavy use of C++ extensions. CLIPS was chosen due to its flexibility and previous positive experience with the language. More information on the evaluation and choice of technologies can be found in [5].

The TDAQ system is naturally divided into sub-systems (sub-detectors, event building farms, etc). The system configuration is therefore divided into *segments* typically representing a sub-detector, crates of readout modules or similar. A segment contains a set of applications, and possibly, other segments. The top-level segment is called a *partition* and may or may not represent the entire TDAQ system. The RC builds a tree of controllers where each controller corresponds to a segment. The controllers are then in charge of all the applications contained by the segment (as specified in the configuration database) and possibly other sub-segments. Figure 1 shows an example of a simple control tree.

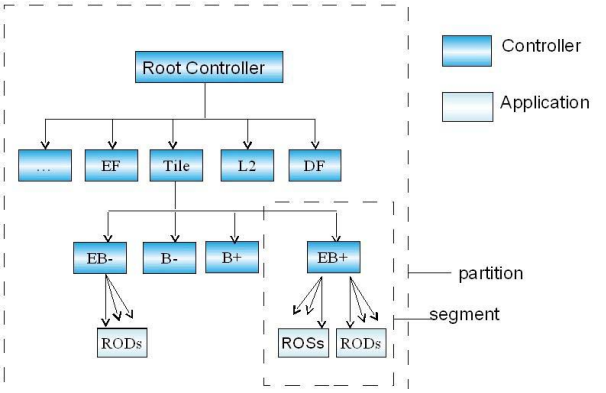


Fig. 2: A simple TDAQ control tree showing the concept of partition and segments

To keep a status of all the applications in the control tree and to ensure a coherent execution of commands a Finite State Machine (FSM) has been implemented for the RC system. The applications are subdivided into two groups:

1. State aware applications. These follow the state machine and receive and confirm commands from their controller. Examples: ReadOut System (ROS), Event building applications, etc.
2. Stateless applications. These are the applications that run independent of the state machine, but do still belong to a controller and report any errors to it. Example: monitoring applications, etc.

All controllers and state-aware applications follow the same global FSM. This is necessary to ensure that certain systems are started and stopped in a well-defined way and that dependencies across the system are taken into account. To allow for a more flexible structure, the possibility of hidden sub-states for any controller (and thus its sub-tree) has been implemented in the system. Figure 2 shows the state machine currently used for the experiment.

Whenever a controller receives a transition command it sends it to all the applications and controllers in its segment. Any child controllers propagate the command and so on throughout the

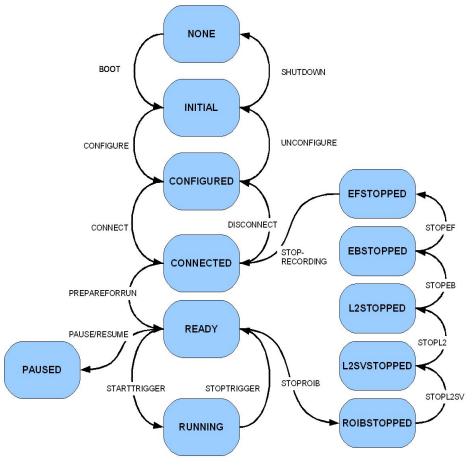


Fig. 2: The state machine currently used for the TDAQ system

system. A controller confirms the transition only when all the children have confirmed the new state, so any errors while performing a state change is effectively propagated to the root controller.

B. The DCS control hierarchy

The commercial Supervisory Control And Data Acquisition (SCADA) package PVSS-II [6] has been chosen by the Joint COntrols Project (JCOP) [7] at CERN to implement the BE system of the four LHC experiments.

The FSM tool forms part of a software framework developed in the context of JCOP and it is based on both PVSS-II and SMI++ (State Management Interface) [8]. SMI++ is a tool for developing control systems and it is based on the FSM concept. Complex systems can be broken down into simple FSM units that are hierarchically controlled by other FSMs. The detector can be decomposed and described in terms of SMI++ objects, which behave as FSMs. These objects can represent device entities, like a pump or a high-voltage crate, or logical groups of such devices, like a sub-detector or a gas system. Each object can automatically make decisions based on changes in its own internal status and on those of other components in the hierarchy.

The architecture of the BE system is organized in three functional horizontal layers [9] and FSM engines run in order to ensure a coherent operation of the whole detector (see Figure 3).

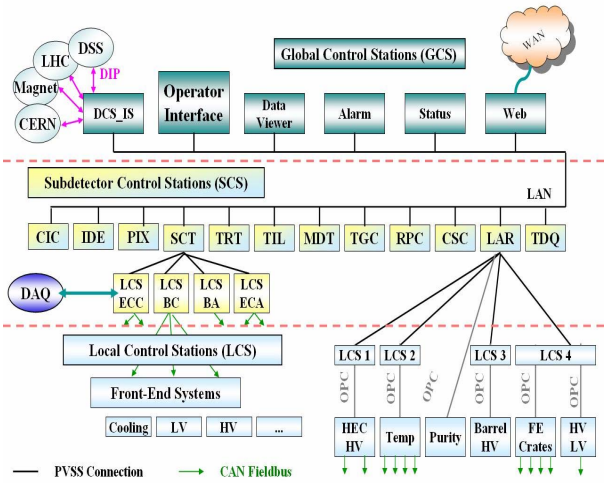


Fig. 3: DCS BE Architecture

At the top layer, there will be Global Control Stations (GCSs) which are in charge of the overall operation of the detector. They provide high level monitoring and control of all sub-detectors, while data processing and command execution are handled at the lower levels. The GCS will be able to access all stations in the hierarchy. In section IV the main operator interface to the control hierarchy is introduced.

The Sub-detector Control Stations (SCSs) form the middle level of the hierarchy. Each sub-detector has its own station and an additional one handles the Common Infrastructure Controls (CIC). Each SCS allows the complete local operation of a sub-detector by means of dedicated graphical interfaces (see section IV). At this level of the hierarchy, the connection with the TDAQ system takes place in order to ensure that detector operation and physics data taking are synchronized. The interaction between DCS and TDAQ is described in more detail in section III.

The bottom level of the hierarchy is made up of the Local Control Stations (LCSs), which handle the low level monitoring and control of instrumentation and services belonging to the sub-detector. The LCSs execute the commands received from the SCS at the layer above, but also trigger predefined actions autonomously following the FSM procedure.

III. INTERACTION DAQ-DCS DURING OPERATION

Interaction between TDAQ and DCS is needed in order to enable the coherent operation of the experiment as a whole. The control interface between DCS and TDAQ is arranged with the assumption that the latter is the master when the experiment is taking data. Thus, the TDAQ control applications are able to drive the DCS by sending commands and getting feedback about the result. Furthermore, the DCS should inform asynchronously TDAQ about any failure on the detector preventing incorrect data taking.

The interaction is done by means of the command transfer application DDC-CT of the DAQ-DCS Communication (DDC) software package [3], after called the DDC controller. Like any other leaf controller, the DDC controller is responsible for receiving commands from the parent TDAQ controller and transferring these commands to the domain under its control, which is a DCS subsystem. It also receives error signals from the corresponding DCS FSM and passes them to the TDAQ. The mapping between the TDAQ transition commands and the FSM commands on DCS is defined by the DDC controller configuration [3].

The number of DDC controllers running for a sub-detector corresponds one-to-one to the number of Trigger and Timing Control (TTC) partitions [1] of that sub-detector [10]. A TTC partition is the minimal part of a detector that can be controlled autonomously for data taking. By running one DDC controller for each partition we achieve the finest common possible granularity to both systems. On the DCS control hierarchy each DDC controller corresponds to a DDC Device Unit (DDC_DU) (see Figure 4) [11]. The DDC_DU uses the FSM functionality to execute the

commands sent by TDAQ and inform about a change of state.

The interaction between RC and DCS is performed at the supervisory level; there is no direct interaction between TDAQ and the DCS front-end. A typical example would be when the DAQ operator sends the command "prepare for run" to a certain sub-detector or part of it. This command implies a set of actions to be performed at the DCS side by means of the FSM (i.e. goto_ready means ramp up HV, switch on LV, etc). The assignment of DAQ commands to the relevant DCS commands is done through the DDC controllers; the DDC_DU applies these commands within the DCS hierarchy of FSMs. Moreover, in case of any failure that could damage the quality of the data taking (i.e. error while ramping up HV), DCS at the TTC level in its hierarchy informs TDAQ through the DDC mechanism.

From the TDAQ system point of view a DDC controller is treated as a normal leaf controller in the control tree. It accepts all transition commands and follows the global TDAQ FSM. If a DDC controller enters an error-state (for example due to a problem within the DCS) this state is propagated up the TDAQ control tree.

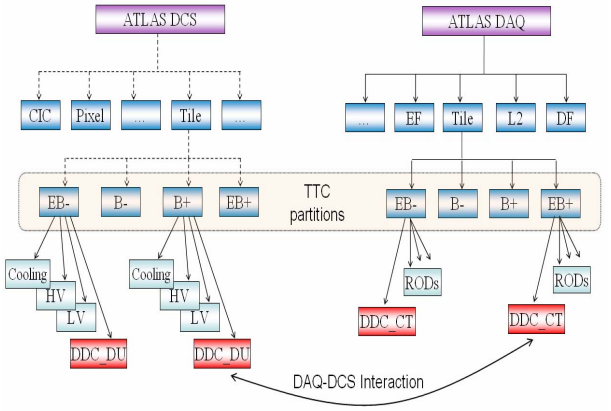


Fig. 4: Interaction TDAQ-DCS

IV. INTERFACE FOR THE OPERATION

The ATLAS experiment will be controlled from a central control room. A large number of displays and interfaces will be available in the control room including the interfaces to TDAQ and DCS. The TDAQ interface gives access to the RC, but also interfaces with a variety of other systems necessary for the operator. The DCS system on the other hand gives an in-depth view of the DCS and provides tools and functionality to monitor and operate the system and also locate, identify and fix errors that may occur. As mentioned in Section III, the TDAQ controls the DCS while data taking. The two interfaces might be located at different stations within the control room so efficient coordination is therefore of importance both for security reasons and to obtain high-quality data. An access management system will be put into place to ensure that contradictory actions are not taken (e.g. the DCS operator changes the voltage set while data taking).

A. DCS Operator Interface

The ATLAS Operator Interface (OI) is the principal human-machine tool for control of the detector equipment and related infrastructure. It interfaces to the hierarchical control of the DCS. Each node in the hierarchy corresponds to a process controlled by an FSM that has an associated display called *workspace* (i.e. SCS, HV, LV or a cooling system). In total, around 1200 different workspaces will be accessible from the OI.

In order to support the operator in controlling the detector, maintain integration of the displays, and support efficient navigation, a frame integrating the displays of the control hierarchy with navigation facilities, has been developed [11]. This is the ATLAS OI (see Figure 5).

In order to navigate across the control hierarchy, a main and a secondary module are available. Thus, the operator can keep an overall view of the detector while studying in detail a problem deeper in the hierarchy. In addition, alarms and messages are displayed in two separate modules. Given the appropriate access rights, the operator is able to command the entire detector. From the OI the operator can send commands to the children in the hierarchy and include/exclude a certain part of the control hierarchy. For instance, the shift operator in could exclude a certain part of the detector that has a problem (i.e. a full barrel, the HV system, the cooling system, a single sensor, etc), and then, the control can be taken by an expert for intervention. After solving the problem the expert returns the control to the operator).

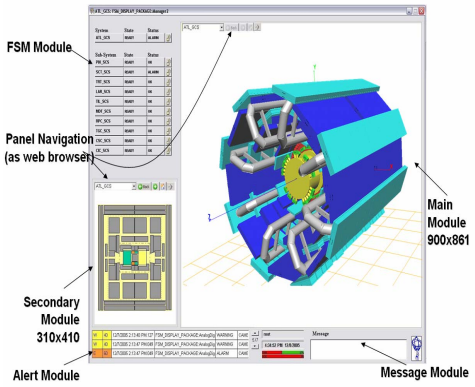


Fig. 5: Common OI Layout

B. TDAQ Operator Interface

The Integrated Graphical User Interface (IGUI) is the main interface to the TDAQ system. The IGUI is a JAVA based application and provides the operator with

all tools needed to successfully control and monitor the data taking. The main functionality is to configure the TDAQ system and to send commands to the RC, but it also provides a series of monitoring and other necessary functionality. The interface is based on tabbed browsing, and a framework is provided to easily create new tabs for the display. This is used to present different configuration screens, to interface with other TDAQ systems (such as message-reporting) and, by monitoring groups, to integrate their displays with the IGUI. Figure 6 shows a screenshot of the IGUI.

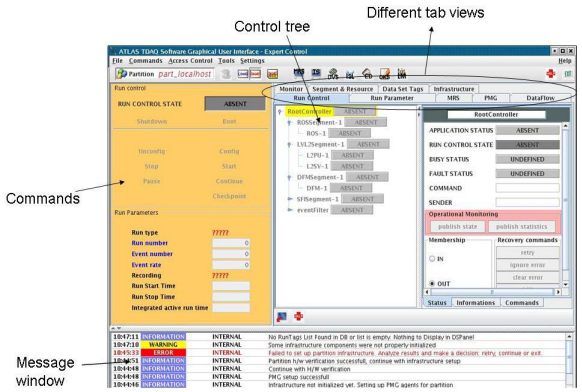


Fig. 6: The TDAQ IGUI

V. Conclusions

This paper has shown the organization, implementation and coordination of two of the main systems for the ATLAS detector, namely TDAQ and DCS. We have looked closely at the use and implementation of the FSM concept in both systems and pointed out similarities and differences between the two.

The ATLAS detector, together with the DCS, is currently being built (first operation will take place in 2007) and the top level OI has been in use since February 2006 with satisfactory results. The first control hierarchy included was the Common Infrastructure Controls (CIC) and the integration of the rest of sub-detectors will be realized in the coming months. For the commissioning period the OI has been installed both, underground in the electronics room next to the equipment, and in the ATLAS control room.

The TDAQ RC has been operational for several years, though it is still undergoing development and testing. It is being used throughout the commissioning of the ATLAS detector. However, the interaction with the DCS has naturally been limited, but becomes increasingly important as the detector nears completion and is controlled through the DCS. Close interaction between DAQ and DCS will be of prime importance both for safety reasons and in order to obtain good-quality data.

References

[1] ATLAS Collaboration, "ATLAS: Detector and Physics Performance Technical Design Report", CERN/LHC/99-14, TDR 14 and 15, 1999

[2] Boterenbrood H. et al., "Design and Implementation of the ATLAS Detector Control System", IEEE Real Time Conference, Montreal, May 2003

[3] Khomoutnikov, V., "ATLAS DAQ-DCS Communication Software, Users Guide", ATL-DQ-ON-0008 June 2006

[4] D. Liko et all, "Control in the ATLAS TDAQ system", CHEP2004 Interlaken, Switzerland, September 2004

[5] CLIPS, A tool for building expert systems, *http://www.ghg.net/clips/CLIPS.html*

[6] http://www.pvss.com/

[7] http://itco.web.cern.ch/itco/

[8] Franek, B., Gaspar, C., "SMI++ - Object Oriented Framework for Designing Control Systems for HEP Experiments", CHEP 97, Berlin, Germany, April 1997

[9] Barriuso A., Burckhart H., "Hierarchical Control for the ATLAS Experiment", ICALEPCS 05, Geneva, Switzerland, October 2005.

[10] Khomoutnikov, V., Barriuso A., Burckhart H., Subdetector Control Interaction with TDAQ Controls, ATL-DQ-EN-0027, July 2006

[11] Barriuso A., "FSM Integration Guidelines", ATL/DQ/ON/0010, December 2005

The ATLAS DAQ System Online Configurations Database Service Challenge

J. Almeida
Faculdade de Ciências - Universidade de Lisboa, Edifício C8, Campo Grande, 1749-016 Lisboa, Portugal

M. Dobson, A. Kazarov[1], G. Lehmann Miotto, J.E. Sloper[2], I. Soloviev[1] and R. Torres[3]
CERN (European Organization for Nuclear Research), Geneva 23, CH-1211, Switzerland
1 – on leave from PNPI (Petersburg Nuclear Physics Institute)
2 - on leave from University of Warwick
3 - on leave from UFRJ (Universidade Federal do Rio de Janeiro)

Abstract – **This paper describes challenging requirements on the configuration service for the ATLAS experiment at CERN. It presents the status of the implementation and testing one year before the start of data taking, providing details of:**

- **the capabilities of the underlying OKS object manager to store and to archive configuration descriptions, its user and programming interfaces;**
- **the organization of configuration descriptions for different types of data taking runs and combinations of participating sub-detectors;**
- **the scalable architecture to support simultaneous access to the service by thousands of processes during the online configuration stage of ATLAS;**
- **the experience with the usage of the configuration service during large scale tests, test beam, commissioning and technical runs.**

The paper also presents pro and contra of the chosen object-oriented implementation compared with solutions based on pure relational database technologies, and explains why after several years of usage we continue with our approach.

I. Introduction

The configurations database service described in the paper is a part of the ATLAS High-Level Trigger, Data Acquisition (DAQ) and Controls system [1]. It is used to provide the overall description of the DAQ system and partial description of the trigger and detectors software and hardware. Such descriptions cover the control-oriented configuration of all ATLAS applications running during data taking (including information such as: which parts of the ATLAS systems and detectors are participating in a given run, when and where processes shall be started, what run-time environment to be created for each of them, how to check status of running processes and to recover run-time errors, when and in what order to shut down running processes, etc.) and provide configuration parameters for many of them (overall DAQ data-flow configuration, online monitoring configuration, connectivity and parameters for various DAQ, trigger and detector modules, chips and channels).

The configurations database is used by many groups of developers from various systems and detectors. Each group is responsible for preparing their own part of the configuration and for writing the code configuring their applications using common configuration service tools.

The configuration service is accessed simultaneously by thousands of processes during the boot and configuration phases of the run. If the configuration is changed during a run, the configuration service is responsible for notifying and reconfiguring affected processes.

And finally, the configuration service archives all configurations used during data gathering, so they can be browsed later by experts and accessed by programs performing events processing.

The use cases and requirements listed above for the configuration service result into several challenges discussed in the following section.

II. Configuration Service Challenges

A. Data Model

The database schema used by the configuration service for descriptions mentioned in the previous section includes approximately three hundreds classes. The classes are designed by different DAQ, trigger and detector groups. To speak in a common language across all of them, it is desirable to have a so-called *core* database schema describing common data types, which are extended by the groups. In the resulting schema all classes are interconnected via inheritance and relation links.

As an example of the schema extension, consider the *core* Application class, which contains several tens of generic properties to describe parameters of a process such as unique identity, command line parameters, various timeouts, actions to be taken in case of errors, environment variables, the application's binary file and host, initialization and shutdown dependencies, etc. Then each group takes the base Application class and extends it providing their specific properties e.g. trigger groups derive TriggerApplication classes to add specific parameters, the data-flow groups derive DataFlowApplication classes having data-flow specific parameters, etc. The important thing is that the instances of all these classes can still be considered as applications and be handled in the same manner, e.g. by the code of the control

1-4244-0867-9/07/$20.00 ©2007 IEEE.

software.

So, the first challenge requires the configuration service to support a sophisticated data model able to describe complex, extendable and interconnected pieces of data.

B. Data Access Programming Interface

The configuration data are accessed by code written by developers from different groups. In many cases the code has to retrieve the same configuration information and it is important that this is done in a consistent way. This becomes essential, if calculation of the configuration information requires several accesses to the database and the access to further parameters depend on the results of previous accesses. For example, to calculate some configuration parameters one has to execute a database query; then an algorithm is applied to the data returned by the query and the algorithm's result is used as parameters for the next query, etc. Or, the configuration information is a result of calculations on top of the data returned by several independent queries. Ideally, such code should be implemented as a function available to any developer and the queries themselves should not be used by developers explicitly.

As well, most of the developers who need to read the configuration information stored in the database, are not database experts. It is therefore desirable that the code to access the database data is completely hidden behind some high-level API of the programming language they are using. This allows users of the configuration service to effectively develop their application code instead of spending time on learning the low-level API of the database implementation and debugging various problems caused by the inefficient usage of the database or by database version changes.

The mechanism used to hide the low-level database implementation and to provide a configuration data access API using data types of a programming language is called the data access library (DAL). A DAL maps database data types on the types supported by the programming languages (e.g. relational table is mapped on C++, Java or Python class). A DAL is also responsible for instantiating database data as instances of the programming languages types (e.g. for each row of the relational table it creates a corresponding C++ object) or to save any changes to the database, if the instances have been modified or new ones have been created. A DAL can also provide functions (called the DAL algorithms) to calculate data based on the result of several database queries, as it was explained in the previous paragraph.

A DAL can be written by hand or automatically generated from the database schema. The latter is useful, if the database schema contains a large number of classes, or it is simply changing often. The DAL generation is absolutely necessary to implement the ATLAS DAQ configuration service because of the high degree of complexity of the schema, its parallel development by several groups and the frequent changes during development cycles. For example, the configuration schema defined by the ATLAS DAQ–HLT (high level trigger) development release was changed at least once per week during last year.

Thus, the second challenge requires that the configuration service provides automatically generated DALs for programming languages used by the ATLAS software (C++, Java and Python).

C. Database Population

To prepare a configuration description for a data taking session one has to put into the database all the data describing the software releases, the online infrastructure, the data-flow, monitoring, trigger and detector applications, the hardware they use, trigger and detector specific parameters; furthermore all elements need to be organized into a controllable hierarchy. Even a minimal configuration for the online infrastructure test is composed of hundreds of configuration objects. It is expected, that the final configuration for the complete ATLAS will contain in the order of ten thousand configuration objects. It is clear that the database describing such configurations cannot be filled manually. Special tools to create, compare or copy configurations and to test their completeness and consistency should therefore be provided.

At the same time, many configurations prepared for different types of runs (cosmic, physics, calibration, etc.) and for different sets of participating trigger and detector parts should co-exist.

To avoid duplication of information belonging to different co-existing configurations it is reasonable to share common configuration data. Note, the sharing of configuration data not only saves the space used by the database, but also improves maintenance of the data in case of changes, since modifications will have to be done only once. This approach is first of all applicable to the software releases descriptions, which are updated only when new releases or patches are installed. In a similar way, the description of all hardware elements needs to be updated only when there are corresponding changes in the physical world such as installation of new computers, changing cables between modules, network reconfiguration, etc.

Many parts of configuration data can be and have to be generated automatically. For example, the description of a software release can be generated by analyzing it's installation area and CMT requirements files[1] used for the build. It contains all binary files with their supported platforms, the run-time environment they need (e.g. paths to shared libraries including used external packages, their specific environment variables, etc.). Another example is the generation of the hardware description. This can be done analyzing the ATLAS technical coordination installation databases containing information about installed hardware organized by ATLAS systems and detectors, or even by reading simple ASCII file containing list of hosts. Based on this information the configuration tool checks the state of the installed hardware, extracts missing parameters (e.g. network or module addresses) and saves the configuration description. Note, that

[1] CMT is a tool to build ATLAS software releases; the requirements file is an analogue of a make file

the automatically generated data described in this paragraph can be shared between configuration descriptions for different runs.

The rest of the configuration data should be built using tools which have a detailed knowledge about the architecture of the ATLAS DAQ, trigger and detector systems. Such tools can be used for different purposes, e.g. to prepare a database configuration to test DAQ read-out or event builder systems on test beds of the ATLAS experiment, to test high-level trigger algorithms with pre-loaded event data on a computer farm of an external institute, to test newly commissioned hardware at the experiment site, or to describe the full ATLAS configuration. A generation tool will on the one hand be used by experts, who need the capability of redefining any configuration parameters they would like to test; on the other hand it shall also be usable by non-experts, who only know the type of configuration they want to run: for those such a tool needs to be able to set default configuration parameters.

Summarizing, the third challenge requires the configuration service to provide a set of tools for automatic generation of configuration descriptions for the software releases, for various hardware installations and for the different data taking configurations.

D. *Performance and Scalability*

One of the goals of any ATLAS online service is to minimize the experiment's downtime during the physics data taking period. The configuration service is used by thousands of processes during the boot and configuration stages of the data taking, and it has a significant influence on the overall time needed to setup a data taking session. By this reason it has to assure a scalable architecture to guarantee that this time does not depend too much on the number of clients of the configuration service. An acceptable time requirement for us would be that if a single client gets the configuration data in a few tens of seconds then all clients should be able to do the same in less than one minute.

Another important requirement is effective usage of the configuration service in case of small modifications of settings between data taking runs. In this case the affected clients should be able to read from the configuration service the changes since the previous run instead of re-reading the complete configuration: this is especially true for clients which are re-reading an unmodified configuration.

Most of the developers using the configuration service are not database experts. One cannot guarantee that they are using the service in an optimal way, e.g. their code is reading any configuration parameter only once. In addition, the configuration data can be read by several libraries developed by different groups and used within a single process, which even more increases probability of code accessing the service in non-optimal way. So, to achieve optimal performance the configuration service has to support caching of configuration data on the client's side and to read the data from the service only when this is really needed.

The forth challenge requires the configuration service to have a scalable architecture and performance in order to meet the demanding experiment's requirements. In case of small changes between runs the service has to support partial reconfiguration. As well, whenever possible, the service has to cache information at the level of client's process to prevent multiple requests to the configuration service for information which was already read.

E. *Archiving*

Once the configuration data were used for data taking, they have to be safely archived by the configuration service. The archived data can be used later for processing of event data, to be browsed by experts or to be retrieved for preparing a new configuration. The configuration service has to guarantee, that once archived, the data will not be removed or unintentionally corrupted, e.g. during a modification of the configuration for the next run. This requirement addresses the fifth configuration service challenge.

F. *Easy Usage*

The configuration service is a part of the TDAQ software releases. They are used not only at CERN, but also in many ATLAS collaborating institutes around the world. To be used the configuration service has to be easily available there. In many cases the institutes cannot use the CERN-based remote configuration service because of various reasons (e.g. slow or unreliable network or even lack of internet connections for certain test-beds, various local software policies) and therefore need to support the configuration databases themselves. At the same time one should not expect, that there will be knowledgeable database administrators or simply advanced database users at any site.

The last configuration service challenge is the requirement to be easily accessible or installable for all ATLAS collaborating institutes.

III. Configuration Service Implementation

The trigger and DAQ system and its predecessors (CERN R&D 13 group [2], ATLAS DAQ Prototype-1 Project [3]) had evaluated several shareware and commercial candidates for the configuration service, including persistent object managers, object and relational databases, but no system satisfying all the requirements was found [4]. Therefore it was decided to use as a prototype the existing object manager OKS [5] and make it capable of fulfilling the DAQ configuration service needs.

Initially, the OKS has used the Rogue Wave object persistency [6] storing objects in cross-platform binary files. At that time the Rogue Wave library was used as the base general purpose library within the DAQ system. The library was then replaced by the C++ Standard one and instead of Rogue Wave persistency the OKS started to use human readable XML files (the possibility to easily browse, modify and distribute these files was one of the key points of its success). Then, to satisfy the DAQ needs the OKS was extended to implement remote access, to provide an abstract

API layer using several OKS access implementations and to realize the OKS database archiving. These features are described in more details in the rest of this section.

A. The OKS Persistent Object Manager

The OKS is a set of tools to provide objects' persistency. It is based on the following object data model:

- the basic entity is an object with unique *object identifier*;
- objects with common properties and behavior are described by a *class*, defining *attributes* (primitive types), *relationships* (links with objects) and *methods* to act on the object properties;
- classes support *inheritance* (multiple inheritance is allowed) with *polymorphism* (overloading of inherited object properties in a child class);
- *composite* objects (i.e. a parent object built from dependent child objects via aggregation relationships);
- *integrity* constraints (i.e. type and value restrictions on attributes and relationships).

The OKS classes and objects can be created and modified dynamically, put into a persistent storage and read back. The native OKS API for this is C++.

For effective data selection there is a query language. OKS allows active notification on data changes as well (i.e. call user callbacks when an object is created/deleted/modified).

The OKS supports several advanced functions to work with persistent schema and data: the *schema evolution* allows modifications to the schema as the user applications evolve, and the *data migration* permits data to be accessed by successive versions of a schema.

The main persistent storage for OKS is XML files. There are two types of them: the *schema* ones define classes and the *data* files store database objects. An XML file can include other files to build a complete database from several well defined files and to share them between different OKS databases.

Another persistent storage for OKS data are relational databases. OKS uses the LCG CORAL [7] library that allows transparent usage of several relational database back-ends such as Oracle, MySQL and SQLite. This type of storage is oriented for archiving purposes: to check-in OKS files, to browse archives, and to check-out archived files. The archiving supports incremental versioning to store only differences between a complete base version of a database and its last version. As an example the modification of an object's attribute will be stored in the database as a new row of relational table and not as a complete copy of object. This feature drastically reduces space used by OKS archives.

OKS provides distant access to the databases via remote database servers (RDB), developed on top of CORBA [8]. The RDB server is used to access a single database from different clients running on computers without a common file system. It also helps solving scalability problems, when access is required by a huge number of clients, by caching the results of OKS queries.

B. OKS User Utilities

OKS provides several graphical applications. The schema editor is UML-like graphical editor to browse and to modify the database schema, see Fig. 1 as example:

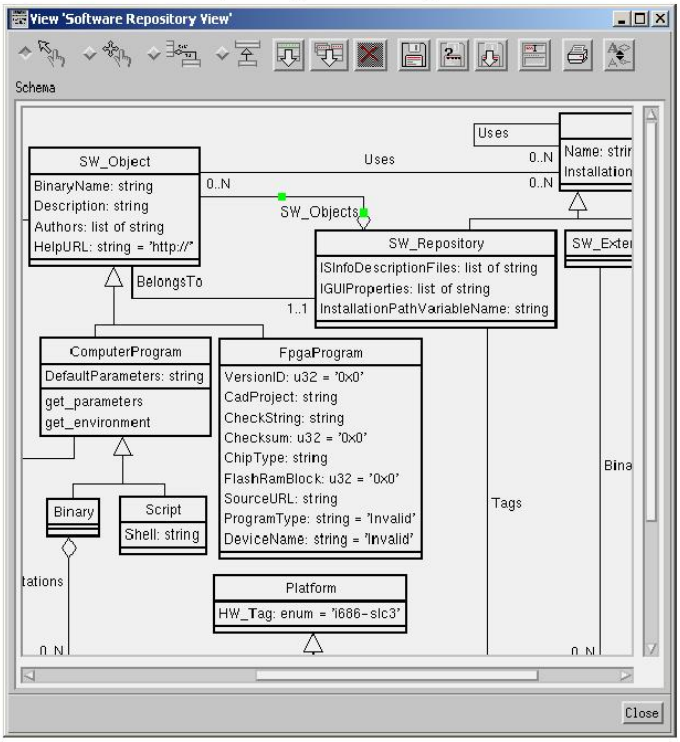


Fig. 1. Example of OKS Schema Editor view.

The data editor is a graphical editor to browse and to modify the database data. It allows customizing appearance of the user-defined views presenting relations between objects. An example is shown on Fig. 2. Alternatively the data can be accessed in a tabular format.

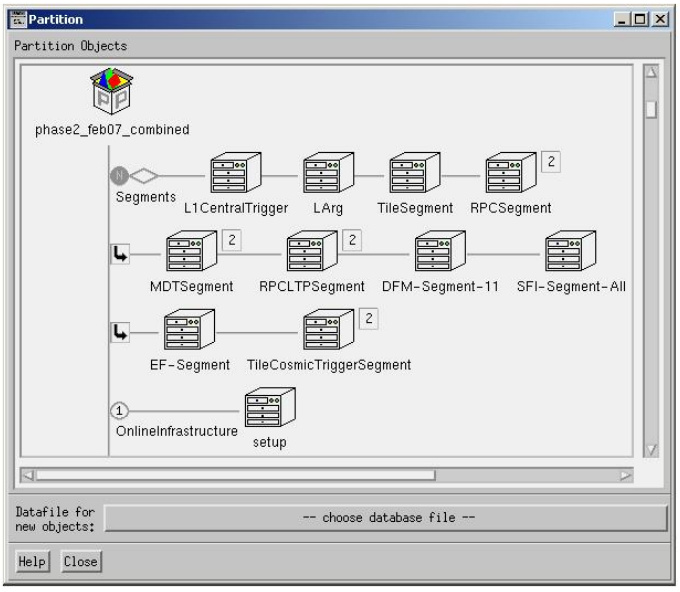


Fig. 2 Example of OKS Data Editor view: a fragment of partition object for ATLAS February 2007 Phase 2 commissioning run.

The data editor has a graphical query constructor. Once a query is constructed, it can be saved and used for future data retrieval using the editor or OKS API.

The OKS provides utilities to print out the contents of a database, to compare schema and data files, to merge databases, to insert and to extract the OKS databases from the archive and to list its contents. The OKS archive Web interface is shown on Fig. 3:

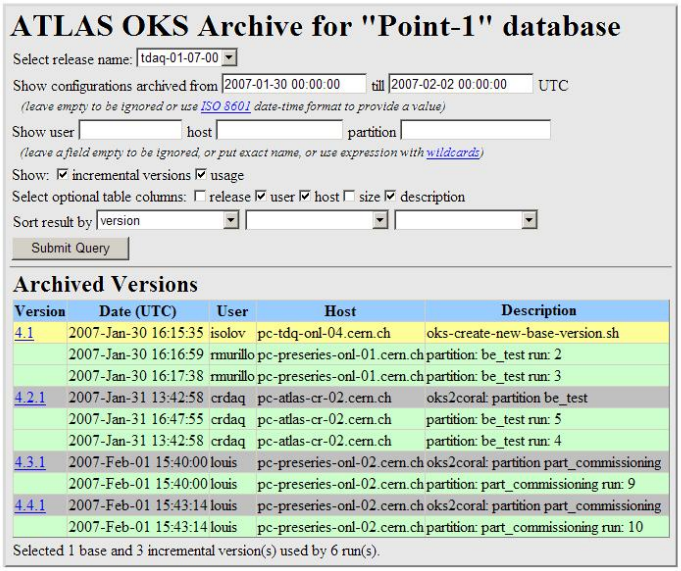


Fig. 3. Example of Web interface to OKS archive

One can select the archived data by release name, by time intervals, and by user, host and partition masks. The result can be presented with different level of details and sorting policies. The archived data can be compared and extracted from the archive as OKS data files.

C. *Configuration Service Programming Interface Layers*

The code of ATLAS programs never uses the OKS or RDB APIs directly. Instead there are two layers of configuration service API to avoid any dependencies on the database implementations.

The first *config* layer provides an abstract interface to work with databases and to access configuration objects. This interface exists for C++, Java and Python programming languages. The implementations of this interface are available as plug-ins for OKS XML files, OKS relational archives and for the RDB server. The user's code does not depend at all on the implementation used. This layer is used to work with an arbitrary database schema for reading its description, loading or creating new databases, querying them to get configuration objects, accessing their properties and for subscribing on database changes. The *config* layer is normally used by the ATLAS control and infrastructure applications which are working with not known at compilation time user classes.

The second *DAL* layer uses the above abstract config one to map database schema on C++, Java and Python classes and to instantiate the database data as appropriate objects of such classes. When necessary, user-defined methods can be inserted into DAL classes to implement sophisticated algorithms on top of the config objects. The DAL is automatically generated from the OKS schema for the above mentioned programming languages by the *genconfig* tool, which is a part of the configuration service. The generated DALs are used by all ATLAS online processes which need to get the configuration description.

Fig. 4 presents the relations between databases interfaces and users of the configuration service (the Partition Maker tool is used to generate configuration descriptions and it will be described in one of following subsections).

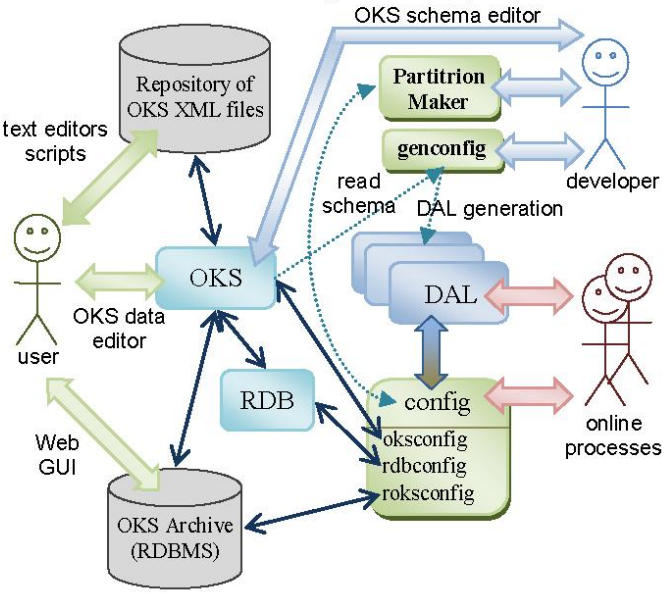


Fig. 4. Configuration service interfaces and users

As mentioned, the user's code does not depend on the database implementation; in the most frequent case of DAL usage it deals with high-level generated classes only. An example of such code is shown in Fig. 5. The generated classes and their methods are shown in bold.

```
  // (1) initialize configuration object
Configuration db("oksconfig:partitions/test.xml");
  // (2) find partition executing a DAL algorithm
const Partition * p = db.get_partition("test");
  // (3) read partition's default host
const Computer * h = p->get_DefaultHost();
  // (4) print out hostname
std::cout << "Default host: " << h->UID() << "\n";
  // (5) get pointer on new host
const Computer * h2 = db.get<Computer>("pc-onl-2");
  // (6) set new partition's default host
p->set_DefaultHost(h2);
  // (7) commit database changes
db.commit();
```

Fig. 5. Simplified example of C++ code using generated DAL

Line 1 shows how to create a configuration service object. The argument of its constructor defines the implementation that will be used and its parameters. In this example the oks xml file will be used. If one wants to use RDB or relational archive, just this argument needs to be changed; there is no

need to recompile or re-link the program. Line 2 shows how to use an algorithm. The *get_partition()* one is written by hand and is integrated into the DAL generated class. Line 3 shows how to access another configuration object referenced by a DAL object using the generated method for the relation "*DefaultHost*" defined in the database schema. Line 4 demonstrates how to print the unique identity of a configuration object. Line 5 shows how to retrieve from configuration service an object of a certain class knowing its identity. The line 6 sets new value for the relation. The last line saves changes to the persistent storage, i.e. in our example into the xml file.

D. Configuration Schema

The structure of any configuration object is defined by the database schema. The DAQ defines a common database schema agreed with trigger and detector groups, which extend it to introduce properties of their specific configuration objects. The common schema contains an order of hundred classes and defines several domains:

- The *software releases* including programs, libraries, supported platforms, external software packages they use and the variables they need.
- The *hardware* including racks, computers, crates, modules, interfaces, cables, links, networks and their composition into hardware systems.
- The *control* including applications (an application corresponds to a process to be started under certain conditions; it points to a computer and program and defines their parameters), resources (a hardware object or an application which can be temporary disabled in the scope of given configuration), resource sets (group of closely coupled resources) and segments (an individually controlled container object, that is composed of resources, applications and nested segments).
- The *configuration* including partition object (contains parameters and segments for given run). The partition object is used as the root of the tree of other objects, describing the configuration. Via segments, their nested segments, resources, applications and their relations with parameters the partition object implicitly references other objects participating in the configuration description. This makes it possible to find any piece of configuration data by simply navigating between objects instead of execution of queries. It also makes it possible to pre-load a configuration description into the client's cache and thereby avoid later requests from those clients to the configuration service. This is a clear benefit in regard to performance and scalability issues.

The DAQ data-flow and monitoring groups extend the common schema and define another hundred classes to provide the overall description of their configuration parameters. The trigger and several detectors also extend the schema to introduce classes describing their specific configuration parameters. The total number of configuration classes used for last detector commissioning run was about three hundred.

E. Organisation of Database Repositories

The oks xml files are stored as database repositories. To make a repository visible for configuration service it is enough to add it to the special colon-separated environment variable (like the PATH variable on UNIX). Then an oks file can be referenced relative to the value of this variable.

Each repository is versioned, when the database schema is modified. Usually this happens together with the installation of new software releases.

A repository contains folders assigned to different groups of users. A folder has a predefined structure for schema extension, software, hardware, segments and stand-alone partitions. Each group has write access permissions for its folder and is responsible for preparing segments, which can be inserted into combined ATLAS partitions.

There are several repositories across CERN sites. One of them is dedicated for *development* and is available on the afs file system. Its latest version corresponds to the last release (rebuilt each night). The software and hardware descriptions are automatically regenerated together with the release build.

Another one is the ATLAS *production* repository. It is available at ATLAS experiment site and contains configurations for the detector technical and commissioning runs.

F. Generation of Configurations

To generate descriptions prepared for different tests and participating systems the Partition Maker tool has been developed. It contains embedded ATLAS DAQ system and trigger expert knowledge in order to guide a user through the configuration process, minimizing the risk of configuration errors.

The architecture is organized into three distinct layers, where each level uses resources from lower levels. The lowest layer is responsible for representing trigger or DAQ components as classes and ensures correct configuring of them. The middle level creates segments composed of objects built from previous level and guarantees their correct interaction within each segment. The highest level links together segments and certifies that they result in a meaningful configuration.

The Partition Maker allows the user to work in any of the three levels. Novice users can work in the highest level only, in order to quickly generate a configuration with minimal input information. The users with more experience can use resources from lower levels to prepare specific configurations.

The Partition Maker is written in the Python programming language. This allows easy development of scripts using it and integration of user configuration knowledge.

G. Scalable Online Access at ATLAS Point-1

The programs using configuration service are running on computers inserted into racks, separated by ATLAS systems and detectors. So, the programs belonging to a rack require

similar configuration data. To achieve the best possible performance we are running one RDB server on the rack's local file server (LFS). Such RDB server reads master copy of configuration data from central file server and provides access to them for all processes running inside the rack as shown on Fig. 6:

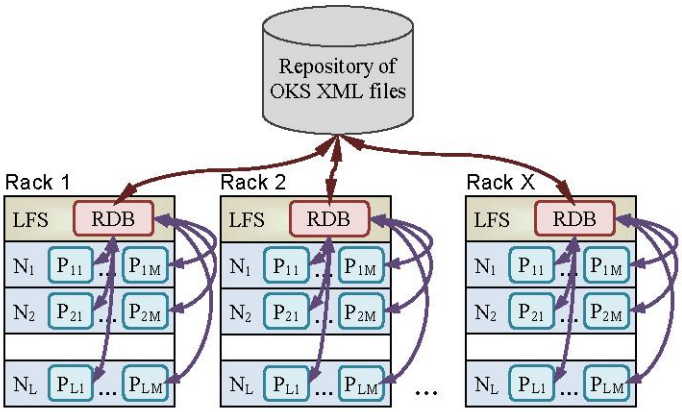


Fig. 6. Schema of the RDB-based scalable access on Point-1

The maximum number of clients served by one RDB is approximately 240 (30 high-level trigger computers per rack, 8 processing tasks per computer). To achieve better performance, the number of clients per RDB server can be reduced by running more RDB servers per rack or using dedicated racks for online services.

H. *Offline Access*

When a new data taking run is started, the configuration data are automatically archived into relational database and relevant record referencing archived configuration is added to the ATLAS conditions database. Later one can access configuration data using programming interfaces or to browse and restore them using graphical user interfaces.

I. *Experience with OKS Usage*

The OKS-based configuration service has been used since the very beginning by the ATLAS online software. It has been exploited in combined test beams [9] [10], large scale tests [11], recent technical and commissioning runs. The service was actively used to prepare the configuration descriptions of the test runs and to provide access to them for the online processes. It always met the functionality and performance requests of the users.

IV. The Current Object-Oriented Approach: Advantages and Drawbacks

In the past we have tested many candidates for the implementation of the configuration service for ATLAS. The object database management systems seemed to be the best suited choice, since they have the data model and programming interfaces satisfying our needs and they supported data versioning that is useful for data archiving. However we have failed to a find scalable and reliable implementation. This was the main reason, why we have started our own persistent object manager prototype project ten years ago. Since then we are keeping an eye on technologies used by other groups and experiments to implement configuration databases. Most frequently such solutions are based on the usage of relational database management systems. This can be explained by several reasons including their prevalence and maturity in the database world, the availability of commercial and freeware systems, a suitable data model for certain types of configuration data. Another important booster for this choice is that relational database specialists are very requested on the employment market also in non-science sectors: this makes their study appealing for young scientists without an already well defined career path. Below we summarize why the usage of relational databases does not give to us benefits comparing with existing solution.

In the beginning we listed several challenging requirements to the configuration service. A pure relational technology does not address any of them and cannot be used without tools developed on top of it.

To populate databases one has to use special utilities. It is unrealistic to put all data by hand for both, the existing OKS-based implementation and the relational one. The prototyping of such utilities is a bit simpler with current implementation, since XML files can be produced by simple scripts, easily taken into account by configuration service tools, editable and removable. In case of relational databases such prototyping requires more complicated actions dealing with relational storage especially if tables are shared with other developers.

The installation and support of relational databases requires a qualified support. Many users from external institutes installing trigger and DAQ releases, whenever possible, are trying to avoid programs using relational databases.

The classical relational data model does not support inheritance playing one of the key roles in the data model used by the configuration service. One cannot create a new table and mark it as derived from an already existing table, so that existing queries would take this new table into account. As well, the relational data model does not support arrays. To implement them one has to create extra tables or encode them as BLOBs, and then provide special functions to calculate values of a "multi-value" column. Without a layer built on top of the relational data model it cannot satisfy our requirements.

The relational databases do not provide a DAL. Instead a developer using relational databases has to learn the SQL language and the database API allowing using it. A DAL for a relational database can only be designed, when the relational data model extension discussed above is known. In our case this means development of own tools or usage of third-party ones, which do not satisfy all our needs (e.g. user-defined algorithms, mapping on *all* used programming languages).

The advanced relational database management systems provide scalable solutions. However even an Oracle farm composed of several servers cannot handle in an acceptable time simultaneous requests coming from tens of thousands clients. To achieve the required performance one has to

develop tools similar to our RDB servers, which serves a set of relational database clients and caches the results of SQL queries for them [12]. Such tools access the relational database server only when a query coming from a client was never performed.

On the client side the results of SQL queries can also be cached. However when the same data or parts of them are retrieved by different SQL queries, such caching mechanism cannot help. This is different from the DAL and config objects created using our configuration service, where the effectiveness of the cache mechanism does not depend on the relations used to access the object.

And finally, the safe archiving of configuration data is one of the most difficult problems arising for users of the relational databases. Since configuration data are changed quite often (e.g. pieces of ATLAS systems and detectors may be enabled / disabled before each new run), it is not enough just to modify existing relational data. In such case their history of usage will be erased. Thus, the relational data used for a run have to be archived and some versioning mechanism has to be provided. Using a true relational model such mechanism cannot be easily implemented, since any single change of data results in the necessity to version all data having relations to it; in turn all data having relations on data versioned during the previous step have to be versioned also and so on. The requirement to archive data complicates the relational tables storing configuration data and especially the tools modifying them. Differently, the OKS archiving provides incremental versioning and any single modification of OKS data results into a single record in the archive (i.e. one new row in OKS relational table describing value of object's attribute or relationship). This is possible due to fact that OKS relational tables used for archiving store values of OKS entities like class, object, and value of attribute or relationship. One should not expect to find in the OKS archive a table with name "Module" or "Application". Their descriptions will be stored into several OKS tables allowing to deal with arbitrary database schemas and their evolutions, and to implement space-preserving archiving.

V. Conclusions

In this paper we have shown how the present implementation of the configuration service for the ATLAS experiment is capable of meeting all the challenging requirements posed to it.

A comparative analysis with other approaches such as the usage of relational databases has been performed showing that, despite the power and maturity of relational databases, it would not be possible to use them directly without implementing several layers on top of them for the purpose of the experiment's configuration. Especially the requirement of providing data access libraries in an environment of still frequently changing database schema and the need for efficient archiving of different configuration versions remain very hard to meet.

Therefore we consider that the choice of basing the configuration service on a homemade persistent object manager, OKS, is still justified and is, at present, the best way to satisfy all requirements put forward by the ATLAS experiment.

References

[1] ATLAS Collaboration, "ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report", - 2003-022, June 2003

[2] G. Ambrosini et al., "The RD13 Data Acquisition System", proceedings of the Computing in High Energy Physics conference, San Francisco, California, USA, 1994

[3] G.Ambrosin, et al., "The ATLAS DAQ and Event Filter Prototype "-1" Project", Computer Physics Communications 110, 1998, pp.95-102

[4] Maria Skiadelli, "Object Oriented database system evaluation for the DAQ system", diploma thesis, Patras Polytechnic School, Greece, March 1994

[5] R.Jones, L. Mapelli, Yu.Ryabov, I.Soloviev, "The OKS Persistent In-memory Object Manager", IEEE Transactions on Nuclear Science, vol 45, No 4, Part 1, August, 1998, pp. 1958 – 1964

[6] Rogue Wave Software, Inc., "Tools.h++ Foundation Class Library for C++ programming", March 1996, for more information see URL http://www.roguewave.com/products/tools/tools.html

[7] CORAL - COmmon Relational Abstraction Layer, a package of CERN LHC Computing Grid Project, see http://pool.cern.ch/coral/

[8] I. Alexandrov, A. Amorim, E. Badescu, D. Burckhart, M. Caprini, M. Dobson, N. Barros, J. Flammerd, R. Jones, A. Kazarov, D. Klose, S. Kolos, S. Korobov, V. Kotov, D. Liko, L. Mapelli, M. Mineev, L. Pedro, Y. Ryabov, I. Soloviev, "Experience with CORBA communication middleware in the ATLAS DAQ", proceedings of the Computing in High Energy Physics conference, Interlaken, Switzerland , 2004

[9] I. Alexandrov, A. Amorim, E. Badescu, M. Barczyk, D.Burckhart-Chromek, M. Caprini, J. Da Silva Conceicao, J. Flammer, B. Di Girolamo, M. Dobson, R. Hart, R. Jones, A. Kazarov, S. Kolos, V. Kotov, D. Klose, D. Liko, J. Lima, L. Lucio, L. Mapelli, M. Mineev, L. Pedro, Y. Ryabov, I. Soloviev, H. Wolters , "Online Software for the ATLAS Test Beam Data Acquisition System", proceedings of the 13th IEEE - NPSS Real Time Conference, Montreal, Canada , 2003

[10] S. Gadomski et al., "Deployment and use of the ATLAS DAQ in the Combined Test Beam", proceedings of the 14th IEEE - NPSS Real Time Conference, Stockholm, Sweden , 2005

[11] D. Burckhart-Chromek et al., "Testing on a Large Scale : running the ATLAS Data Acquisition and High Level Trigger Software on 700 PC Nodes", proceedings of the Computing in High Energy Physics conference, Mumbai, India , 2006

[12] L. Lueking, S. Kosyakov, J. Kowalkowski, D. Litvintsev, M. Paterno, S.White, B. Blumenfeld, P. Maksimovic, M. Mathis, "FroNtier : High Performance Database Access Using Standard Web Components in a Scalable Multi-tier Architecture", proceedings of the Computing in High Energy Physics conference, Interlaken, Switzerland , 2004

Appendix B

Design and implementation of the CGP framework

Following is an high level design and implementation overview of the CGP framework used in this thesis. The program was designed and developed by myself and as always when designing computer software there are a myriad of choices to be made and many ways of accomplishing the same functionality. The design presented here is how I did it and may indeed contain many sub-optimal choices, but is presented in order to give the reader an understanding of the CGP implementation used in the thesis.

The entire program is available in the CD accompanying this thesis and online at www.github.com/jesloper/CGP.

B.1 Overall remarks

The CGP is designed with two main components: the actual CGP implementation and a graphical user interface (GUI) used to interact with the program. I will describe only the CGP part as the GUI is very much a matter of personal choice and preference and has no real impact on the results. This is a truth with some modifications as a good GUI can present results in such a way as to give the operator a better insight into the problem, inspire new ideas, etc hence it can be an important part of a framework. Still now further details of the GUI will be given in the following

document.

Cartesian Genetic Programming software - Design and implementation

John Erik Sloper, CERN/University of Warwick

April 25, 2010

Abstract

This document provides the design and implementation of the Cartesian Genetic Programming (CGP) software developed by John Erik Sloper for his PhD thesis.

Contents

1 Introduction

This document provides a high-level overview of the Cartesian Genetic Programming framework. It explains all major components of the framework and their interconnections. It is assumed that the reader has a good knowledge Genetic Programming (GP) in general and CGP in particular.

2 Goals and guidelines

The main goal while designing and developing the system has been to learn and gain an insight into CGP as an evolutionary approach. The choices made are therefore not optimised neither for speed nor memory usage.

3 Architectural Strategies

The underlying strategy was to create a framework for CGP capable of solving any type of problem. Emphasis was therefore made on making the system extensible. It should be possible to define new problems and easily implement them for use in the CGP framework.

The system was built using C++ mainly due to the experience of the developer and the availability of good graphical libraries with which user interfaces can be built.

Note that no focus has been made on concurrency at this time and this is therefore left for future extensions of the framework.

4 System Architecture

There a four main components in the system as described below:

4.1 Problem

Describes a particular problem to be solved by the CGP. The problem also incorporates functionality in order to provide test cases to a population and to evaluate the output of candidate solutions.

4.2 Individual

Represents a candidate solution to a problem. Each individual consists of a number of interconnected nodes/genes each encoding a specific function and which inputs to use.

4.3 Population

This represents a population/collection of Individuals/candidate solutions to the problem at hand. It supports the basic operations one would expect to be able to perform on a population such as retrieving the best individual, retrieving fitness statistics (best, average, worst), etc. It also supports the functionality of creating a new generation including all common evolutionary operators. It also encompasses functions to evolve the next step generational step. In order to evaluate the fitness of the individuals the Population retrieves inputs and fitness function from the Problem (see 4.1).

The population stores the results of each generational step using the RunInfo (see 4.4).

4.4 RunInfo

The RunInfo stores all information about the current evolutionary process including parameters to the CGP. It keeps a record of the best individual and all relevant statistics at each generational step.

5 Detailed System Design and implementation

In this section a more detailed overview of the design and implementation of the core parts will be presented. Figure 1 shows a class diagram of the implementation. In the following sections each of the major classes are discussed:

5.1 Problem

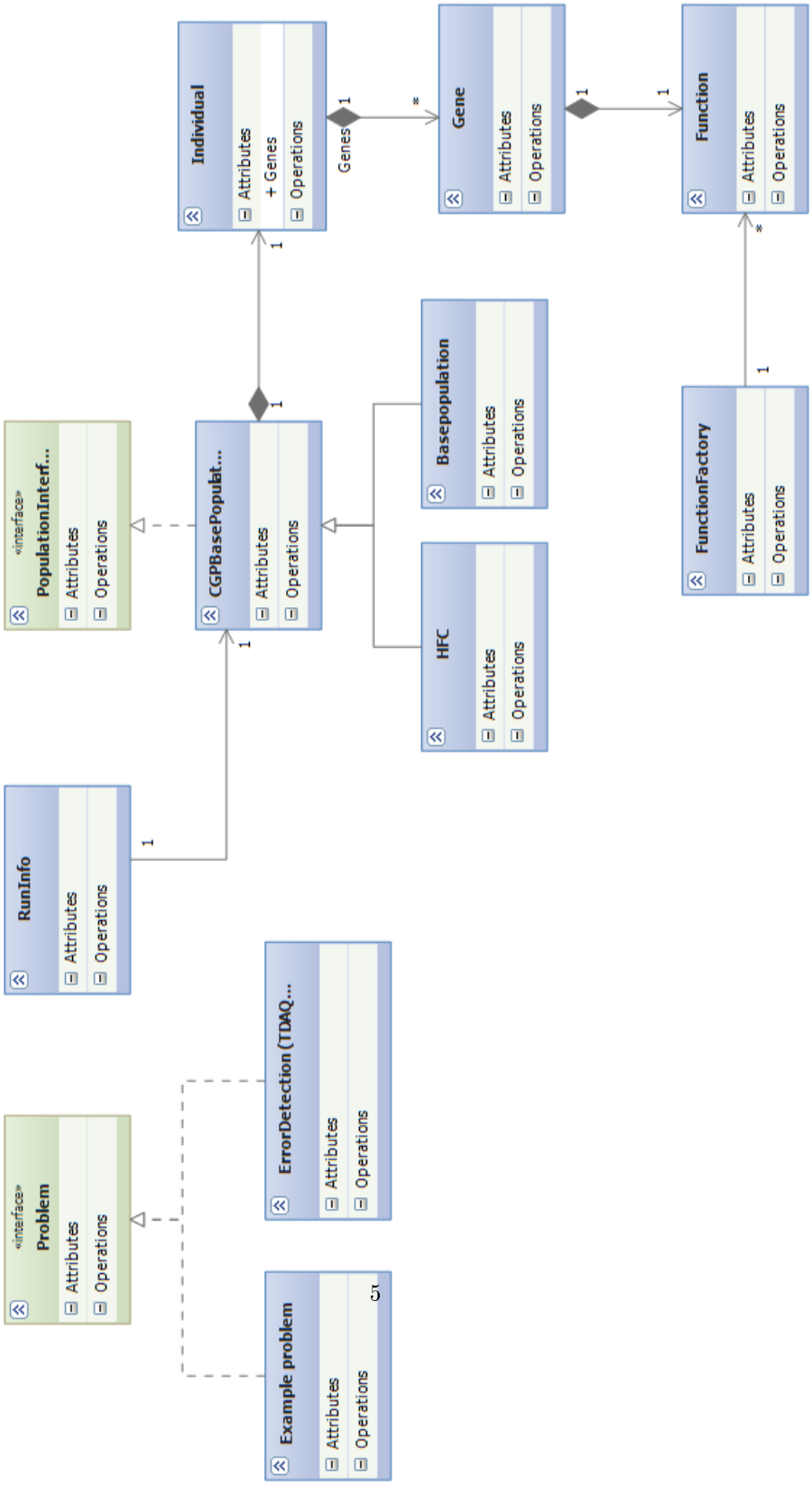
The problem contains all relevant information about a given problem. The framework provides an abstract base class called *Problem* from which all problem **implementations** should inherit. Each problem must hold information such as the number of fitness cases, the number of inputs and outputs for the particular problem etc. The problem class also defines methods to retrieve all fitness cases and to calculate fitness for a given output; i.e. it defines the *fitness function*.

```
class Problem {
public:
    Problem() { ... }

    virtual ~Problem() { ... }

    /**
     *\brief sets up the problem according to number of inputs, outputs, etc.
     */
    void setup() { ... }
    virtual double setFitness(TwoDArray<double> &output) =0; ///< \brief sets the fitness affection for the current input
    virtual double answer() = 0;            ///< \brief should return the correct answer for a given inputs
    virtual double* getCurrentInputs() = 0; ///< \brief this should return the current inputs
    TwoDArray<double>* getAllInputs() { ... }
    virtual QString getName() = 0;  ///< \brief this should return the name of the problem
    virtual void reset() {}         ///< \brief should reset any data if needed
    virtual QString description() { ... }
    virtual void inputStringValues(QVector<QString>& inp) { ... }

    /* ... */
    int NumberOfFitnessCases() { ... }
    /* ... */
    int NumberOfInputs() { ... }
    /* ... */
    int NumberOfOutputs() { ... }
```

5.1.1 Constraints

The fitness function should provide a numerical value as a fitness measure. A lower fitness value should be used to represente a better fit individual as the framework will attempt to *mimize* the overall fitness.

Figure 1: Class diagram



5.1.2 Composition

The Problem is not composed of any other classes.

5.1.3 Uses/Interactions

The Problem class is utilized by the Population in order to retrieve fitness cases and to calculate the fitness value for a given output.

5.2 Individual

An individual represents a candidate solution to a particular problem. Each Individual consists of a number of interconnected Genes in the manner of a directed graph. The last n nodes represent the output of the individual, where n is the number of outputs needed by a particular problem. The Individual class provides methods to access the Genes, retrieve the output of the individual, etc. Some of the methods availble in the Individual class can be seen in the program listing below:

```
class Individual
{
public:
    Individual(){
        this->numberOfNodes=0;
        Genes.clear();
    }
    ~Individual(void);                ///< \ brief dtor
    void copy(const Individual &rhs);   ///< copies and individual.  We cannot use shallow copy because of the vector
    void insertGene(int f, int s, QString func); //< \brief inserts a new gene at the back of the individual
    void swapGenes(Individual& other, int f, int s) throw(OutOfRange);
    void printGenes() const;           ///< \brief prints the genes using qDebug
    bool equals(const Individual& other) const; ///< \brief check if two individuals are equal
```

5.2.1 Constraints

An Individual can currently only consist of Genes. Future plans to incorporate other types of building blocks, e.g embedded components, are planned.

5.2.2 Composition

An Individual consists of a set of Genes organized as described above.

5.2.3 Uses/Interactions

The Individual is used by the Population class or its derived classes.

5.3 Gene

The gene represents a single node in a candidate solution. It consists of a number of inputs, a function and a single output. The inputs to the Gene are either the problem inputs, or outputs generated by other genes. The function will utilize the input values as its parameter in order to calculate its output which can then be utilized by other nodes (or potentially as the output of the candidate solution).

5.3.1 Constraints

Currently a Gene can only be connected to 3 other Genes and/or inputs. This will be extended so that any number of connections can be used.

5.3.2 Composition

A Gene is composed by inputs in the form of integer values indicating the connection to other Genes or to inputs. It also contains a pointer to a Function object which represents the function associate with this node.

5.3.3 Uses/Interactions

The Gene is used internally by the Individual class in order to represent a candidate solution.

5.4 Population

The population encomasses both the set of potential solutions and all methods for generating subsequent generations including all evolutionary operators.

The framework provides an abstract base class called PopulationInterface from which all implementations myst inherit. By default two different types of Populations are provided in the framework:

Regular Implements the typical evolutionary model found in literature.

Hierarchical-Fair-Competetion Implements an hierarchical fair competition (HFC) approach which includes a number of sub-populations (demes). This implementation controls all

The population utilizes the **Problem** in order to retrieve inputs and to evalute the fitness of the Individuals.

The population consitsts of a set of Individuals.

The population encompasses methods to generate a new generation, retrieve best individual etc.

An population interface defines all methods that must be determined by any population implementation.

5.4.1 Composition

The Population is composed of a list of Individuals (see5.2).

5.4.2 Uses/Interactions

The Population interacts with the Problem class in order to evaluate the fitness of each Individual. It also interacts with the RunInfo in order to access the different parameters of the CGP and in order to store the progress at each generation.

5.5 RunInfo

The RunInfo contains all parameter about the current run. This includes all CGP specific parameters such as mutation and crossover rate, population size, number of nodes, selection strategy, etc. The results of each generational is stored with the RunInfo which thus contains the complete information about a single run, both the parameters used and the results at each stage of the evolution.

```
class RunInfo {
    /**
     * Nested class to keep all GP parameters
     */
    struct GPParameters { ... };

    /**
     *  Nested class to keep all selection paramaters
     */
    struct SelecetionParameters { ... };

    /**
     *  Nested class to keep data about each generation
     */
    struct RunData { ... };

public:
    RunInfo();
    ~RunInfo(void) { ... }
    QMap<QString, Function<double>*> functions; ///< \brief available functions in this run
    QMap<QString, int> functionHeuristics; ///< \brief keeps count of mostly used functions
    QString getRandomFunction() { ... }

    /**
     * Stores all data in the DomDocument
     */
    void store(QDomDocument& doc);
    void setGPInfo(int ind, int gen, int nodes, int mut, int cross, int point, int back, bool tree);
    void setGridInfo(bool grid, int Rows, int Cols, int cons);
    void setSelectionInfo(bool elitism, int selType);
```

5.5.1 Constraints

The RunInfo currently stores all values as integer and/or real numbers.

5.5.2 Composition

The RunInfo class is composed of 3 interal classes representing CGP parameters, selection parameters and the data for each generation.

5.5.3 Uses/Interactions

The RunInfo is used both by a user interface to set the parameters of the run, and by the Population in order to retrieve this data and to store the results.

5.6 Interaction

Figure 2 shows a complete interaction diagram for a typical execution using the CGP framework.

Figure 2: Basic flow diagram