A MULTI-AGENT SYSTEM FOR AUTOMATED POST-FAULT DISTURBANCE ANALYSIS

John A. Hossack

MEng

Submitted for the Degree

Of

Doctor of Philosophy

Institute for Energy and Environment

Department of Electronic and Electrical Engineering

University of Strathclyde

Glasgow G1 1XW

UK

June 2005

Abstract

Within today's privatised electricity industry, post-fault disturbance analysis is becoming an increasingly challenging prospect for protection engineers. Not only must they be proficient at operating a diverse range of data gathering tools but they must also be able to spend the time necessary to interpret the large volumes of data generated by modern network monitoring devices. Although a degree of automated assistance is provided by existing intelligent decision support tools, it remains for the protection engineer to manually collate and interpret the output of each system in order to compile a comprehensive understanding of each disturbance.

As detailed in this thesis, the requirement for manual intervention has been eliminated through the development of the Protection Engineering Diagnostic Agents (PEDA) decision support architecture capable of automating all aspects of post-fault disturbance analysis. An essential component within this architecture is an alarm processor developed specifically to assist protection engineers with the early stages of post-fault disturbance analysis. The novel reasoning methodology employed emulates a protection engineer's approach to alarm analysis, providing automatic identification of transmission system disturbances and events.

PEDA achieves fully automated post-fault disturbance analysis through the novel use of Multi-Agent Systems (MAS) to integrate the alarm processor with other automated systems for fault record retrieval, fault record interpretation and protection validation. As will be described in the thesis, achieving systems integration using MAS provides for levels of architecture flexibility and extensibility not previously realised within existing integrated decision support architectures.

The PEDA architecture was developed following a comprehensive eleven stage methodology created as part of the reported research to assist with the specification of MAS for decision support within the power industry. Each stage of the PEDA specification process is detailed together with its implementation. Finally, the implemented architecture has been shown to offer automated retrieval, interpretation, collation and archiving of disturbance information within five minutes of a disturbance occurring. The beneficiaries of this near real-time provision of disturbance information need not be limited to protection engineers.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to take this opportunity to express my heart felt thanks to the people who have helped and supported me during my research and, believe me, there have been too many to mention!

I would like to start by thanking Professor Jim McDonald, for offering me the opportunity to work in a dynamic, challenging and industrially relevant research environment. It was only through his support and business acumen that I could combine my career ambitions with my goal to see the world!

Special thanks must go to Dr Stephen McArthur for his never ending support, encouragement and, often unnerving, but understandable, excitement for the research field of intelligent agents. Thanks also to Graeme Burt, Ian Elders, Euan Davidson, Eleni Mangina and Gordon Jahn for not only providing expertise in intelligent systems and intelligent agents but also assistance with the software development essential to this research.

Formal thanks must go to SP PowerSystems for providing financial support during my research years. Special thanks to Tom Cumming, Jim Farrel and John Stokoe for providing data, protection expertise and a challenging environment in which to apply the research. Their enthusiasm and interest in the research was appreciated.

Special thanks to my friends who helped and supported me, especially when I thought things were not going as well as they should. Thanks for putting up with my absence from many social gatherings during my 'write up' period. I will now have no excuse to avoid the fun, but often wild, social occasions!

Last, but by no means least, thanks to my parents, brothers and all of my family for their support and encouragement over the past years. Together with my friends, they have ensured that I can at least finish this chapter of my life with a degree of sanity left.

Glossary of Terms

ACL	Agent Communication Language
AI	Artificial Intelligence
ANN	Artificial Neural Network
APEX	Alarm Processing EXpert System
ARCHON	ARchitecture for Cooperative Heterogeneous ON-line systems
CBR	Case Based Reasoning
COMMAS	COndition Monitoring Multi Agent System
COMTRADE	COMmon format for TRAnsient Data Exchange
DAR	Delayed Auto-Reclose
DAI	Distributed Artificial Intelligence
DESIRE	DEsign and Specification of Interacting REasoning components
DFR	Digital Fault Recorder
DPM	Dynamic Protection Models
DSS	Decision Support System
EMS	Energy Management System
FIPA	Foundation for Intelligent Physical Agents
IS	Intelligent System
JESS	Java Expert System Shell
KBS	Knowledge Based System
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
MAS	Multi-Agent System
MaSE	Multiagent Systems Engineering methodology

MBD	Model Based Diagnosis
PEDA	Protection Engineering Diagnostic Agents
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
SL	Syntactic Language
SPID	Strategic Power Infrastructure Defence
TWFL	Travelling Wave Fault Locator
UML	Unified Modelling Language
VHF	Very High Frequency

Chapter 1: Introduction

1.1 Justification for and Introduction to Research

Since the privatisation of the UK electricity industry, the utility companies responsible for operating the electricity infrastructure have focussed on maximising the return on assets inherited from their publicly owned predecessors. This has led to components of the distribution and transmission networks, such as transformers and transmission lines, being operated at or close to their stability and/or thermal limits.

Given this operating regime, the correct detection and removal of faulty power system plant by protection is paramount if the extent to which a fault affects the network is to be minimised. The impact of incorrect protection operation, whether attributed to incorrect specification, commissioning or maintenance of protection schemes, can be severe. The south London blackout in 2003 is just one of many where human error resulted in the incorrect commissioning of a protection relay and the unnecessary disconnection of a large network area [1].

Unfortunately, as reported in [2], there is an increased likelihood of such severe network disturbances due to the industry wide shortage of qualified and experienced protection engineers available to specify and commission protection schemes.

To ensure the correct operation of protection schemes, a utility's protection engineers must conduct a post-fault disturbance analysis to validate that the protection operated correctly and identify any anomalous behaviour. To facilitate this protection engineers have at their disposal a range of network monitoring devices. Using the data provided by these devices, the protection engineer can determine the state of the network prior, during, and post fault and the timing of protection operations.

The primary data types available to protection engineers are alarms generated by the Supervisory Control And Data Acquisition (SCADA) system and fault records. The SCADA alarms are time-stamped messages providing data relating to changes in plant status, protection operations and other pertinent network events. Fault records are generated by Digital Fault Recorders (DFRs), which are placed within substations at each circuit end and monitor and record circuit voltages, currents and protection scheme timing sequences during a disturbance.

Post-fault disturbance analysis is a manual process, which not only relies on the availability of disturbance data but also on the knowledge and experience of the protection engineer conducting the analysis. Protection engineers at ScottishPower PowerSystems traditionally follow a multi-stage approach to manual analysis:

Stage 1. The engineer retrieves from a SCADA database the alarms generated around the time of the disturbance.

Stage 2. The retrieved alarms are interpreted to identify when protection detected a fault and to group disturbance related alarms.

Stage 3. The grouped disturbance alarms are further interpreted to identify the key events in the protection operating sequence.

Stage 4. Other data sources such as DFRs, which may have recorded additional disturbance data, are identified and data retrieval initiated.

Stage 5. The additional data is interpreted to gain more information.

Having retrieved, interpreted and collated all disturbance data the protection engineer has gathered sufficient information to decide whether the protection operated correctly. Information on the faulted plant, protection that operated, the protection operating times and the fault types are the most common pieces of information.

Unfortunately, this manual approach to disturbance analysis suffers from a number of problems putting additional strain on the small pool of protection engineers:

- Data overload is a major difficulty, particularly following significant network disturbances such as those caused by storms [3]. This was illustrated in 1998, when an experienced ScottishPower protection engineer took over 3 weeks to manually interpret 15000 alarms and 150 fault records following a storm.
- To retrieve data, the protection engineer must be able to operate a diverse range of proprietary data gathering software tools. This problem is being exacerbated by the introduction of new technologies and software tools.
- The retrieved data is presented in different formats, requiring application of different interpretation techniques and reasoning knowledge.

- Only through time-consuming data interpretation can the engineer decide what data is of interest and should be collated.

It was clear that the protection engineers would benefit from the introduction of decision support and automation into the post-fault disturbance analysis process. This was addressed through an integrated decision support architecture developed by researchers at the Institute for Energy and Environment [4].

Up until the development of the integrated system, decision support had traditionally been provided by a suite of standalone intelligent systems adapted from existing control room applications, also known as Decision Support Systems (DSS). Two rule-based DSS were available to the protection engineers, both performing online SCADA alarm interpretation: an alarm processor [5] providing concise summaries of protection events and a fault diagnosis system providing diagnoses on the possible root causes of fault. An additional model-based DSS was also available, which had been developed with protection engineers in mind and was capable of validating the operation of protection schemes and diagnosing protection failures [6].

The new integrated decision support architecture enhanced the provision of decision support through the integration of these existing DSS with improved data retrieval functions. Integration ensured the individual decision support capabilities of each system could be maintained and that the disturbance information they generated could be used to prioritise fault record retrieval and automate protection validation.

However, experience with the architecture during its operational period had raised questions as to the viability of not only the DSS used within the architecture but also the approach taken to systems integration. A number of issues had been identified which together led to the eventual removal of the architecture from service:

- Both the alarm processing and fault diagnosis expert systems had originally been developed for control room applications with their knowledge bases having been adapted to generate events and diagnoses of interest to the protection engineer. However, neither system was capable of generating information in a format which was immediately amenable to the protection engineer.

- The hardware platforms for both the alarm processor and fault diagnosis expert system were obsolete and proved difficult to maintain. Furthermore, each system used an operating system with which the protection engineers were not adept, hindering their ability to query the systems for decision support information.
- Due to the non-standard protocols and communications languages employed by the alarm processor and fault diagnosis expert systems, interfacing with the corporate IT network to retrieve and exchange data proved difficult, on many occasions resulting in the loss of data and system crashes.
- The next generation of DFRs had been installed on the network and new software systems introduced to manage the retrieval of fault records from the new devices. This new fault record retrieval software introduced communications protocols and data formats which the existing architecture had not been designed to accommodate. In order to prioritise retrieval of fault records from these devices, the existing architecture would require some invasive software modifications.

Upon the removal of the decision support architecture, protection engineers had to yet again resort to manual post-fault disturbance analysis. Nevertheless, operational experience with the architecture had demonstrated the potential of integrated systems for improving the provision of decision support and the protection engineers were keen to see the introduction of an improved decision support architecture.

The operational experience with the previous integrated architecture highlighted three challenges which had to be met if the decision support achievements realised by the previous system were to be exceeded and the ultimate goal of fully automated post-fault disturbance analysis realised:

- Tailoring of alarm processing to better meet the needs of protection engineers

  Protection engineers will only truly embrace new decision support systems if they provide disturbance information which is both pertinent and in a format requiring minimal additional analysis effort to be of use during disturbance analysis. Existing approaches to alarm processing within control room environments must therefore be tailored, and a new alarm processor developed, to emulate the post-fault SCADA interpretation reasoning followed by protection engineers.

- Increasing the degree of autonomy exhibited by decision support tools

  Significant time-savings would be obtained if decision support tools could be completely autonomous, being capable not only of managing their own tasks but also actively participating in the prioritised retrieval and collation of disturbance related data. The requirement for manual intervention in the data retrieval process would be minimised and all pertinent disturbance data and information would be available prior to the protection engineer commencing analysis.

  Encapsulation of data collation knowledge and introduction of a reasoning capability would be essential if systems are to be autonomous and participate in the proactive dissemination of disturbance data and information to other systems.

- Achieving flexibility and scalability in an integrated architecture

  Integrated decision support architectures must be scalable and flexible to adapt to the introduction of new technologies and decision support tools. Architectures which prohibit the easy integration of new systems will be unable to maintain comprehensive levels of decision support leading to their eventual obsolescence. Furthermore, the architecture must be dynamic and sufficiently flexible to cope with temporary loss of communications between system components and adjustments to the configuration and network locations of each system.

The research reported in this thesis, by way of introduction, embraces these challenges through the development of a novel integrated architecture automating post-fault disturbance analysis through the integration and automation of individual software components. An essential component within this architecture is an alarm processor developed specifically to assist protection engineers with the early stages of post-fault disturbance analysis.

At the outset of the reported research, protection engineers had to conduct manual post-fault SCADA interpretation due to the existing alarm processor having been removed from service. The provision of an alarm processor capable of grouping disturbance alarms and events and emulating their approach to disturbance analysis would clearly be beneficial.

One possible approach was to port the existing alarm processor onto another platform and adapt it to provide the required functionality. However, assessment of the alarm processor's internal reasoning architecture indicated that it was not suitable for disturbance alarm and event grouping with a software rewrite being required to adapt the architecture. It was concluded that this provided a sufficient incentive to develop a new alarm processor specifically for post-fault SCADA interpretation. The development of this alarm processor will be reported upon in chapter four.

The next stage in the reported research focussed on identifying a suitable technology for automating decision support tools and implementing a flexible and scalable decision support architecture. Multi-Agent Systems (MAS) [7] were identified as the ideal technology as it offered features which facilitated the easy integration of existing systems, such as the new alarm processor, with other decision support tools.

It will be demonstrated throughout this thesis that systems integration can be achieved within MAS through the wrapping of each system as an 'intelligent agent'. The agent wrapper provides a reasoning capability enabling the system to react to its environment and automate its internal functions and reasoning. Furthermore, the MAS provides a standardised communications mechanism and common communications vocabulary (an 'ontology' in agent terms) facilitating the social interaction of the integrated systems using the agent wrappers. Finally, the provision of utility agents [8] such as nameservers and facilitators provide the information discovery functions necessary for flexibility and scalability. Chapter five provides a comprehensive review of MAS and their application within the power industry.

Prior to the research reported in this thesis, MAS had not been identified as a possible mechanism for realising integrated decision support systems within the power industry and, consequently, no methodology or formal process for achieving systems integration existed. A number of methodologies were available to assist with the specification, design and development of MAS in general. However, evaluation of these methodologies indicated that none were suitable for encapsulating the characteristics of decision support. A key outcome of the reported research work was the creation of a new methodology developed specifically for the specification of MAS for decision support within the power industry.

With the creation of the methodology, the specification and implementation of a MAS for automating post-fault disturbance analysis could commence. As will be detailed in this thesis, an eleven stage specification process resulted in the specification and eventual implementation of the Protection Engineering Diagnostic Agents (PEDA) MAS.

PEDA achieves automated disturbance analysis through the use of four core functional agents: an Incident and Event Identification (IEI) agent, a Fault Record Retrieval (FRR) agent, a Fault Record Interpretation (FRI) agent and a Protection Validation and Diagnosis (PVD) agent. The disturbance analysis functionality within each agent is realised through reuse of existing systems, such as the new alarm processor, and development of new software. Flexibility and scalability in the architecture is achieved through the design of a disturbance analysis ontology and use of an industry standard communications protocol. Chapter eight of this thesis will assess the disturbance analysis capabilities of this architecture using case studies derived from actual power system disturbances.

In summary, the research reported in this thesis proposes a novel application of agent-based systems and discusses the issues associated with their development within the post-fault disturbance analysis arena. In particular, it is argued that multi-agent technologies provide a means of not only optimising the provision of decision support to protection engineers in the short-term but also in the longer term due to the open architecture offered by modern MAS.

In terms of novelty of the research undertaken, five contributions can be identified:

- The design, development and implementation of an intelligent system focussed on assisting protection engineers with post-fault SCADA interpretation.
- The creation of a methodology for the specification of MAS for decision support within the power industry using multi-agent technology.
- The design of a multi-agent architecture for providing post-fault disturbance analysis decision support assistance to protection engineers through integration and automation of existing software systems.

- The implementation of a multi-agent architecture for automating post-fault disturbance analysis using hybrid-data interpretation techniques across a number of intelligent agents.
- Demonstration of the benefits adopting a multi-agent approach can bring to the integration of decision support systems through the evaluation of a multi-agent architecture using power system data generated from actual disturbances.

1.2 Thesis Overview

The remainder of this thesis has been divided into eight principal chapters:

Chapter two introduces the application domain in which the research described in this thesis has been applied. The basic concepts of the electrical power transmission system, its protection and monitoring are described. In the latter sections of the chapter, the post-fault disturbance analysis process commonly followed by protection engineers will be described.

Chapter three introduces intelligent systems discussing the fundamental technologies relevant to the research described later in this thesis. The knowledge engineering process required to capture the knowledge implemented within an intelligent system is also presented. The chapter concludes with an overview of the intelligent systems and decision support tools available to protection engineers during post-fault disturbance analysis.

Chapter four presents an intelligent system developed to automate the SCADA alarm interpretation task conducted during the early stages of disturbance diagnosis. A brief overview of alarm processing research to date is presented followed by the protection engineers' alarm processing requirements. How these requirements are met, and the alarm interpretation process emulated by a novel reasoning architecture is then described. The application of this architecture within the intelligent system and its online performance in an industrial setting is then assessed using a case study based on actual power system data.

Chapter five describes the research area of MAS, introducing intelligent agents and outlining the issues related to agents' communications, ontologies and knowledge representation. The application of MAS within the power industry is also described.

Chapter six introduces a methodology developed to assist with the specification of MAS for decision support within the power industry. The characteristics of power engineering systems, which distinguish them from the more common applications of MAS, are described and used to critically assess the suitability of existing MAS design methodologies for implementing hybrid systems as MAS within the power industry. Each stage of the methodology is described in detail.

Chapter seven details the application of the methodology for the specification of the Protection Engineering Diagnostic Agents (PEDA) MAS for automating post-fault disturbance analysis. Each stage of the specification process and the design choices made are described in detail. The inter-agent communications essential to achieving automated analysis are modelled in addition to agent behaviour.

Chapter eight describes the implementation of the PEDA specification and the deployment of the PEDA agents. The performance of PEDA is then assessed using a case study based on actual power system data identifying the benefits offered by the multi-agent approach to post-fault disturbance analysis.

Finally, chapter nine summarises the principal conclusions of the work carried out, highlighting the main achievements, and proposes further research and development work to build on the results to date.

1.3 Associated Publications

The following publications have arisen from the research detailed in this thesis:

- S.D.J. McArthur, E. Davidson, J.A. Hossack, J.R. McDonald, "Automating Power System Fault Diagnosis Through Multi-Agent System Technology", (Invited Paper) Hawaii International Conference on System Sciences (HICSS), Hawaii, USA, January 5-8, 2004.

- S.D.J. McArthur, J.R. McDonald, J.A. Hossack, "A Multi-Agent Approach to Power System Disturbance Diagnosis", (Invited Book Chapter) Autonomous Systems and Intelligent Agents in Power System Control and Operation, Christian Rehantz (Editor), Springer-Verlag, 2003.
- J.A. Hossack, S.D.J. McArthur, J.R. McDonald, "Integrating Intelligent Protection Analysis Tools Using Multi-Agent Technologies", ISAP 2003, Lemnos, Greece, September 2003.
- J.A. Hossack, J. Menal, S.D.J. McArthur, J.R. McDonald, "A Multi-Agent Architecture for Protection Engineering Diagnostic Assistance", IEEE Transactions on Power Systems, v18, n2, May 2003.
- S.D.J. McArthur, J.A. Hossack, G. Jahn, "Multi-Agent Systems for Diagnostic and Condition Monitoring Applications", (Invited Panel Session Paper) IEEE Power Engineering Society General Meeting, 2003, Toronto, Canada, July 2003.
- J.A. Hossack, J. Menal, S.D.J. McArthur, J.R. McDonald, "A Multi-Agent Architecture for Protection Engineering Diagnostic Assistance", (Poster Session) IEEE Power Engineering Society General Meeting, 2003, Toronto, Canada, July 2003.
- J.A. Hossack, E. Davidson, S.D.J. McArthur, J.R. McDonald, "A Multi-Agent Intelligent Interpretation System for Power System Disturbance Diagnosis", Expert Systems 2002 Conference (ES2002), Cambridge, UK, December 2002.
- J.A. Hossack, S.D.J. McArthur, J.R. McDonald, J. Stokoe, T. Cumming, "A Multi-Agent Approach to Power System Disturbance Diagnosis", IEE Power System Management and Control (PSMC) Conference, London, UK, April 2002.
- J.A. Hossack, G.M. Burt, J.R. McDonald, T. Cumming, J. Stokoe, "Progressive power system data interpretation and information dissemination", Proceedings of the IEEE Power Engineering Society Transmission and Distribution Conference, Atlanta, GA, US, v 2, pp 907-912, October 2001.
- J.A. Hossack, S.D.J. McArthur, G.M. Burt, J.R. McDonald, T. Cumming, J. Stokoe, "An Integrated Approach to Telemetry Processing for Alarm

Management and Diagnosis", 35th Universities Power Engineering Conference (UPEC), Belfast, UK, September 2000.

Other publications on related topics are:

- I. Elders, J.A. Hossack, A. Moyes, G.M. Burt, J.R. McDonald, "The Use of Internet Technologies to Enable Flexible Alarm Processing", IEE Power System Management and Control (PSMC) Conference, London, UK, April 2002.

1.4 References

[1]. "Tracing the London Blackout", The IEE Power Engineer, October/November 2003, pp 8-9.

[2]. W. Laycock, "Protection Engineer Shortage", IEE Power Engineer, February/March, 2004, p47.

[3]. L.E. Smith (Chairman), "Analysis of Substation Data", Report from IEEE/PSRC Working Group 19, 2002.

[4]. S.D.J. McArthur, S.C. Bell, J.R. McDonald, et al, "The Development of an Advanced Suite of Data Interpretation Facilities for the Analysis of Power System Disturbances", CIGRE 1998, Paris, France, August 1998.

[5]. S.D.J. McArthur, J.R. McDonald, S.C. Bell, G.M. Burt, "An Expert System for On-line Analysis of Power System Protection Performance", in Proc. 1994 Expert Systems conference: Applications and Innovations in Expert Systems, Dec 1994, pp 125-142.

[6]. S.C. Bell, S.D.J. McArthur, J.R. McDonald, G.M. Burt, et. al., "Model Based Analysis of Protection System Performance", *IEE Proc. Gen. Trans. & Dist.*, vol. 145, n 3, pp 547-552, 1998.

[7]. M. Wooldridge, et al, "Intelligent Agents: Theory and Practice", The Knowledge Engineering Review, vol. 10, n 2, pp. 115-152, 1995.

[8]. H.S. Nwana, D.T. Ndumu, L.C. Lee, J.C. Collis, "ZEUS: A Toolkit for Building Distributed Multi-Agent Systems", Applied AI Journal, v13 n1, 1999.

Chapter 2: Fundamentals of Protection and Post-Fault Disturbance Analysis

2.1 Chapter Overview

The following chapter begins with an overview of the power system, relevant to the research described later in this thesis. A detailed, but not exhaustive, review of transmission protection schemes is then presented describing the components, some of the most popular configurations and common problems experienced with the schemes. A description of the monitoring technologies commonly found on a transmission system follows.

In the latter sections of the chapter, the post-fault disturbance analysis process commonly followed by protection engineers will be described. The terminology used during disturbance analysis, and throughout the remainder of this thesis, will be defined. This will be followed by a description of the manual data retrieval and interpretation tasks required for post-fault disturbance analysis.

2.2 The Power System

The primary purpose of a power network is to transmit electrical energy from where it is generated to where it is consumed. This is achieved in three stages:

- **Generation:** Generation is the process of creating electrical energy from its raw form, which in the UK is typically coal, gas, hydro, nuclear and wind. For reasons of practicality and cost, generation facilities are usually located close to their source of energy, which is often remote from load centres. The voltage level used to generate electricity varies, but is typically between 11 and 23.5kV.
- **Transmission:** The transmission network provides for the bulk transport of electricity from the generation sites to the distribution network. To minimise electrical losses, as high a voltage as possible is used for transmission, the voltage being stepped up at generating stations and reduced again as the electricity passes into the distribution network. In the UK, the principal transmission voltages are 400kV and 275kV with the 132kV network only being regarded as part of the transmission system in Scotland. For economic reasons the majority of the

transmission network consists of overhead lines, making the network particularly susceptible to natural elements such as wind and lightning.

- **Distribution:** From the transmission system connection points, electrical energy is conveyed to the customers via the cables and overhead lines which form the distribution network. The voltage levels are dependent on the size of the load. Large industrial customers are supplied at either 33kV or 11kV, office buildings at 415V and domestic customers at 230V.

2.3 Power System Faults

A fault on the power system can be defined as "any abnormal condition which causes a *significant reduction* in the basic insulation strength between conductors, or between phase conductors and earth or any earthed screens surrounding the conductors" [1]. In practice, a reduction is not regarded as a fault until it is detectable – that is, until it results either in an excess current or in a reduction of the impedance between conductors, or between conductors and earth, to a value below that of the lowest load impedance normal to the circuit.

The three phase overhead line and underground cable circuits, which form the majority of a transmission and distribution network, can be subject to many types of faults. The principal types of fault are:

- three-phase (with and without earth connections), e.g. Red-Yellow-Blue.
- phase-to-phase (two-phase), e.g. Red-Yellow.
- phase-to-earth (single-phase), e.g. Blue-Earth
- double phase-to-earth (phase-phase-earth), e.g. Red-Blue-Earth

Protection engineers further classify faults based on their permanence as follows:

- **Permanent:** A fault where permanent damage has been done to the insulation or conductors and quick restoration of the circuit is not possible.

- **Transient:** A fault where insulation has broken down temporarily without any permanent damage to the insulating medium and the circuit can be quickly restored by automatic switching and reclosing facilities.
- **Persistent:** A recurring transient fault, with sufficient delay between recurrences for automatic switching and reclosing schemes to reenergise the circuit before recurrence of the fault.

Due to the differences in construction, materials and exposure to environmental factors, overhead line and underground cables are prone to different kinds of faults:

- **Overhead Lines:**

  Approximately 80% of all system faults occur on overhead line circuits and virtually all are due to environmental causes such as lightning, snow, ice, fog, pollution and high winds. Contact by trees, cranes, aircraft and various other objects is another major cause of faults.

  An important feature of overhead line faults is that since air is the main insulating medium a significant majority of flashovers are transient and cause no permanent damage to the circuits. In such cases, 80% of fault clearances can be quickly followed by the circuit's return to service by operation of automatic switching and reclosing facilities. Only about 1% of overhead line faults are due to equipment failure.

  Lightning is a major cause of overhead line faults. A severe direct lightning strike to a transmission tower may raise the tower potential sufficiently to cause insulator flashover of phases of both circuits of a double circuit line resulting in a simultaneous double circuit fault and the tripping of both circuits.

- **Underground Cables:**

  Faults on underground cables are caused by: third party damage, deterioration of the solid cable insulation, joint failures and sealing end flashover and failures. The faults may be caused or precipitated by external factors, for example, damage caused by mechanical excavators, moisture intrusion or the effects of transient overvoltages caused by lightning or system conditions.

2.4 Transmission System Protection

The transmission network is the backbone of the power system and faults occurring on the feeders or plant, which constitutes the network, can have a significant impact on the continued operation of the entire power system. Not only can expensive power system plant be damaged, but network stability and power system security can also be adversely affected. To minimise these risks, protection schemes are employed on each feeder to rapidly disconnect the faulty component(s) from the power system.

The following sections describe the main features of these protection schemes, their operation and problems they commonly suffer from.

2.4.1 Protection Scheme Components

The principal components of a transmission feeder protection scheme are presented in Figure 2-1.

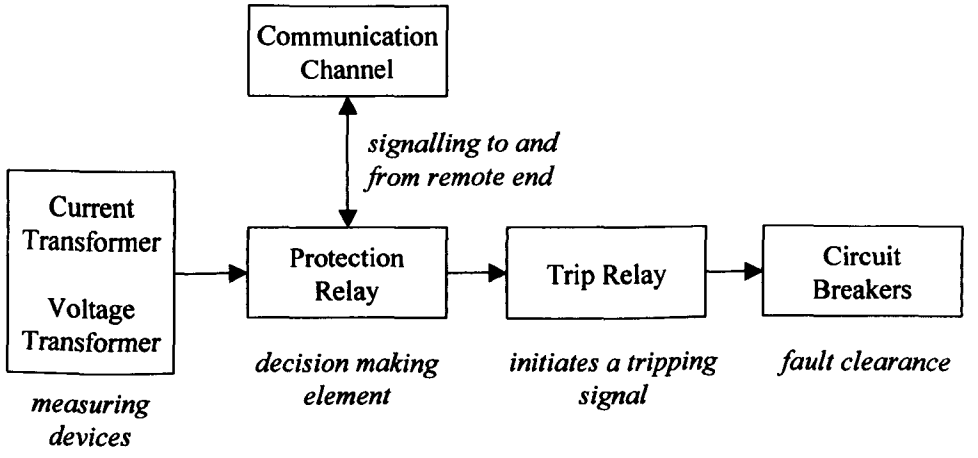


Figure 2-1 Components of a protection scheme

- **Measuring Devices:**

  Protection relays are connected to the transmission network via current and voltage transformers. The magnitudes of the transmission network's current and voltage are too large to connect the protection relays directly to the feeder and require to be scaled down. Voltage transformers scale down the system voltage to

the nominal rating of 110V for the protection relays. Current transformers scale down the system current to the nominal rating of either 1A or 5A for the protection relays.

- **Protection Relays:**

  The voltage and current indications provided by measuring devices provide the data necessary for protection relays to detect anomalies and determine whether a protective response is required. If the protection relay detects an anomaly within the area the protection relay has been set to protect, and considers it to require a protective response, a trip signal is initiated.

- **Trip Relays:**

  Trip relays are simple auxiliary relays which are used to amplify the tripping signal sent from the protection relay to operate the circuit breakers. Some modern protection relays are now designed with trip signal ratings capable of operating circuit breakers directly.

- **Circuit Breakers:**

  Circuit breakers are used to disconnect the faulted plant from the rest of the power system. Circuit breakers must be capable of interrupting the maximum fault level rating of the plant on which they are fitted.

- **Communication Channels:**

  Communications channels are required for signalling between feeder ends. They must be very reliable as a signalling failure can reduce the effectiveness of the protection scheme or even cause it to mal-operate. Three means of communication commonly used are:

  - Pilot Wires: These are low voltage cables which are either privately owned or rented from a communications company.
  - Power Line Carrier: This makes use of the power system conductors to transmit modulated high frequency signals. Line traps, which are high frequency filters, must be fitted at each end of the feeder to ensure the signal is not sent in the wrong direction.

- Radio: Transmitters and receivers are used to send and receive very high frequency (VHF) signals between sites.

2.4.2 Categories of Protection Schemes

The protection used within a feeder protection scheme may be categorised into either unit or non-unit configurations. Unit protection schemes only protect one part or component of the power system. Non-unit protection schemes provide fault detection over a large part of the power system, with operating times related to how distant the power system fault is. Both are briefly described.

2.4.2.1 Unit protection

Unit protection schemes compare either the current or voltages at two or more measuring points in order to determine whether a fault exists in the protected area.

To illustrate the unit protection principle, a single phase representation of circulating current protection schemes on two connected circuits is shown in Figure 2-2. A fault has occurred on feeder 2 and fault current is flowing through feeder 1 to the fault.

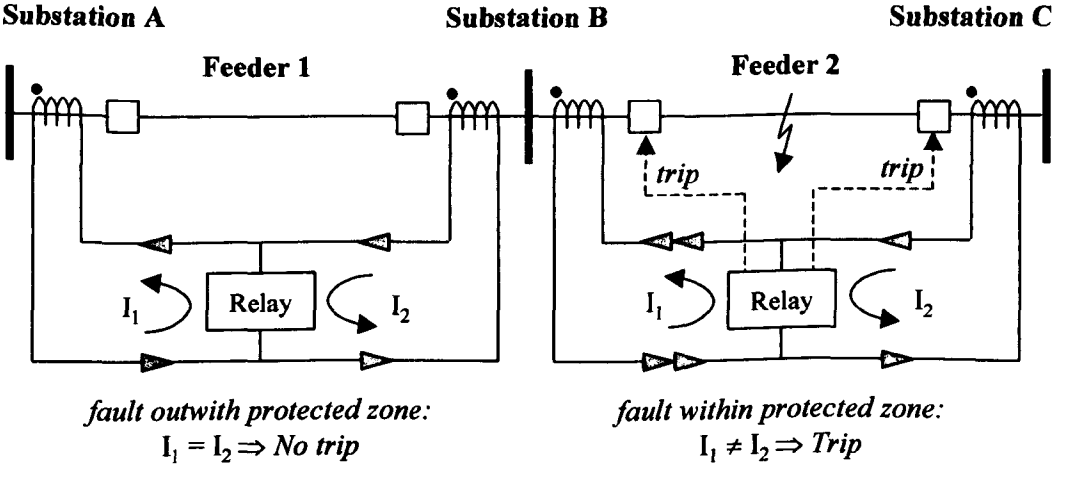


Figure 2-2 Circulating current unit protection

As can be seen in Figure 2-2, no fault exists on feeder 1 so the measured currents are equal and the protection does not trip. Conversely, the fault on feeder 2 is between

the measuring points and causes unequal currents to flow in the pilots to the protection relay initiating tripping of the circuit breakers.

2.4.2.2 Non-unit protection

Non-unit protection relies on values measured at a single location and protects an area known as its operating reach or characteristic. The most common type of non-unit protection used in the transmission network is distance protection.

Distance protection uses an impedance measurement derived from voltage and current measurements to determine whether a fault condition exists. This in conjunction with the feeder's impedance is used to calculate the distance to fault. Distance protection relays normally have a stepped operating characteristic as shown in Figure 2-3.

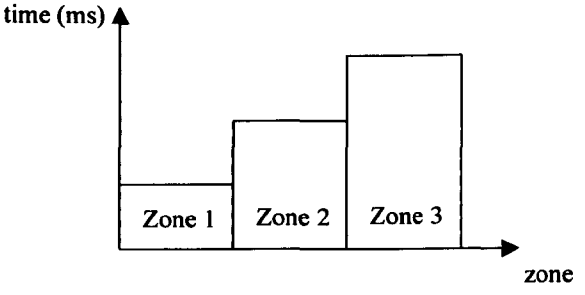


Figure 2-3 Operating characteristic of a distance protection relay

Zone one operating time is instantaneous once the protection relay has detected a power system fault. Zone two operating time is at least the time to clear a zone one power system fault, thus ensuring the protection relay does not operate unnecessarily. Similarly, zone three operating time is at least the time to clear a zone two power system fault. The protection relay zones are usually set as shown in Figure 2-4.

The protection relay will react in zone one time if a fault is detected within the first 80% of feeder one. Zone one is restricted to the first 80% of feeder one, to ensure the protection relay does not race the feeder two protection to remove a fault on feeder two. The protection relay operates in zone two time if a fault is detected in the last 20% of feeder one or the first 30% of feeder two. Similarly, the protection relay

operates in zone three time if a fault is detected in the last 70% of feeder two, the first 25% of feeder three or 10% in the reverse direction.

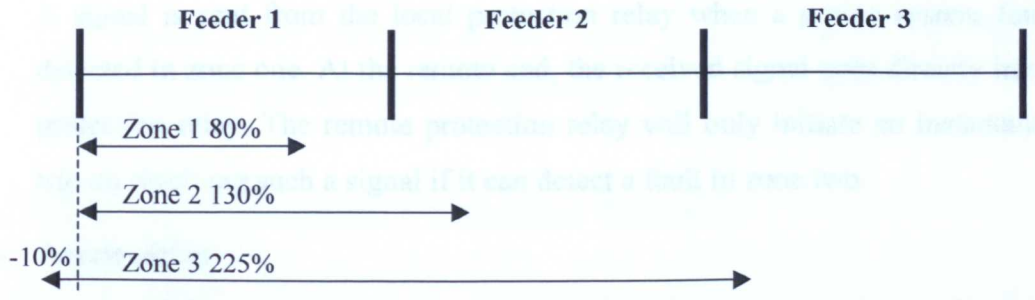


Figure 2-4 A distance protection relay's zone settings

With a single distance protection relay, with reach characteristic shown as in Figure 2-4, faults in the last 20% of feeder one would not be cleared in zone one time. To overcome this, a second distance protection relay is placed at the other end of feeder one 'looking' in the opposite direction as shown in Figure 2-5.

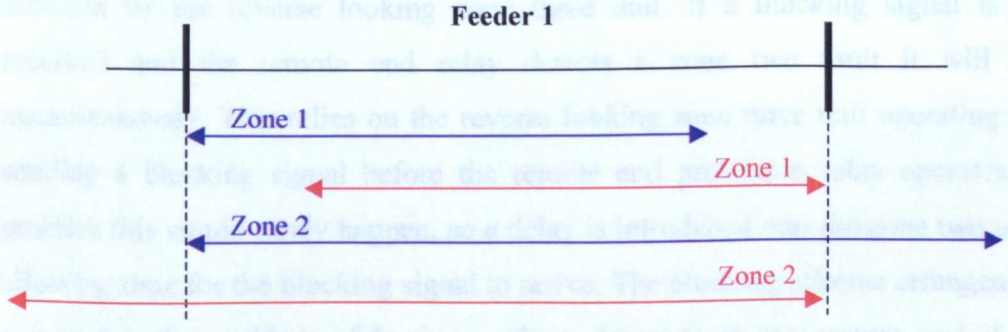


Figure 2-5 Overlap of protection relay zones

Signalling between the two protection relays allows power system faults anywhere on feeder one to be removed in zone one time. Different signalling methodologies can be adopted such as:

- **Intertripping:**

  A signal is sent from the local protection relay when a fault is detected in zone one. At the remote end, the inter-trip signal trips directly into a trip relay which

operates the circuit breaker. No checking is done at the remote end to ensure the fault has been detected.

- **Permissive under-reach:**

  A signal is sent from the local protection relay when a power system fault is detected in zone one. At the remote end, the received signal goes directly into the protection relay. The remote protection relay will only initiate an instantaneous trip on receiving such a signal if it can detect a fault in zone two.

- **Acceleration:**

  In an acceleration scheme, the protection relay's zone one and zone two fault detection is performed by the same unit. When an acceleration signal is received by the remote protection relay, it changes the zone one reach to that of zone two enabling it to operate simultaneously if a zone two fault is detected.

- **Blocking:**

  A blocking scheme utilises inverse logic to that of the previous three. A blocking signal is sent to the remote end when a fault external to the protected zone is detected by the reverse looking zone three unit. If a blocking signal is not received and the remote end relay detects a zone two fault it will trip instantaneously. This relies on the reverse looking zone three unit operating and sending a blocking signal before the remote end protection relay operates. In practice this would rarely happen, so a delay is introduced into the zone two units allowing time for the blocking signal to arrive. The blocking scheme arrangement overcomes the problem of having a slow clearance at the remote end if the communication channel fails.

2.4.3 Delayed Auto Reclose (DAR)

To address the problem of transient faults caused by wind and other natural phenomena, transmission feeder protection schemes are often augmented with Delayed Auto-Reclose (DAR) equipment, which attempts a reconnection of the tripped line after a specified delay. The protection relay and DAR sequencer / timer elements may appear as separate units or be combined together.

The DAR attempts a reconnection of the tripped line after a specified dead time (the period of time taken to initiate a reclosure of the circuit breaker) - circa 15 seconds at transmission voltages. If the fault is transient, the feeder will remain in service once re-energised. If the fault is seen to be permanent, or is persistent, the protection will operate instantaneously disconnecting the feeder again. In the UK delayed auto-reclosing is usually only performed once on each transmission feeder.

2.4.4 A Typical Feeder Protection Scheme

On the transmission system, power system faults should be cleared in approximately 100ms. To illustrate the sequence of protection operations required to clear a fault within these timescales, a protection scheme for a typical 400kV transmission feeder circuit will be described. Note that for reasons of clarity, backup protection and DAR will not be described.

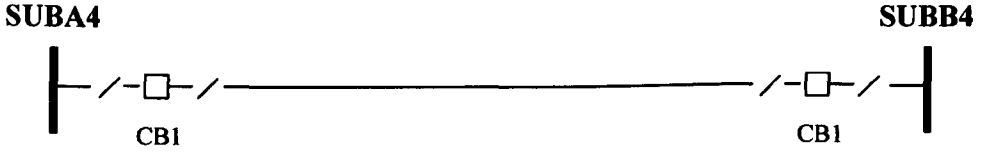


Figure 2-6 Single line diagram representation of a 400kV feeder circuit

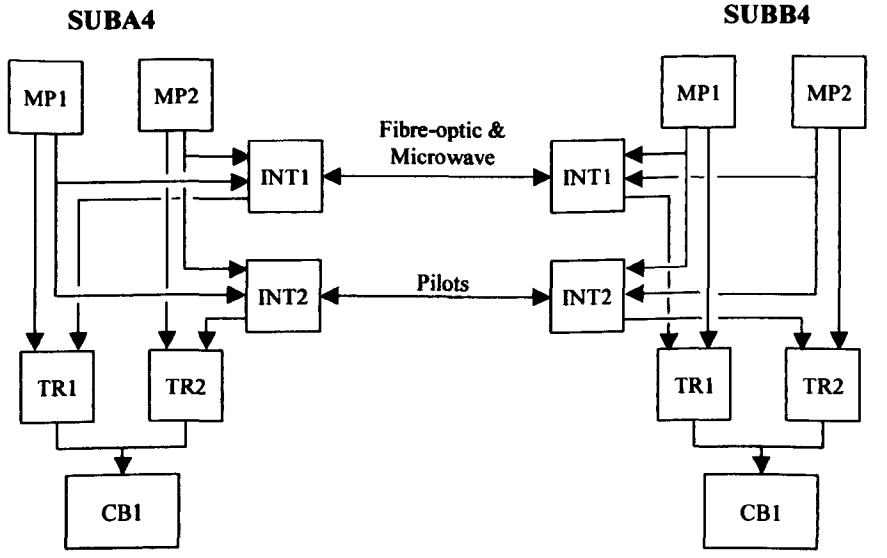


Figure 2-7 Example of a 400kV feeder protection scheme – no DAR.

The protection scheme used to protect the feeder in Figure 2-6 is illustrated in Figure 2-7. A description of the components represented within Figure 2-7 follows:

- MP1 (Main Protection 1): Typically unit protection, MP1 will monitor the feeder for a fault in the protected area.
- MP2 (Main Protection 2): Typically distance protection, MP2 will monitor the feeder for faults within zones 1, 2 and 3 of its characteristic.
- INT1 (Intertrip 1): An intertripping relay, which can relay intertrip signals from both MP1 and MP2 to the remote end via a fibre-optic and microwave communications medium. When INT1 receives an intertrip signal from the remote end, it will generate and send a trip signal to trip relay 1 (TR1).
- INT2 (Intertrip 2): An intertripping relay, which can relay intertrip signals from both MP1 and MP2 to the remote end via pilots. When INT2 receives an intertrip signal from the remote end, it will generate and send a trip signal to trip relay 2 (TR2). Note that a different communication medium from INT1 is used for INT2 to reduce the chances of a common communications fault inhibiting both intertrips.
- TR1 and TR2 (Trip Relays 1 and 2): Simple auxiliary relays which take the trip signals from the main protection and intertrip relays and magnify them to a sufficient level to operate the circuit breaker. Two trip relays are used to ensure MP1 and MP2 have independent tripping mechanisms in case one should fail.
- CB1 (Circuit Breaker 1): Trip coils within the circuit breaker will be energised via the trip relays, operating the circuit breaker mechanism and opening the circuit breaker contacts interrupting the fault current.

2.4.5 Common Protection Problems

A transmission protection scheme consists of a complicated array of integrated components ranging from microprocessor-based relays to electromechanical trip relays and fibre-optic communications mediums. The complexity, together with the fact that protection schemes may lie dormant for many years without being called

upon to operate, can mean that potential problems with the protection scheme are not apparent until the protection is required to operate.

Common problems with transmission protection schemes fall into three broad categories:

- Dependability: the protection does not operate correctly when required.
- Security: the protection operates when not required, unnecessarily reducing the overall system security.
- System Restoration: the protection fails to restore the circuit.

A study of utilities by Working Group 13 of the IEEE Power System Relaying Committee has identified the most common mechanisms for protection maloperation within these three categories - illustrated in Table 2-1 [2] and described in the accompanying legend.

	Dependability			**Security**		**System Restoration**
	Failure to Trip	Failure to Interrupt	Slow Trip	Unnecessary Trip During Fault	Unnecessary Trip for Non-Fault Event	Failure to Reclose
Relay System [i]	A	N/A	B	C	D	E
Circuit Breaker [ii]	F	G	H	N/A	I	J

Table 2-1 Common problems with a protection scheme

i – Relay System defined as the protective relays, communication system, voltage/current measuring devices, and dc system for tripping up to the circuit breaker.

ii – Circuit Breaker is a generic term for any fault interrupting device

LEGEND for Table 2-1:

A) **Failure to Trip (Relay System):** Any failure of a relay system to initiate a trip to the appropriate terminal when the fault is within the intended zone of protection.

B) **Slow Trip (Relay System):** A correct operation of a relay scheme for a fault in the intended zone of protection where the relay scheme initiates the trip slower that the system design intends.

C) **Unnecessary Trip During a Fault (Relay System):** Any undesired relay-initiated operation of a circuit breaker during a fault when the fault is outside the intended zone of protection.

D) **Unnecessary Trip for Non-Fault Event (Relay System):** The unintentional operation of a protection relay which causes a circuit breaker to trip when no system fault is present: may be due to environmental conditions, vibration, improper settings, heavy load, stable load swings, defective relays or SCADA system malfunction.

E) **Failure to Reclose (Relay System):** Any failure of a relay system to automatically reclose following a fault if that is the design intent, e.g. DAR.

F) **Failure to Trip (Circuit Breaker):** The failure of a circuit breaker to trip during a fault even though the relay system initiated the trip command.

G) **Failure to Interrupt (Circuit Breaker):** The failure of a circuit breaker to successfully interrupt a fault even though the circuit breaker mechanically attempts to open.

H) **Slow Trip (Circuit Breaker):** A circuit breaker which operates slower than the design time during a fault following the trip initiation from the relay system.

I) **Unnecessary Trip for Non-Fault Event (Circuit Breaker):** The tripping of a circuit breaker due to breaker problems such as low gas, low air pressure, etc.

J) **Failure to Reclose (Circuit Breaker):** Any failure of a circuit breaker to successfully reclose following the reclose initiate signal from the relay system.

2.5 Transmission System Monitoring

A wide array of monitoring technologies have been developed to record protection operations, provide data on network performance and on the evolution of transmission network disturbances. The following sections present a brief resume of the monitoring technologies pertinent to this thesis:

2.5.1 Remote Terminal Units (RTUs) and SCADA

A Supervisory Control And Data Acquisition System (SCADA) provides engineers with indications of which equipment operated, what equipment is in or out of service and alarms when measured parameters move outside normal thresholds. This data is collected via Remote Terminal Units (RTUs) installed in each substation.

RTUs record the status of all substation devices by repeatedly scanning the signalling channels of each device, e.g. protection relay, trip relay, circuit breaker. If a signalling channel is found to have changed state, the RTU generates a time-stamped SCADA alarm indicating the new status. The scanning rate and resolution of the time stamps varies depending on the generation of the RTUs and the SCADA system itself. Older SCADA systems can generate time stamps that may not be very accurate or precise. This problem has been overcome in newer SCADA systems, which can provide sequence of events type data accurate to a millisecond.

To transfer the generated SCADA alarms to a central location, traditionally the control room, each RTU is scanned in a predefined order and the alarms uploaded to a central database. Advances in software and the increasing use of corporate Intranets and the Internet, have driven many utilities to provide users outside the control room, e.g. protection engineers, with access to this SCADA archive [3].

2.5.2 Digital Fault Recorders (DFRs)

The Digital Fault Recorder (DFR) is the preferred monitoring technology for disturbance analysis, since it is optimised for capturing and handling fault-relevant data and is fully independent from power system control and protection [4].

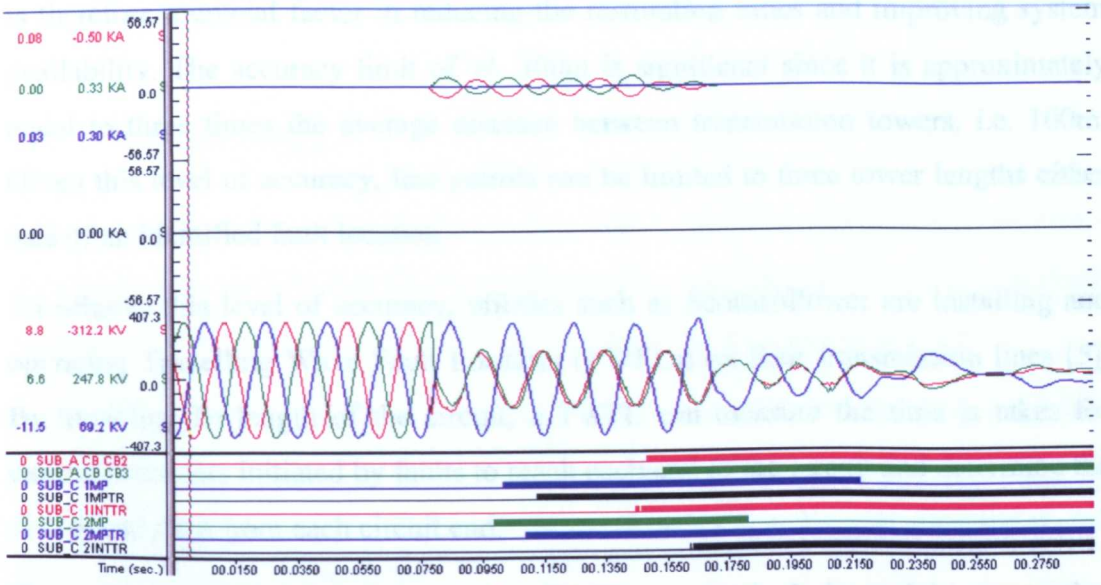


Figure 2-8 A fault record generated by a DFR

The data produced by DFRs comes in the form of fault records (see Figure 2-8) containing instantaneous measurements of the three phase analogue voltage and currents and recordings of digital protection scheme operations. Unlike RTUs, the DFR does not continuously scan the signalling channels it monitors; only beginning recording once a triggering signal is received. DFRs can be set to trigger on lower and upper voltage and current thresholds, rates of change or on operation of protection scheme components. DFRs use a higher sampling rate than most RTUs (typically above 5kHz) giving resolutions in excess of 1ms. Such high-resolution data is particularly useful to protection engineers since it provides a true picture of the operation and response of the power system during disturbances.

Each DFR will store fault records in a local storage medium until retrieved via modem dialup using proprietary software provided by the DFR manufacturer.

2.5.3 Travelling Wave Fault Locators (TWFLs)

A transmission circuit can stretch for upwards of 100 km, with the majority of the circuit traversing remote countryside as overhead lines. A permanent fault on such a circuit can require a significant amount of time and manpower to locate the fault

before restoration can commence. Accurate fault location, i.e. better than +/- 300 m, is therefore a crucial factor in reducing the restoration times and improving system availability. The accuracy limit of +/- 300m is significant since it is approximately equal to three times the average distance between transmission towers, i.e. 100m. Given this level of accuracy, line patrols can be limited to three tower lengths either side of an identified fault location.

To achieve this level of accuracy, utilities such as ScottishPower are installing and operating Travelling Wave Fault Locators (TWFLs) on their transmission lines [5]. By knowing the length of the circuit, a TWFL can measure the time is takes for current transients initiated by faults to reach each end of the circuit and determine the distance to fault from each circuit end.

The accurate fault locations provided by TWFLs are particularly useful to protection engineers as they provide the information necessary to validate whether distance protection detected the fault in the correct zone of operation.

2.6 Post-Fault Disturbance Analysis

As outlined earlier in section 2.4.5, transmission protection schemes are complex and can suffer from a variety of problems, which may not be apparent until the protection is required to operate. The impact of protection failures or mal-operations on system security can be significant. To reduce this risk a protection engineer must conduct a post-fault disturbance analysis following each protection scheme operation to validate the operation of protection and diagnose protection failures. Based on the results of this analysis, protection maintenance or reassessment of protection settings can be scheduled to alleviate any identified problems.

Fundamental to the post-fault disturbance analysis task are the SCADA and fault record data generated by the monitoring technologies described earlier. Using the SCADA data protection engineers can obtain a high-level overview of the protection devices that have operated across the transmission network to a resolution of 10ms. Fault records can then be used to provide a more detailed picture of the instantaneous

analogue currents and voltages and protection scheme digitals around the time of the protection operation to a resolution in excess of 1ms.

2.6.1 Terminology

Before describing the post-fault disturbance analysis process it is necessary to define the terminology used by ScottishPower Protection engineers, and adopted throughout this thesis, to describe the sub-sets of data used during the analysis process:

- The **Incident** data set contains all the data pertaining to a particular protection operating sequence. *Note that although a protection operating sequence will most often be initiated by a fault, it may be initiated as the result of a mal-operation.*
- The **Event** data set contains the data related to operation of particular protection scheme components and is a more focussed look at what actually happened during an incident. The primary indicators are SCADA alarms and there will be many events associated with an individual incident.

Other literature uses more generic descriptions such as the definition of an event used in [6]: "*a relay or switching operation or an inadvertent operation caused by changes in power system parameters measured at the substation*". Although the definitions adopted in this thesis may seem restrictive, they provide a clear distinction between the data sets used to assess each operation of a protection device. Furthermore, due to the similar nature and evolution of faults on transmission systems worldwide, the definitions are generically applicable regardless of the particular power system fault being analysed.

2.6.2 Manual Post-Fault Disturbance Analysis

To transform the raw data generated by monitoring devices into information that can be used during analysis, the protection engineers adopts the approach illustrated in Figure 2-9 and described in sections 2.6.2.1 to 2.6.2.4.

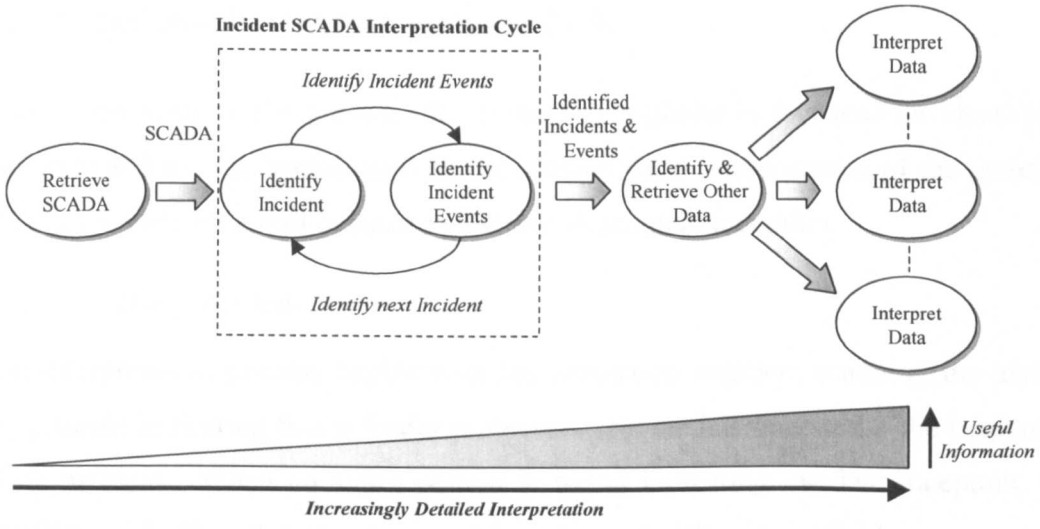


Figure 2-9 Manual post-fault disturbance analysis

Protection engineers will only conduct post-fault disturbance analysis once they are made aware that a protection operation has occurred. Notification that a protection operation has occurred often comes from the control engineer, or through regular checks of the SCADA and DFR data generated by the monitoring technologies. It must be emphasised that the protection engineer is unable to determine whether a fault caused the protection operation, until conclusion of post-fault disturbance analysis.

2.6.2.1 Retrieve SCADA

The protection engineer commences post-fault disturbance analysis by retrieving from the SCADA archive the alarms and indications recorded by RTU's throughout the transmission network during the period under analysis. In the case of post-storm analysis, this window may span several hours and many distinct incidents. At this stage in the process the retrieved data will contain many alarms and indications which have no bearing on the analysis. Only once this SCADA data is retrieved can analysis progress into the Incident SCADA interpretation cycle.

2.6.2.2 Incident SCADA Interpretation Cycle

During this stage in the process, the protection engineer is focussed on identifying incidents and events, beginning with the scanning and interpretation of the gathered SCADA alarms for alarm patterns indicating an incident inception.

2.6.2.2.1 Identify Incidents

The interpretation process begins with the protection engineer scanning the alarms for patterns indicating that a feeder protection scheme has operated – the beginning of an incident. When an alarm pattern is found indicating incident inception, the start time and affected feeder are noted from the protection alarm fields.

The protection engineer then continues scanning the SCADA, identifying any alarms occurring after the incident inception and on the affected feeder. As each incident related alarm is identified, it is noted and added to a grouped data set of incident related alarms.

Whilst scanning the alarms the protection engineer is also interpreting the alarms for patterns indicating DAR activity. If DAR has been initiated and the circuit re-energised onto a fault, the protection will again see fault current and operate – a protection operating sequence distinct from the first, which must also be analysed. In this instance, the protection engineer will conclude the first incident data set and begin grouping the alarms occurring following the second protection operation under a second incident data set. If no DAR activity has been initiated, the protection engineer will conclude the incident once they are satisfied that the protection operating sequence is over.

At this stage, an incident will have been identified and all incident related alarms grouped into an incident data set for the next stage in analysis. If DAR has been initiated, two incidents will have been identified related by a common permanent or persistent fault.

2.6.2.2.2 *Identify Pertinent Events*

The grouped set of incident alarms (two sets in the case of a DAR operation) is then interpreted to identify the events of interest to the protection engineer, e.g. circuit breaker operations, protection operations and intertrips.

Initially the protection engineer performs a first pass of the alarms, interpreting the incident alarms for individual alarms or patterns of alarms indicating events of interest such as protection relay operations or changes in circuit breaker status. This results in the recording of 'Low-Level' events indicating the nature and time of pertinent events in the protection operating sequence.

The engineer then conducts a second pass of the grouped incident alarms taking into account the identified low-level events. At this stage in analysis the protection engineer is looking to identify 'High-Level' events indicating whether the protection scheme operated correctly and if there could be any anomalies.

On completion of event identification, the protection engineer resumes incident identification, beginning with alarms occurring after the last incident start. The Incident SCADA interpretation cycle continues until no more alarms remain.

2.6.2.3 Identify and Retrieve Other Data

Until now the protection engineer has only used SCADA data, which has enabled the circuits affected by each incident to be identified along with the time when the protection operation was initiated. This geographical and temporal locality information allows the monitoring devices on the affected circuits to be identified and their storage mediums queried for incident related data.

The primary source of additional disturbance data is DFRs since they capture the analogue voltages and currents on the circuit being monitored and the changes in status of the components in the protection scheme monitoring the circuit.

Knowing where the feeders on which the incidents have occurred, the protection engineer can access the DFRs at each circuit end and retrieve the fault records archiving them at a central location ready for analysis. A similar procedure is followed for retrieval of data from other monitoring technologies such as TWFLs.

The protection engineers will again use the incident information to identify the devices of interest and retrieve the incident related data ready for analysis.

At this stage, the protection engineer has gathered a comprehensive data set mainly consisting of SCADA alarms and fault records and often with additional data such as from TWFLs. The engineer will have interpreted the SCADA data to identify incidents and events and have collated additional data such that data pertaining to an individual incident is grouped as such.

2.6.2.4 Interpret Additional Data

Thus far interpretation has been limited to SCADA data and has identified the disturbance locales, when the protection detected the fault and the key protection operating events. Although useful, this information does not provide the detail necessary for a protection engineer to determine the cause of protection operations and to validate whether the protection operated correctly. This is achieved through interpretation of the additional non-SCADA data retrieved during the previous stage.

Fault records are the first to be analysed since they are the most likely to shed light on what caused the protection operation. Only when fault record analysis has been completed will data gathered from additional sources be analysed.

Analysis of the fault records relies on the use of software tools provided by the DFR manufacturers. Combined with an extensive knowledge of the protection schemes and experience the protection engineer will use the software to interpret the captured data. The software can provide accurate measurements of time and magnitude which are useful for identifying parameters such as fault clearance times, peak fault current, minimum voltage etc.

Data from sources such as the TWFL will require relatively little interpretation as the data is formatted as a text file with the key attributes, such as distance to fault, already highlighted. In cases where it isn't the software provided with the monitoring device is used.

The interpretation of fault records and other data concludes the manual post-fault analysis process. The protection engineer now has a comprehensive set of data and

information on each incident. Using this, they will use experience and knowledge of the protection schemes to validate the protection operations.

2.6.3 The Data Overload Problem

A modern SCADA system can generate tens of thousands of unique alarms and indications due to the significant number of signalling points monitored by RTUs across a transmission network. DFRs can be set to trigger and generate a fault record whenever the monitored parameters move outside the triggering thresholds regardless of whether the change in conditions is due to a fault or the routine operation of the transmission system. Both these characteristics of the monitoring technologies commonly used by protection engineers combine to present a significant data overload problem during post-fault disturbance analysis.

Although the monitoring devices will capture data of direct relevance to the post-fault disturbance analysis being conducted, e.g. fault records and SCADA alarms pertaining to the circuit affected by the fault, additional data will be generated by devices outside the affected zone. This occurs due to the effect a fault has on the surrounding network.

A fault on an item of plant or circuit will often lead to voltage dips and increases in current magnitude in its immediate vicinity until the fault is cleared by the protection. The devices monitoring the faulty equipment will see the change in conditions during the evolution of the fault and generate data directly related to the disturbance. However the monitoring devices on the circuits surrounding the fault will also measure a change in network conditions albeit not as severe as in the immediate vicinity of the fault and, providing the change is beyond trigger settings, generate data. This data is of no immediate use to protection engineers and only clutters the data set available post-fault.

The problems of data overload are significant after major disturbances, such as storms, where large numbers of faults have occurred resulting in tens of thousands of SCADA alarms and many hundreds of fault records of which only a comparatively small sub-set are of immediate interest. Furthermore, with the reduced staffing levels in modern privatised utilities the few remaining protection engineers [7] are faced

with more data than can be processed and assimilated within the timescales permitted.

2.7 Chapter Summary

This chapter has provided an overview of power systems relevant to this thesis. Specific attention has been paid to transmission protection schemes and the monitoring technologies that provide engineers with data on the performance of transmission protection.

The chapter has also described one of the key responsibilities bestowed on a protection engineer: post-fault disturbance analysis. The protection engineer fulfils their responsibility by following a structured process of data retrieval, interpretation and collation. The terminology used during this process, and to be used throughout the remainder of the thesis, has been introduced. This was followed by a description of each stage of the post-fault disturbance analysis process, focussing on the data retrieval and interpretation requirements.

2.8 Bibliography

- B.M. Weedy, "Electric Power Systems", Third Edition Revised, John Wiley & Sons, 1986.
- "Protective Relays Application Guide", Third Edition, GEC Measurements, Stafford, UK.

2.9 References

[1]. "An Introduction to the Protection of Transmission and Distribution Systems", Electricity Training Association, 1997.

[2]. W.M. Carpenter (Chairman), "Transmission Protection Relay System Performance Measuring Methodology", IEEE/PSRC Working Group 13, September 1999.

[3]. M. Srinivas, N. Sreekumar, K.V. Prasad, "SCADA-EMS on the Internet", IEEE Conference on Energy Management and Power Delivery, March 1997.

[4]. "Fault and disturbance diagnosis data analysis including intelligent systems – Part 1: Existing Practice", WG 34.03 Cigre Study Committee 34 – Protections and Local Control, 2000.

[5]. P.F. Gale, J. Stokoe, P.A. Crossley, "Practical experience with traveling wave fault locators on Scottish Power's 275 & 400KV transmission system", IEE Developments in Power System Protection Conference, 25-27 March 1997.

[6]. L.E. Smith, "Analysis of Substation Data", Report from Working Group I-19 to the Relaying Practices Subcommittee of the IEEE Power System Relating Committee, Fault and Disturbance Analysis Conference, 2002.

[7]. W. Laycock, "Protection Engineer Shortage", IEE Power Engineer, February/March, 2004, p47.

Chapter 3: Fundamentals of Intelligent Systems and their Application to Post-Fault Disturbance Analysis

3.1 Chapter Overview

This chapter introduces intelligent systems discussing the fundamental technologies relevant to the research described later in this thesis. The knowledge engineering process required to capture the knowledge implemented within an intelligent system is also presented. The chapter concludes with an overview of the intelligent systems and decision support tools available to protection engineers during post-fault disturbance analysis. The required enhancements to these tools are also discussed.

3.2 Intelligent Systems

As technological advances in computer hardware, and software techniques, have come to fruition, researchers have attempted to exploit these advances to create intelligent systems capable of emulating a humans' reasoning processes, as applied to the solution of a certain problem. The term 'Artificial Intelligence', or 'AI', has become the term synonymous with this endeavour.

The debate over what exactly constitutes artificial intelligence is still ongoing and is likely to remain so as long as a widely accepted definition for intelligence eludes us. However, in relation to the research described in this thesis, the term 'AI' is defined as:

> *'the ability of a software system to solve a particular engineering problem, which would otherwise require specialists' knowledge'.*

For example, in order to build an intelligent system for automating alarm interpretation, the expert's knowledge has to be captured for the particular application (e.g. gathering different types of alarm data and emulating the engineer's reasoning for processing this data to provide meaningful conclusions).

Early attempts at using AI for problem solving focussed on the development of intelligent systems aimed at solving broad classes of problems using generalised reasoning steps. However, it was soon found that such general systems performed poorly on problems of real-world complexity and applicability. A solution to this problem was found to be to focus in on a specific problem area and to provide the

intelligent system with specific, high-quality knowledge, about the problem area. This approach led directly to the development of expert systems.

3.2.1 Expert Systems

Defining an 'expert system' is not the most straightforward of tasks, with various positions being advanced by different researchers. However, for the purposes of this thesis it is appropriate to define an 'expert system' as:

> *'any software system that employs knowledge about its application domain and uses an inferencing (reasoning) procedure to solve problems that would otherwise require human competence or expertise'.*

By exploiting the same knowledge and problem-solving techniques that make domain experts effective in solving problems in their field of expertise, expert systems have a number of associated advantages:

- They can act as a repository for knowledge captured from a number of domain experts, thereby ensuring the domain experts knowledge and expertise is not lost when they leave a company.
- Combined with an inference engine, the captured knowledge can be used to reason about a given data set and provide solutions or decision support to an engineer much faster than would be achieved through manual data analysis.
- Given the same data set, the solutions provided are consistent – a quality which cannot be attributed to human experts.

During problem solving and data analysis, domain experts will use a number of different types of knowledge, including: heuristics (or 'rules of thumb'), knowledge derived from similar problems (or cases) and models representing the behaviour of components or a process. To reflect this diversity a range of expert systems have been developed, each utilising a particular type of knowledge. However, before knowledge can be used within in an expert system it must first be acquired.

3.2.1.1 Knowledge Engineering

The process of capturing knowledge from a domain expert and modelling that knowledge so it can be structured for use within expert systems is termed knowledge engineering. This is a very important aspect in an intelligent system's development, as it is only as good as the knowledge it contains.

Domain knowledge can be acquired from a number of different sources including documentation, design and functional specifications and books. Knowledge can also be gleaned from interviews with domain experts, a process known as knowledge elicitation.

During the preliminary stages of knowledge elicitation, unstructured interviews are often used as they provide an effective means of scoping the problem domain. However, abiding solely by an unstructured knowledge elicitation strategy can result in a patchy coverage of the problem domain and inconsistent knowledge. An accepted solution is, having scoped the problem domain using unstructured interviews, structured interviews should be conducted where the elicitation process is planned and directed by the person eliciting the knowledge. These structured interviews are often recorded electronically, e.g. taped, and a transcript created.

On some occasions, the knowledge transcripts alone will provide sufficient information to commence structuring the domain knowledge ready for implementation within the expert system. However, in cases where a significant volume of knowledge has been captured and the problem domain is vast, it is often advisable to model the elicited knowledge.

A popular methodology for knowledge modelling is CommonKADS, which supports the analysis, specification and development of intelligent systems [7]. One aspect of CommonKADS is using six models:

- **Organisation Model:** Describes the functions, tasks and bottlenecks in the organisation environment the intelligent system will have to function;
- **Task Model:** Specifies, at a general level, how the function of the system is achieved through a number of tasks that the system will perform;

- **Agent Model:** Describes the capabilities and characteristics of the agents within the organisation. An agent is an executor of a task. It can be a human, computer software or any other entity capable of executing a task.
- **Communication Model:** Describes the exchange of information between the different agents involved in executing the tasks described in the Task Model.
- **Expertise Model:** Models the problem solving knowledge used by an agent to perform a task. This model is split into sub-levels: domain level, inference level and task level.
- **Design Model:** Describes the architecture and design of the KBS.

Together, the organisation, task and agent models analyse the organisational environment and the corresponding requirements for an expert system. The expertise and communication models yield the conceptual description of problem solving functions and data that are to be handled and delivered by the expert system. The design model converts into a technical specification that can be used as the basis for expert system implementation. It should, however, be noted that not always do all models needs to be constructed. This depends on the scope and goals of the project.

3.2.1.2 Knowledge Based Systems

Knowledge Based Systems (KBS) [2] are particularly suited to the emulation of an expert's reasoning where the expert uses a combination of theoretical understanding of the problem domain and a collection of problem solving heuristics to reason about a problem and reach a solution.

The domain knowledge and heuristics can be acquired from the experts, modelled and then structured as rules within the KBS knowledge base. Although other forms of knowledge representation exist, such as objects and semantic networks, rules are by far the most common knowledge representation technique used within KBS due to their relatively straightforward application. They can be described as simple 'if' – 'then' statements, of the form:

IF (premise) THEN (conclusion)

The typical components of a KBS, as shown in Figure 3-1, comprise of a *knowledge base*, which contains the problem solving domain knowledge; the *working memory*, which contains the observed facts about the problem domain under consideration upon which the inference engine will reason; the *inference engine*, which drives the KBS, deciding which rules to fire, how they will be applied during reasoning and finally provides possible solutions; the *user interface* through which the user interacts with the KBS.

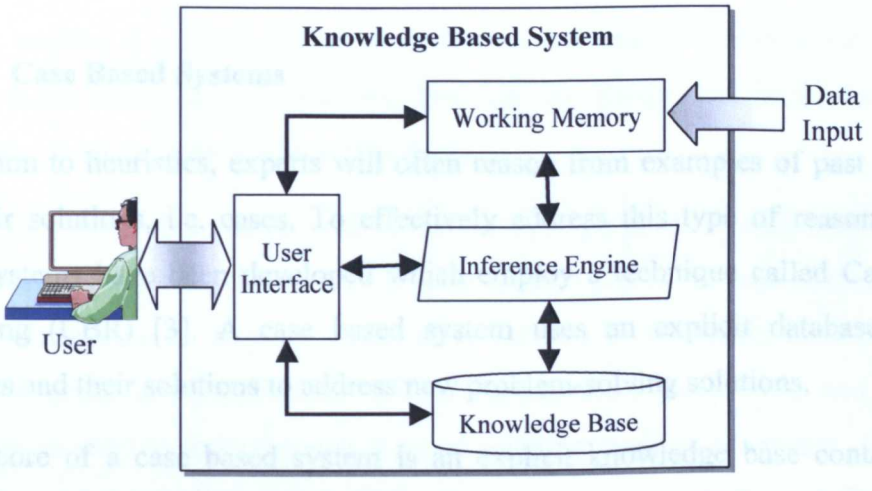


Figure 3-1 Architecture of Knowledge Based Systems

The inference engine is at the hub of a KBS, providing the mechanism for extracting the appropriate knowledge from the knowledge base and combining it with the observed domain facts to generate a diagnosis. Two commonly used inference mechanisms are:

- Forward chaining (or data driven) where the inference engine begins with the observed facts of the problem domain and infers the diagnosis.
- Backward chaining (or goal driven) which starts from the goals to be solved, and given a set of rules, determines what evidence is required to prove them.

The search space of observed domain facts that the inference engine must investigate to reach a diagnosis can be vast. Two of the most common search techniques employed to decide which to investigate are depth first and breadth first:

- In a depth first search, when a possibility is examined, all of its successors are investigated as far as possible. Only if this is unsuccessful will the other possibilities be considered.
- In a breadth first search, all the possibilities are explored in a level-by-level fashion. Only when no more possibilities are to be explored does the algorithm move onto the next level.

The breadth first approach ensures the best solution is always found, however, a depth first search, if directed in some manner, will be faster.

3.2.1.3 Case Based Systems

In addition to heuristics, experts will often reason from examples of past problems and their solutions, i.e. cases. To effectively address this type of reasoning, case based systems have been developed which employ a technique called Case Based Reasoning (CBR) [3]. A case based system uses an explicit database of past problems and their solutions to address new problem-solving solutions.

At the core of a case based system is an explicit knowledge base containing an expert's solutions to a number of past problems - the cases. When presented with a new problem the case based reasoner must search this case base and retrieve the most appropriate cases for use in solving the new problem. The most appropriate cases are determined by looking for similarities between the cases and the current problem. Each case is assigned a set of indices based on their significant features thus enabling a more rapid search through the knowledge base to identify the cases that have most features in common with the current problem.

More often than not, none of the cases within the case base exactly match the problem that the case based system is called on to solve. To address this, the system will select the most suitable from those retrieved and modify it in order to reflect the differences between the cases. This may not guarantee the generation of an acceptable solution and further modifications may be required. Each proposed solution is saved in the case base as a new case with its indices.

3.2.1.4 Model Based Systems

On many occasions experts, particularly in engineering fields, are called upon to diagnose failures within physical systems containing many individual, yet, interconnected components. Both knowledge of how each individual component is expected to work and how the entire system is expected to operate is utilised in such cases. The most popular structure for this knowledge is models.

Depending on the nature of the system, the models utilised will not only vary in complexity but also in model type, i.e. algorithmic, functional, qualitative and physical models. A number of data structures can be used for representing the causal and structural information in models, however the design and implementation of each model is beyond the scope of this thesis.

Intelligent systems that reason with models for the purposes of diagnosis utilise an AI technique called Model Based Diagnosis (MBD) [4]. MBD was first used in the field of electronics to diagnose component failures in electronic circuits through device and circuit analysis. Models of both the individual components and how they are interconnected to form the circuit are required to perform diagnosis. To explain the principles behind MBD the multiplier-adder system used in [5], and illustrated in Figure 3-2, will be utilised.

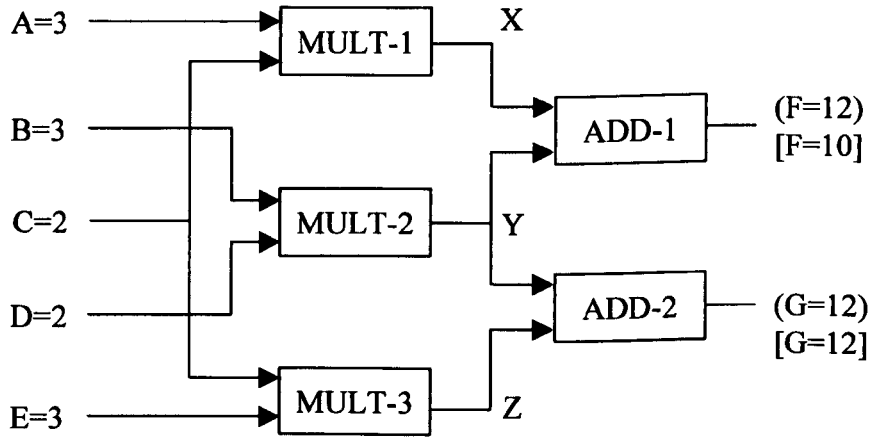


Figure 3-2 Example of multiplier-adder test system

In Figure 3-2 the behaviour of each component can be represented by a set of expressions that capture relationships between values at the terminals of the device, e.g. ADD-1 can be represented by: X+Y=F, X=F-Y and Y=F-X. The inputs to the circuit are marked A-E and the circuit outputs F and G, with () indicating the expected result and [] the actual result.

It is clear that the output from component ADD-1 is not as expected indicating a faulty component, or set of components, within the circuit. To identify the faulty component(s) the model based system must trace back through the circuit and, using the component models (in this case simple relationships) of expected behaviour identify where discrepancies between the component inputs and outputs lie given the circuit inputs and outputs.

In addition to diagnosis, the consistency-based reasoning followed by a model-based system will validate if the output from a physical system is as expected given the inputs. Furthermore, with adaptations to the MBD reasoning process abductive models (that is models of faulty behaviour) can also be included to provide knowledge of potential fault mechanisms. These can be used to enhance the diagnosis provided by the systems.

Despite the advantages of model based systems, a significant disadvantage is that a theoretical understanding of the devices and their explicit modelling are both essential. Consequently, the knowledge acquisition process can be quite demanding and the resulting software code large, impacting on the speed with which diagnoses can be generated.

3.2.2 Hybrid Intelligent Systems

Although the AI techniques employed within the intelligent systems discussed thus far have their own advantages and specific application areas, very often they are insufficient to resolve the real data analysis and decision support problems facing the engineering industry [6]. The use of Artificial Neural Networks (ANNs) within real time decision support, such as alarm processing within a control room, is a typical example where the ANN's lack of a structured knowledge representation and its inability to explain the reasons for the conclusions reached have been highlighted as

contributing to an operators lack of confidence in the system [7]. The time and resources associated with knowledge elicitation for expert systems are another practical problem often cited as limiting the real world application of the technology.

To deal with the increasing complexity of engineering problems an integrated approach is needed where the merits of individual techniques can be exploited to overcome the shortcomings in other techniques. Hybrid intelligent systems provide the hybridization or fusion of the individual techniques necessary to achieve this.

Hybrid intelligent systems began to emerge in the 1990's and have become an active research field within the wider AI community. Within this research field, hybrid intelligent systems are commonly grouped into three broad classes [7]:

- *Transformational hybrid architectures* are where one technique is used to transform one form of representation into another. From a practical perspective, they are used in situations where knowledge required to accomplish the task is not available and one AI techniques relies on another for its reasoning or processing. A good example is where ANN's are used to transform continuous numerical data into discrete data sets which can then be used as input to a KBS and processed further via rules and inference. In this architecture it is important to note that each technique will remain as individual computational units but will be connected in series.

- *Fusion hybrid architectures* combine different techniques into one computational unit with the data structures and knowledge representations of each technique shared and hard wired to the other. This enables one AI technique to augment its reasoning process in a manner which allows it to overcome its weaknesses. That is, unlike in transformational architectures, the transition from one representation to another does not occur naturally.

- *Combination hybrid architectures* put a number of AI techniques on a side-by-side basis. There is no fusion of one technique to another, nor is data transformed by one technique into a form suitable for another. Instead the reasoning capability of the hybrid architecture comes from the combined roles of each technique in the overall problem-solving process. Each technique retains its separate identity and is used at a level within the architecture where its strengths can best be exploited.

This not only facilitates complex problem solving but also provides a closer synergy with human reasoning.

Although the aforementioned classes of hybrid intelligent systems are commonly referenced in literature on the subject, for the purposes of the research described in this thesis the combinational hybrid architecture is the most relevant. The combinational hybrid architecture is the most pertinent to this thesis, since a wide array of standalone intelligent systems, employing a number of different techniques, have already been developed for utilities and are currently in use. It will be shown later in this thesis, that through the use of multi-agent systems as a platform for combinational hybrid architectures, the individual functionality provided by each system can be leveraged to provide an enhanced overall level of functionality. To this end, and to provide clarification, for the remainder of this thesis a hybrid intelligent system will be defined as:

> *'any integrated suite of software components providing an overall problem solving capability through sub-division of the problem to two or more intelligent reasoning techniques.'*

3.3 Decision Support Systems

As reported earlier in chapter two, the power industry is suffering from a lack of protection engineers and this combined with the problems of data overload during post-fault disturbance analysis, puts a significant strain on the engineers. As a result many of the protection problems the engineers are trying to identify can be overlooked negating the benefit of the post-fault disturbance analysis process.

Some relief can be achieved through the provision of intelligent systems that automate the data interpretation tasks and assist the protection engineer by extracting the pertinent information from the raw data and presenting it in an amenable format. Such systems are commonly referred to as Decision Support Systems (DSS) since the information they generate assists an engineer in deciding the next course of action, i.e. in the case of post-fault disturbance analysis, identification of unusual events that require further investigation.

The following sections will introduce the DSS and associated tools utilised by ScottishPower protection engineers during post-fault disturbance analysis.

3.3.1 Alarm processors

During the preliminary stages of post-fault disturbance analysis the protection engineer manually interprets SCADA alarms to determine the incidents and events. This information helps the engineer decide where to focus further data retrieval and interpretation. However, the sheer volumes of alarms make this manual alarm processing task time-consuming and prone to human error.

This alarm overload problem has long been associated with control rooms where control engineers can be overwhelmed by the rate at which alarms are received [8]. As early as the 1980s, expert systems were developed to interpret alarms automatically in real-time and provide control engineers with summarised messages, thereby reducing the data volume. It is only within the last decade that the alarm processing needs of protection engineers have been addressed.

To assist the ScottishPower protection engineers an alarm processor entitled APEX (Alarm Processing EXpert system) was developed and implemented in the early 1990's [5]. The system operates online and was an adaptation of a previous alarm processor developed for control room environments [10].

APEX is a KBS with the architecture shown in Figure 3-3. This architecture is fundamentally the same as the generic KBS architecture illustrated in Figure 3-1 except for the addition of a topology and a possible alarm database listing all alarms which can be generated by the SCADA system and may be received by APEX.

The topology database provides a representation of the network connectivity which is read into memory when APEX is initialised. While APEX is running, changes in the network indicated by received plant status alarms are reflected in this topology representation. APEX can therefore provide protection engineers with information on the circuits that have been isolated by protection operations.

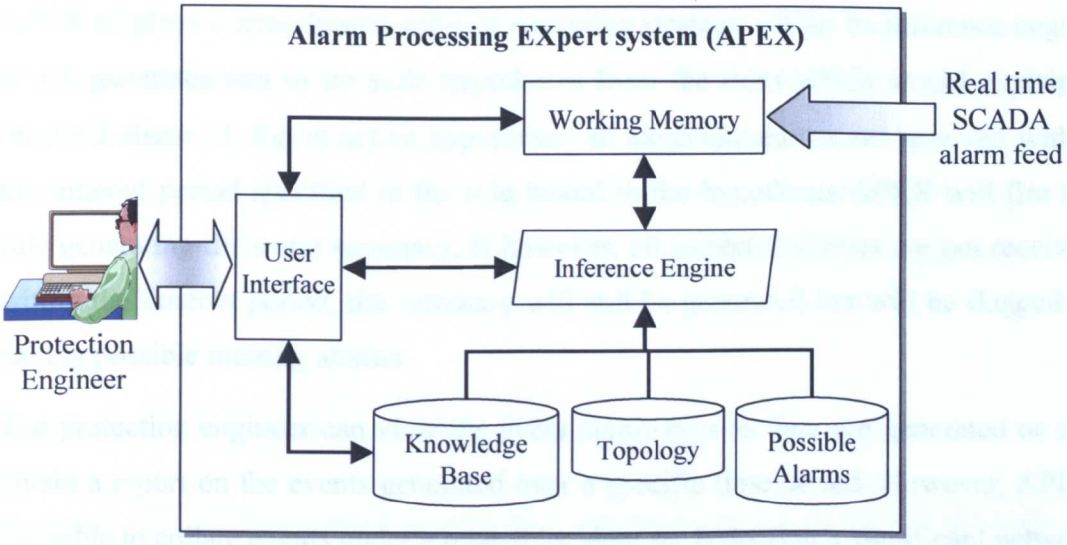


Figure 3-3 APEX architecture

The APEX knowledge base consists of rules of the form illustrated in Figure 3-4. These rules start with the event summary to be displayed when the rule fires and conclude with a list of the expected alarms. The rules use wildcards, e.g. <StationName>, specifying the alarm field which is required: this allows the rules to be generic.

Event "Protection operation at *<StationName>*

Class isolation
Summary protection
Priority 25
Timeout 150000
Expect
{
Alarm "FIRST MAIN PROT OPTD" ON *<StationName>*
Alarm "FIRST MAIN PROT OPTD" OFF *<StationName>*
Alarm "SECOND MAIN PROT OPTD" ON *<StationName>*
Alarm "SECOND MAIN PROT OPTD" OFF *<StationName>*
}

Figure 3-4 Typical APEX rule for protection engineers

APEX employs a hypothesise-and-test reasoning strategy within its inference engine which generates sets of possible hypotheses from the rules which would explain a received alarm. If, for an active hypothesis, all expected alarms are received within the timeout period specified in the rule linked to the hypothesis APEX will fire the rule generating the event summary. If however, all expected alarms are not received within the timeout period, the summary will still be generated but will be flagged as having possible missing alarms.

The protection engineer can view the event summaries as they are generated or can obtain a report on the events generated over a specific time period. However, APEX is unable to collate events under a related incident so, following a significant network event, manual collation of the event summaries and further interpretation is required in order to identify the incident. This is due to the original APEX inference engine being optimised for real-time alarm processing in the control room where reduction in alarm volumes is the priority and the control engineer can use the mimic diagram to ascertain the incidents. Despite APEX not being optimised for post-fault alarm processing, the ScottishPower protection engineers have found APEX a useful tool.

At the time of implementing the version of APEX for the protection engineers an existing fault diagnosis DSS, entitled RESPONDD [11], was also adapted from a control room application to provide fault diagnostic support to protection engineers.

RESPONDD used detailed knowledge of the network and protection schemes to interpret SCADA data and provide post-fault diagnosis. The results of this interpretation were textual summaries which identified the faulted phases, whether the fault was permanent or temporary and protection equipment which failed to operate. A detailed discussion on the reasoning approached used by RESPONDD can be found in [11].

Although RESPONDD generated information of interest to protection engineers, the system proved difficult to maintain since the knowledge of protection schemes and network topology had to be kept current if the correct diagnosis was to be reached. As a result, the system quickly became obsolete and was not used by protection engineers. This obsolescence was hastened by the introduction of DFRs and the protection validation toolkit which will be discussed in section 3.3.3.

The alarm processing and fault diagnosis DSS introduced above are those employed by ScottishPower and, although these systems have been adapted for other utilities, they only represent a small subset of the intelligent systems developed for these purposes. For the purposes of this thesis it is sufficient to merely note that other systems exist which could, with minor modifications, have been employed within ScottishPower.

3.3.2 Fault Record Analysis Engines

Although the ScottishPower protection engineers begin post-fault disturbance analysis with SCADA alarms, protection engineers from other corners of the globe typically begin analysis with the fault records generated by DFRs [12]. It is unclear why this difference arises but it is most probably due to a reduced number of DFRs in use by other utilities, limiting the number of records which must be interpreted thereby making DFR analysis a more desirable option. For example, as of the year 2000, Reliant Energy HL&P in the USA only had 33 DFRs [13] compared with 100 deployed within ScottishPower.

Regardless of whether DFR analysis is conducted at the outset or later in the post-fault analysis process, the protection engineers would benefit from automated fault record analysis. However, before fault record analysis is discussed the mechanisms by which fault records are retrieved from DFRs must be introduced.

3.3.2.1 Fault Record Retrieval

The majority of DFR manufacturers software is intended for use at a central location, such as the utility's head office, where communications from the master station running the software to the DFRs will be via the public telephone network. Each DFR has a unique telephone number which the software can dialup to establish a connection. The number of available phone lines and the capabilities of the software limit the number of DFRs that can be connected to at any one time.

DFR manufacturers offer a range of fault record retrieval options:

- **Manual Retrieval:** The protection engineer can select a DFR from a list and initiate dialup. If the connection is successful, any records not previously retrieved will be uploaded to the master station and archived at head office.
- **Autopolling**: This is a feature popular with utilities with large numbers of DFRs as the software autopolls all the DFRs over a time period set by the protection engineer and automatically retrieves any new fault records.
- **Automatic Upload:** This is where the uploading of new fault records to the master station is initiated by the DFR. This is desirable when there are only a few DFRs, since with large numbers the limited number of communications channels can quickly become clogged during periods of high activity, such as storms, just when they are needed most.

The choice of retrieval mechanism is down to the protection engineer and is dictated by the number of DFRs and the available communications channels. At ScottishPower, the software is set to autopoll the DFRs over a 24-hour period with the protection engineer still having the option of initiating a manual retrieval.

It is important to note that autopolling only ensures retrieval of all the data and does not prioritise the retrieval based on the evolving system conditions. Furthermore, the limited storage capacity of many older generation DFRs mean that if fault records are not retrieved quickly newer records may overwrite them, a problem particularly apparent during storms. Although this problem has been largely overcome with the greater storage capacity of newer generation DFRs, populations of older generation DFRs may still exist in many utilities. The research described later in this thesis will propose an effective means for ensuring prioritised retrieval of fault records, thereby significantly reducing the possibility of disturbance related fault records being overwritten in older generation DFRs.

3.3.2.2 Fault Record Analysis

The protection engineer is now confronted with a large number of fault records, which have either been retrieved manually or by autopolling, and must be interpreted to ascertain what has happened.

Manual interpretation would require the use of the DFR manufacturers software to visualise each record. These visualisation tools vary depending on manufacturer but have a number of generic features:

- Scaling to enable zooming in or out on areas of interest
- Configurable colour schemes allowing different colours to be assigned to the different DFR channels, e.g. Red, Yellow and Blue for the channels recording red, yellow and blue phase voltages.
- Cursors which provide the value of each channel at a given instance in time.

Using their knowledge of how faults manifest themselves and the protection schemes, the protection engineer can use these visualisation tools to gain the information they require. However, in similar to other aspects of manual post-fault analysis, this process is laborious.

To help the protection engineers decide which fault records contain disturbance related data, some manufacturers are incorporating expert systems, similar to that developed by Kezunovic [14], into their software which classifies the records as they are retrieved.

Using signal processing algorithms to identify the faulted phase(s) and simple parameter calculations, the pre-fault, fault and post-fault currents and voltages and the protection digitals are input into an expert system, which uses simple rules to classify the fault. The output of the expert system are plain English descriptions of the recorded event, such as "The disturbance is a phase yellow to ground fault", which can be displayed against each fault record. This, combined with different display colours for each fault type, provides the protection engineer with the visual cues necessary to decide which records to focus analysis on.

At the time of writing this thesis, this automated classification facility was not available within the DFR software used by ScottishPower protection engineers. Instead they had to rely on a simplified fault record interpretation tool, which extracted the parameters of interest and output them to a text file. This software used simple algorithms to identify the faulted phases, fault clearance times and protection operating times.

3.3.3 Protection Validation Toolkit

Perhaps the most onerous task is validating whether the protection scheme operated correctly and, if incorrect operation is discovered, deciding which component malfunctioned and why.

To assist with this task, many protection engineers have at their disposal libraries of protection models and simulation packages, including: MATLAB, SIMULINK, C++, Java and software models of a protection relays' dynamic behaviour (Dynamic Protection Models or DPM) [15] Using the analogue voltages and currents captured in the fault records as input to the models, the protection engineer can determine the expected behaviour of the protection. This behaviour can then be compared against the actual behaviour indicated in the fault record and the protection operation validated. This is a laborious process requiring the protection engineer to be competent at using each simulation package and to be familiar with the models.

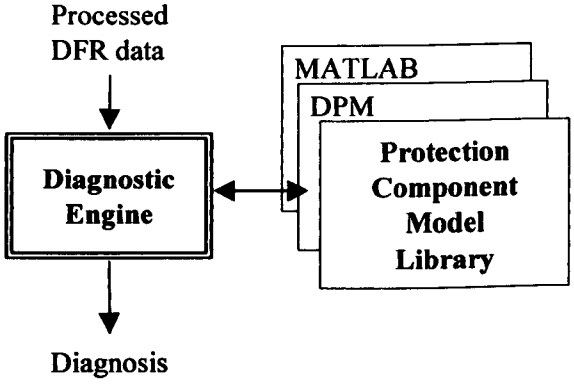


Figure 3-5 The diagnostic engine draws on a library of protection models

To assist protection engineers with the validation of protection performance a unique toolkit has been developed that utilises the existing model library and processed DFR data as input to a diagnostic engine using MBD [16], as illustrated in Figure 3-5.

The toolkit is based on an earlier diagnostic engine developed by Bell et al [17], which was deployed at ScottishPower and used consistency based diagnosis; diagnosis only using models of correct behaviour. This diagnostic function is achieved by detecting deviation from nominal behaviour and identifying the

components of the system whose failure could be logically responsible for the deviation.

The consistency-based approach has a number of advantages. As diagnosis is based on knowledge of function rather than malfunction, novel faults, i.e. faults never experienced before, can be diagnosed. The failure of more than one component of the protection scheme is also considered.

Where consistency based diagnosis fails is that it cannot provide the cause of the failure. For example, if a protection relay operates slowly, a consistency-based system would only be able to detect that it had malfunctioned but not how.

To overcome this failing, the new toolkit extends the consistency based diagnosis algorithm employed in the earlier system to include abductive diagnosis; diagnosis using models of faulty behaviour. This new diagnostic process involves the use of consistency-based methods to identify components that may have malfunctioned. Models of faulty behaviour are then used to identify particular fault modes.

The toolkit is a standalone system that requires the user to input processed fault records and select the protection scheme to be modelled – the system then runs the appropriate models and generates a diagnosis. The same fault record interpretation tool utilised by ScottishPower protection engineers and mentioned in section 3.3.2.2 provides the processed fault records.

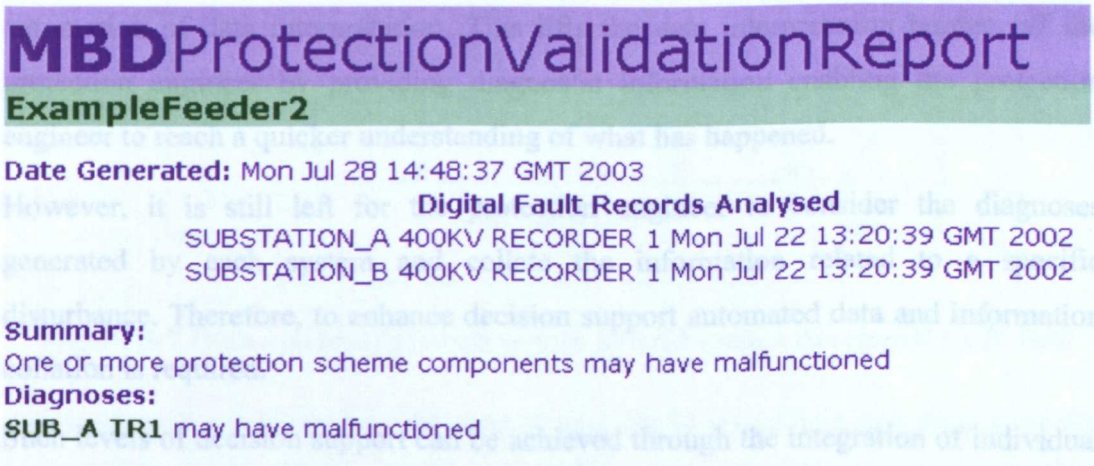


Figure 3-6 Protection validation report for a disturbance

The diagnosis produced by the toolkit is formatted suitable for viewing via an Internet browser and provides a summary and diagnosis, as illustrated in Figure 3-6.

The protection validation report illustrated in Figure 3-6 is the result of analysis of two fault records from the substations at each end of a circuit: SUBSTATION_A and SUBSTATION_B. Using a representation of the protections scheme and models of its components, the diagnostic engine has identified that Trip Relay 1 (TR1) at substation A may have malfunctioned. Note that there is no failure mechanism identified since there were no failure models available for the trip relay.

The information provided in these reports is extremely useful since it both validates the protection operation and identifies any components that may have malfunctioned. Furthermore, this is achieved without the protection engineer having to run any models or simulations. The protection engineer can use this information as the justification for further investigations into why the failure occurred, hopefully leading to a remedy.

At the time of writing this thesis, this toolkit is only available to ScottishPower protection engineers. Other similar systems have been developed, such as Timely [18], however the extent to which they have been deployed is not clear.

3.3.4 Integrated Systems for Decision Support

The DSS available to protection engineers provide decision support through the automation of data interpretation. This lifts the data interpretation burden off the protection engineer by providing diagnostic information enabling the protection engineer to reach a quicker understanding of what has happened.

However, it is still left for the protection engineer to consider the diagnoses generated by each system and collate the information related to a specific disturbance. Therefore, to enhance decision support automated data and information collation is required.

Such levels of decision support can be achieved through the integration of individual systems into an architecture that facilitates the communication of information between each system. Each system can be then utilise the received information to

decide on an appropriate action, e.g. change fault record retrieval priorities to retrieve fault records from DFRs on a circuit where protection operation has been detected.

One such integrated system is that developed for ScottishPower by Bell [19] which integrates APEX, RESPONDD, the consistency based diagnostic engine for protection validation (the forerunner of the protection validation toolkit) and proprietary fault record retrieval and TWFL software. The main goal of this architecture is to automate the retrieval of data to facilitate automated protection validation.

To realise this goal, control modules were integrated along with APEX and RESPONDD to provide the DFR retrieval and TWFL retrieval software with the information necessary to prioritise retrieval of disturbance related data. The required date, time and location of the fault are determined by these modules by using simple algorithms to interpret the diagnoses generated by APEX and RESPONDD. This prioritised data retrieval process is described in [20] and illustrated in Figure 3-7.

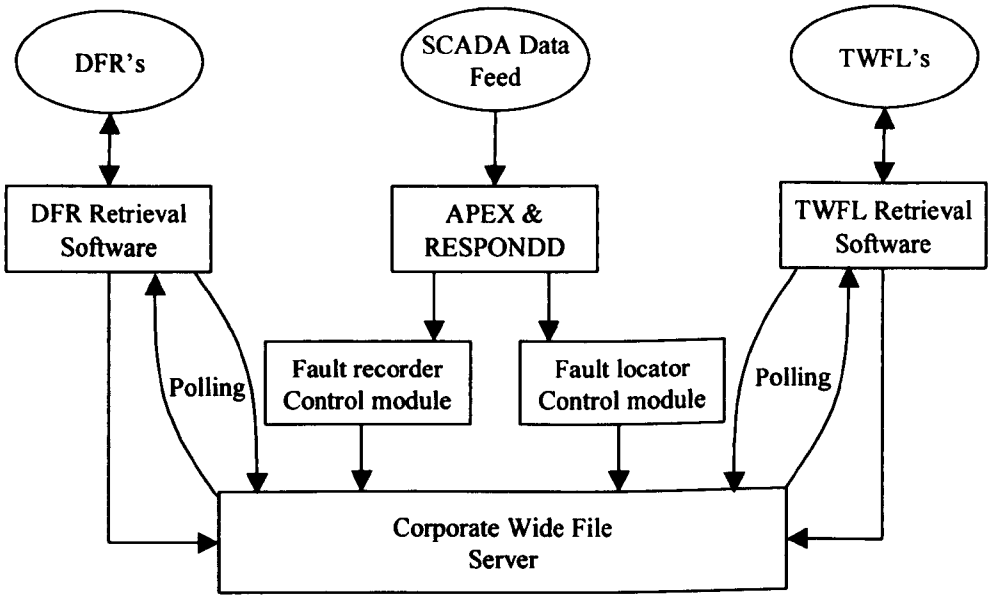


Figure 3-7 Data retrieval process within hybrid system developed by S. Bell

The information generated by the control modules is stored on a corporate wide file server, which both the DFR retrieval and TWFL retrieval software poll to identify if any new power system fault has occurred. On receipt of information from their

respective control modules, the DFR retrieval and TWFL retrieval software dial-up and retrieve fault records and locations produced at the related substations around the time specified.

Although this integrated data retrieval architecture solves the problem of prioritised data retrieval by sub-division of the fault identification, retrieval prioritisation and data retrieval initiation to different software components, it cannot be described as a hybrid intelligent system since both the intelligent systems utilised (APEX and RESPONDD) are KBS. However, the later inclusion of the protection validation system introduced another AI technique, MBD, to complement the knowledge-based systems employed by APEX and RESPONDD, thus enabling the integrated architecture to be classed as hybrid intelligent system.

Although the hybrid intelligent system, developed by Bell is the only one to focus on protection engineering, other hybrid intelligent systems have been developed for fault diagnosis of power systems. Both P.R.S. Jota [21] and R. Rayudu [22] have developed hybrid systems for assisting control engineers with fault diagnosis of power systems, with P.R.S. Jota having also developed a system for diagnosing power transformer faults.

3.4 Required Enhancements to Decision Support

Prior to the introduction of DSS, the protection engineers had to conduct the entire post-fault disturbance analysis process manually, relying on their experience and knowledge of the protection schemes. The introduction of intelligent systems, such as APEX and the protection validation toolkit, have automated many of the manual data retrieval and interpretation tasks releasing some of the protection engineers time. Furthermore, the diagnoses generated can assist the engineer in deciding what disturbances require further investigation, and where to focus protection maintenance.

Although advances in decision support have been made, a number of enhancements are required to optimise the assistance provided and ensure the future provision of decision support. These are as follows:

- Either a new alarm processor focussed on post-fault disturbance analysis must be developed or extensive modifications to APEX are required to bring it in line with the disturbance analysis process followed by protection engineers. At the moment, the reasoning mechanism employed within APEX prohibits it from identifying incidents and grouping incident related SCADA.
- Although prioritised fault record retrieval has been achieved using the hybrid intelligent system developed by Bell, the introduction of new DFRs and fault record formats has led to the system becoming obsolete. A new architecture is needed which can accommodate new devices and provide the automated and prioritised data retrieval which is required.
- The new architecture must allow the easy integration of new DSS and devices without the need for extensive modifications. This would ensure the continued enhancement of decision support through the integration of new data sources and DSS.
- Currently, the protection engineer must have knowledge of how to operate each DSS and proprietary tool. To optimise decision support, automation of these DSS and tools is required together with a common interface where user intervention is minimised.

3.5 Chapter Summary

The research field of intelligent systems has been introduced with the fundamentals being described to a level appropriate to this thesis. Particular attention has been paid to describing the components within knowledge based and model based systems as they will be referred to later in the thesis. The combined use of diverse AI techniques within hybrid systems has also been introduced, providing a good foundation for later discussions on the use of multi-agent systems as a platform for such systems.

The intelligent systems that provide decision support to ScottishPower protection engineers during post-fault disturbance analysis were also introduced. A number of required enhancements to these existing systems have been identified together with the need for a flexible hybrid architecture to ensure the long-term provision of

decision support. The remainder of this thesis will demonstrate an effective means of implementing the required enhancements, beginning with the introduction, in chapter four, of an alarm processor specifically designed for post-fault disturbance analysis.

3.6 References

[1]. G. Shreiber, et al, 'Knowledge Engineering and Management: The CommonKADS Methodology', MIT Press, 1999.

[2]. D.A. Waterman (Editor), "A guide to expert systems", Addison-Wesley Publishing company, 1986.

[3]. J. Kolodner, "Case Based Reasoning", Morgan Kaufman Publishing, 1993.

[4]. J. de Kleer, B.C. Williams, "Diagnosing Multiple Faults", Artificial Intelligence, v32, n 1, pp. 97-130, April 1987.

[5]. R. Davis, W. Hamsher, "Model-Based reasoning: Troubleshooting", Readings in Model Based Diagnosis, Morgan Kaufman Publishers, pp 3-24, 1992.

[6]. A.K. Kordon, "Hybrid Intelligent Systems for Industrial Data Analysis", First International IEEE Symposium on Intelligent Systems, September 2002.

[7]. Khosla. R, Dillon. T, "Engineering Intelligent Hybrid Multi-Agent Systems", Kluwer Academic Publishers, 1997.

[8]. D.S. Kirschen, B.F. Wollenberg, "Intelligent Alarm Processing in Power Systems", Proceedings of the IEEE, v80, n5, May 1992, pp 663-672.

[9]. S.D.J. McArthur, J.R. McDonald, S.C. Bell, G.M. Burt, "An Expert System for On-line Analysis of Power System Protection Performance", in Proc. 1994 Expert Systems conference: Applications and Innovations in Expert Systems, Dec 1994, pp 125-142.

[10]. J.R. McDonald, et al, "Alarm Processing and Fault Diagnosis Using Knowledge Based Systems for Transmission and Distribution Network Control", IEEE Transactions on Power Systems, v7, n3, August 1992, pp 1292-1298.

[11]. G.M. Burt, "An Expert System Approach to on-line fault diagnosis in power system networks", PhD thesis, University of Strathclyde, Department of Electronic and Electrical Engineering, 1992.

[12]. M. Kezunovic, T. Popovic, D.R. Sevcik, A. Chitambar, "Requirements for Automated Fault and Disturbance Data Analysis", CIGRE Colloquium, SC BS-Protection, Australia, September 2003.

[13]. D. R. Sevcik, et.al, "Automated Analysis of Fault Records and Dissemination of Event Reports", Fault and Disturbance Analysis Conference, Atlanta, May 2000.

[14]. M. Kezunovic, P. Spasojevic, C.W. Fromen, D.R. Sevcik, "An Expert System for Transmission Substation Event Analysis", IEEE Transactions on Power Delivery, v8, n4, October 1993, pp 1942-1949.

[15]. A. Dysko, J.R. McDonald, G.M. Burt, J. Goody, B. Gwyn, "Integrated Modeling Environment – A Platform for Dynamic Protection Modeling and Advanced Functionality", IEEE Power Engineering Society Transmission and Distribution Conference, April 1999.

[16]. E. Davidson, S.D.J. McArthur, J.R. McDonald, "A Tool-Set for Applying Model Based Reasoning Techniques to Diagnostics of Power Systems Protection", IEEE Transactions on Power Systems, v18, n2, May 2003.

[17]. S.C. Bell, et al., "Model-based analysis of protection system performance", IEE Proceedings: Generation, Transmission and Distribution, v145, n5, pp 547-552, September 1998.

[18]. M. Chantler, et al, "Use of fault-recorder data for diagnosing timing and other related faults in electricity transmission networks", IEEE Transactions on Power Systems, v15, n4, pp 1388-1393, November 2000.

[19]. S.D.J. McArthur, S. Bell, J.R. McDonald, et al, "The Development of an Advanced Suite of Data Interpretation Facilities for the Analysis of Power System Disturbances, CIGRE 1998, Paris, France.

[20]. S.C. Bell, "Model & Knowledge Based Techniques for the Interpretation of Power System Protection & Operational Data", PhD thesis, University of Strathclyde, Department of Electronic and Electrical Engineering, 1998.

[21]. P.R.S. Jota, et al, "A class of hybrid intelligent system for fault diagnosis in electric power systems", Neurocomputing, v23, 1998, pp 207-224.

[22]. R.K. Rayudu, S. Samarasinghe, A. Maharaj, "A Co-operative Hybrid Algorithm for Fault Diagnosis in Power Transmission", IEEE Power Engineering Society Winter Meeting", 2000.

Chapter 4: Intelligent Alarm Processing for Post-Fault Disturbance Analysis

4.1 Chapter Overview

This chapter describes an intelligent alarm processor for assisting protection engineers with the post-fault disturbance analysis alarm interpretation task.

The chapter commences with a review of automated alarm interpretation and the practical problems existing alarm processors have had to overcome to assist control engineers. The existing deficiencies in protection engineering alarm processing are discussed through presentation of the manual approach to alarm interpretation adopted by protection engineers and their requirements for an online intelligent alarm processor. The nature of the domain and topology knowledge used by protection engineers during manual alarm interpretation is also described.

An online intelligent system entitled the Telemetry Processor specifically developed to improve the alarm interpretation and diagnostic functionality available to protection engineers is then presented. The design choices made, a novel reasoning methodology for protection engineering alarm interpretation and the online implementation of the Telemetry Processor are all discussed.

The chapter concludes with a case study and an evaluation of the Telemetry Processor's performance.

4.2 Automated SCADA Interpretation

Energy Management Systems (EMS), and their associated SCADA systems, have long been used by utilities to alert network control engineers to power system parameters that are out of normal range or to changes that may affect the operation of the power system. A survey of control engineers in the early-1980's [1] highlighted that as the number of alarmed parameters increased into the tens of thousands, they were being faced with excessive volumes of alarms which inhibited their effective management of the network. Part of the problem was that many of the alarms are actually presenting data that is intended more for engineers interested in protection and telecommunications and has less immediate operational value to control engineers.

In an effort to alleviate this problem a feasibility study was conducted in the mid-1980's into the possibility of integrating an intelligent alarm processor into the EMS [2]. The results of this study provided the groundwork for a significant body of research during the 1980's and 90's into the application of AI for alarm processing which eventually resulted in a number of intelligent alarm processors.

The primary intent of alarm processors is to present a clear picture of the power system status during and after disturbances by significantly reducing the amount of data presented and providing information in the form of concise event summaries [3]. To realise this a number of practical problems must be overcome as illustrated in Table 4-1 on the next page.

Existing alarm processors employ a range of AI techniques to try and address these practical problems. In [4] and [5] Artificial Neural Networks (ANN's) are applied and in [6] Logic Based Systems are used, however, by far the most applied technique is Knowledge Based Systems (KBS). There are several reasons for the dominance of KBS over other techniques:

- Using a structured knowledge elicitation process such as CommonKADS [7] the heuristic knowledge experts use to perform alarm processing can be captured and modelled. This knowledge can be encapsulated as rules eliminating the need to develop detailed models or logic formulae of how the system operates.
- The KBS architecture (see Figure 3-1) provides for separation between reasoning and knowledge enabling knowledge base maintenance without interfering with the inference mechanism.
- Unlike ANN's, no training of the inference mechanism is required and the quality of conclusions reached is dependent on the extent of the knowledge base, which can be maintained and updated, as opposed to the size and quality of data set used to train an ANN.

Practical Problems	Impact on Alarm Processing
Alarm rate	The volume of alarms received by the alarm processor is dependent on the levels of power system maintenance activity and number of faults occurring. A typical 24-hour period can see 2511 alarms being generated. This can increase substantially during storms with alarm numbers as high as 51410 being recorded in 24 hours, of which 16683 of these alarms were generated over a 5-hour period. Alarms, therefore, are not guaranteed to arrive at a steady rate and any alarm processor must be able to cope with changing alarm volumes while still providing real-time alarm interpretation.
Missing Alarms	An alarm processor can only reason based on the received alarms. Often this data is incomplete, since alarms can be lost before they reach the control centre due to communications problem. Alarms may also be missing if a device failed to operate and they were never generated. In this scenario, the knowledge that an alarm was never generated may be pertinent. It is, however, impossible to determine, purely from the received alarms, whether an alarm was generated and lost during communications or an alarm was never generated due to a device failure.
Delayed alarms	Alarms may be delayed due to communications problems or the SCADA polling schedule. A temporal reasoning capability is therefore required to determine whether all expected alarms have been received.
Multiple simultaneous events	Alarm processors must be able to perform analysis of several events simultaneously to avoid substantial delays in providing information to the control engineer.
Maintenance	The alarm processor must maintain its network connectivity representation in line with that used within the EMS and by the control engineers. The alarm processor must also maintain its list of possible alarms in line with those recognised by the EMS.

Table 4-1 Practical problems and their impact on alarm processing.

Alarm processors developed using the KBS approach follow a reasoning methodology to control the overall alarm interpretation process. If the reasoning methodology is viewed as a series of core alarm interpretation stages surrounded by additional functions for alarm pre-processing and event output, existing alarm processors can be considered as following a single stage reasoning methodology where a single inference engine performs alarm interpretation. This single inference engine, specifically its reasoning strategy, has been the primary focus of alarm processing research to date.

Expert system shells, such as CLIPS [8] and G2 [9], developed to assist with the development of KBS, provide a generic inference engine and knowledge base but are devoid of any domain knowledge. If used to develop an alarm processing KBS, such expert shells can pose a number of problems. Firstly, whilst extensive configurability may be built in to the shell, an alarm processor may only require a subset of these capabilities. The functionality not utilised will represent an overhead in terms of computing resources (memory, storage capacity, etc.), which may impinge on the real-time performance of an alarm processor. Secondly, there may be a significant performance penalty attributable to the use of generalised reasoning, by comparison with reasoning adapted specifically to alarm processing. Finally, and on a related note, alarm processor developers have reported an inability of the reasoning strategy adopted within a shells inference engine to adequately represent and reason about temporal issues [10]. Due to these perceived disadvantages, alarm processor developers have instead opted for application specific inference engines.

The reasoning strategies employed within the application specific inference engines fall into one of two categories: **Pattern Matching** and **Hypothesise-and-test**.

Pattern matching has been adopted as the reasoning strategy within the inference engines of the SPARSE [11], KBAP [12], and NSP [13] alarm processors. In each the pattern matching strategy compares the received alarms with the alarm sequence expected by each rule. When a match is found the rule fires and the appropriate conclusion for the alarm sequence is generated.

The main problem with pattern matching is that an exact match is required between the received alarms and expected alarm patterns before rules can fire. This means

that the process can fail if all the expected alarms are not received. An alternative strategy is the hypothesise-and-test strategy utilised in APEX [14].

Upon receipt of a new alarm, an inference engine following a hypothesise-and-test strategy consults the knowledge base to identify possible causes for the new alarm. A set of hypotheses is then created, one for each possible cause, specifying the alarms expected by the hypothesis. These hypotheses are added to a list of hypotheses under consideration. Each hypothesis in the list is then evaluated using a scoring system to determine whether it may be considered justified and thus identified as a conclusion. Most commonly, justification will require that a number of supporting alarms have been received. Timing constraints may also be included in the process. For example, it may be specified that all evidence must be accounted for within a fixed time of its observation. Following expiry of this time, then the highest scoring hypothesis is selected as offering the best conclusion. This enables conclusions to be reached even in the event of alarms being lost due to communications failures or in situations where alarms have not been generated due to device failures. However, it must be reiterated that there is no means of distinguishing between alarms missing due to communications failures and those missing because they were never generated.

Many of the event summaries control engineers require relate to changes in network topology caused by the protection initiated opening of circuit breakers following a fault. To process topology, knowledge of the network connectivity is required.

The ideal source of network connectivity is the topology database used within the EMS however many researchers have reported that access to this database has been inhibited by difficulties with integrating the alarm processor into the EMS [10]. An alternative approach adopted within many alarm processors, including KBAP [12] and APEX [14], is to have a separate topology database. However, the maintenance of a topology database separate from the EMS can prove extremely time consuming and relies on personnel with knowledge of the intricacies of both the EMS and the alarms processor and who are capable of amending the topology on a regular basis.

4.3 Post-fault SCADA Interpretation

Control engineers are not the only users of SCADA data. Protection engineers use the same SCADA alarms during the Incident SCADA Interpretation Cycle of post-fault disturbance analysis (as illustrated in Figure 2-9) to identify disturbances and investigate protection operations.

The SCADA interpretation needs of protection engineers have been largely ignored by the research community with alarm processing research focussing predominately on the development of efficient temporal reasoning strategies to enhance real-time alarm processing for control engineers.

A number of researchers have tried to address the protection engineers' needs by providing off-line automated alarm processing facilities based on existing alarm processors [6] or modifying the knowledge base of online systems to identify events of interest [15]. However, the single stage reasoning methodology employed within these alarm processors is still based around solving the practical problems facing control room alarm processing. As a result much of the interpretation functionality required by protection engineers is missing.

The following sections help illustrate the deficiencies in these systems by describing the manual approach to SCADA interpretation adopted by protection engineers and the actual alarm processing requirements of protection engineers. This information was elicited from ScottishPower protection engineers.

4.3.1 Manual SCADA Interpretation

It is not until a protection engineer checks the database of generated SCADA or is contacted by a control engineer that the occurrence of protection operations, and possibly faults, on the network is recognised and this can be several hours to a few days after the protection operations have occurred. Having been informed the protection engineer commences interpretation of the alarms generated over the period that the protection operation(s) occurred. The interpretation procedure has already been introduced in chapter two but is elaborated upon below and illustrated in Figure 4-1.

The procedure begins with retrieval from the SCADA archive of alarms generated around the period of interest. Having retrieved the alarms the process of identifying incidents and events can commence. To reiterate the definition of incidents and events presented in chapter two:

- The **Incident** data set contains all the data pertaining to a particular protection operating sequence. *Note that although a protection operating sequence will most often be initiated by a fault, it may be initiated as the result of a mal-operation.*
- The **Event** data set contains the data related to operation of particular protection scheme components and is a more focussed look at what actually happened during an incident. The primary indicators are SCADA alarms and there will be many events associated with an individual incident.

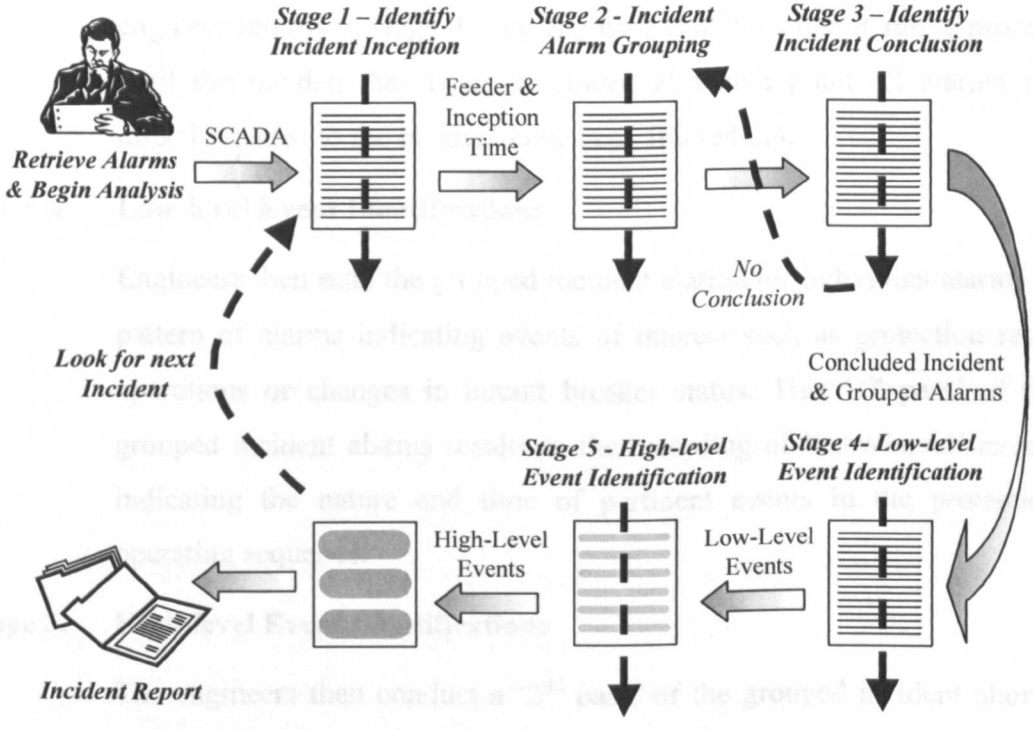


Figure 4-1 Manual alarm interpretation process

The manual incident and event identification process illustrated in Figure 4-1 consists of five stages as described on the next page:

Stage 1. Incident Start Identification:

Engineers scan the retrieved alarms for patterns indicating that a feeder protection scheme has detected a fault and operated. When an alarm pattern is found indicating incident start, the incident start time and incident feeder, from the protection alarm fields, are noted.

Stage 2. Incident Alarm Grouping:

Engineers then begin grouping the incident alarms by selecting the next alarm after incident start that has not already been checked. A topology check of the alarm against the incident circuit is then performed, recording an alarm from the incident circuit as part of the incident.

Stage 3. Incident Conclusion Identification:

The engineers then scan the incident alarms grouped thus far for alarm patterns indicating incident conclusion. If no conclusion is identified the engineer returns to stage 2. The engineer continues this iterative process until the incident has been concluded at which point all alarms not directly related to the incident have been filtered out.

Stage 4. Low-level Event Identification:

Engineers then scan the grouped incident alarms for individual alarms or pattern of alarms indicating events of interest such as protection relay operations or changes in circuit breaker status. This '1st pass' of the grouped incident alarms results in the recording of 'Low-Level' events indicating the nature and time of pertinent events in the protection operating sequence.

Stage 5. High-level Event Identification:

The engineers then conduct a '2nd pass' of the grouped incident alarms taking into account the identified low-level events. At this stage in alarm analysis the protection engineer is looking to identify 'High-Level' events indicating whether the protection scheme operated correctly and any if there could be any anomalies.

On completion of stage 5 an incident report is generated and alarm analysis returns to stage 1, beginning with alarms occurring after the last incident start. The alarm analysis process continues until no more alarms remain to be interpreted.

4.3.2 Protection Engineering Alarm Processing Requirements

The manual approach to alarm interpretation is extremely time-consuming and can, following severe storms, take an experienced protection engineer many days to perform, e.g. for a UK based utility it took one engineer 10 days to process over 15000 alarms. An intelligent alarm processor focussed on the post-fault disturbance analysis alarm interpretation task is therefore necessary in order to reduce the amount of time and effort expended by a protection engineer.

Discussions with protection engineers about their requirements for just such an alarm processor revealed the following:

1. Online interpretation of alarms is essential to avoid delays in engineers retrieving alarms and initiating interpretation using an off-line system.
2. Alarms should be retrieved from the same SCADA archive used during manual disturbance diagnosis, thereby avoiding EMS integration issues.
3. Real-time alarm interpretation is not essential with information provision several minutes after incident conclusion being perfectly acceptable; this is still a significant improvement over the current approach.
4. Provision of identical information to that produced during manual alarm interpretation is required, namely: incident summaries, grouped incident alarms, low-level event summaries and high-level event summaries.
5. Information should be presented in an easily digestible format with access at the user's convenience from locations throughout the company.
6. Low-maintenance of the alarm processing facility is essential to minimise support costs and ensure long-term benefit.

It was evident from the captured requirements that protection engineers are willing to wait for the provision of information with the caveat that the underlying alarm

interpretation process emulates the manual approach, generating identical incident and event information. This is significant, since it suggests that a shift in the focus of alarm processing research is required, from developing efficient temporal reasoning strategies for real-time performance, to developing reasoning architectures more suited to the needs of protection engineers. Therefore, the challenge was to develop a reasoning architecture capable of emulating the multi-stage reasoning adopted by protection engineers during post-fault disturbance analysis. Although real-time performance was not essential, the practical problems of missing alarms, multiple simultaneous events and maintenance still remain.

4.4 Knowledge Capture

Protection engineers use knowledge of the protection engineering domain and network topology to perform alarm interpretation. This knowledge had to be captured before development of the alarm processor could commence.

4.4.1 Domain Knowledge

The domain knowledge used by protection engineers has been learned through years of experience and extensive knowledge of transmission feeder protection schemes. To capture this knowledge the utility's protection engineers were interviewed using the structured knowledge elicitation technique described in chapter three and [7].

Using actual disturbance case studies, the engineers were asked to give a description of the interpretation process for each case study and list the incidents and events generated. For each listed incident and event, the engineers were asked to elaborate on how they came to a particular conclusion based on the available alarms, their general protection knowledge and any peculiarities of the actual protection scheme which operated in response to the disturbance. These structured interviews enabled knowledge transcripts to be produced documenting the domain knowledge used.

Having compiled the knowledge transcripts, further meetings with the protection engineers were conducted to validate the transcribed knowledge and identify any omissions in the elicited knowledge. Further structured interviews were conducted to

elicit knowledge addressing the identified omissions. Following successful validation, rules were created from the transcribed alarm interpretation knowledge, which were again validated by the protection engineers. Validation of the rules was conducted using case studies not previously used during knowledge elicitation.

It should be noted, that the elicited knowledge was generically applicable across all transmission feeders within ScottishPower's network. This is due to every transmission feeder employing a protection scheme based around a common theme: two main protections, a backup protection, intertripping and DAR. Any variation in the protection schemes was captured by selecting case studies where the scheme in question had operated.

The captured domain knowledge can be split into four sub-sets used during different stages of the alarm interpretation process illustrated in Figure 4-2. Each sub-set is summarised below with examples of the elicited knowledge, where appropriate. Note that the alarm grouping stage does not require any detailed domain knowledge, instead only requiring a simple topology check. Although, due to their size, the knowledge transcripts have been omitted from this thesis the elicited knowledge will become clear as the developed rules are demonstrated during performance evaluation in section 4.6 of this chapter.

4.4.1.1 Incident Start Identification

The elicited knowledge revealed that, regardless of the disturbance cause, the first indication from the alarms that a disturbance has occurred and been detected by the protection scheme is the presence of a protection, trip or intertrip relay operated alarm. If within 1 second (in terms of the SCADA clock and not real-time in terms of the alarm processor), this alarm is followed by an alarm indicating the opening of a circuit breaker at the same circuit end as the first alarm then it is almost certain that a protection incident has started. This incident start identification rule can be represented as illustrated on the next page:

RULE: incident_start

IF Alarm with legend indicating protection, trip or intertrip relay operation

AND Alarm indicating an open circuit breaker

AND Both alarms are at the same circuit end.

AND First alarm comes before second alarm

AND Time difference between alarms does not exceed 1000ms

AND First alarm is the earliest protection, trip or intertrip at circuit end

THEN

Note the incident start time and circuit from the first alarm.

It was also noted from the knowledge transcripts that transmission feeder protection schemes have at least two sets of protection both of which are expected to operate in response to a disturbance. The time stamp of the earliest protection, trip or intertrip relay to operate should be taken as the incident start time.

4.4.1.2 Incident Conclusion Identification

As discussed earlier, in section 2.4.3 of chapter two, transmission feeder protection schemes are augmented with DAR equipment, in an attempt to avoid circuits being switched out unnecessarily due to transient faults. The DAR switching sequence is also commonly referred to as autoswitching.

Following a protection operation, a DAR relay will attempt a circuit breaker reclosure. If the disturbance, which caused the initial protection operation, is due to a persistent or permanent fault, then the protection scheme will start another operating sequence to re-open the circuit breaker. The control engineer would traditionally view this as a single incident, however, from the perspective of the protection engineer, it was evident that this should be viewed as two separate incidents. As a result there were only four possible conclusions to an incident which were apparent from the knowledge transcripts:

- *Successful autoswitching sequence:*

  Following the initial protection operation, alarms have been received indicating that the DAR relay has gone through its entire sequence, successfully re-

energising the circuit and resetting its timer ready for the next disturbance. This not only indicates a successful DAR operation but also suggests a transient fault as the root cause of the operation.

- *Successful autoswitching sequence but with circuit breaker trip-on-close:*

  Following the initial protection operation, the DAR relay has gone through its entire sequence, however, on attempting a reclosure, the protection has operated again tripping the circuit breaker and locking out the DAR relay (since it has completed its one allowable reclosure attempt). This suggests a permanent fault as the root cause of the operation. In this case two protection scheme operations have occurred, i.e. two incidents, so the protection engineer would conclude the first incident when the circuit breaker that tripped on closing was closed and the second incident one minute after the last circuit breaker that opened because of the protection scheme operation.

- *Incomplete autoswitching sequence:*

  Following the initial protection operation, no alarms have been received indicating completion of the DAR sequence. Either this suggests a problem with the DAR or that an event has occurred which has inhibited the DAR, e.g. low gas pressure on a circuit breaker. In this case, the protection engineer would conclude the incident after they are satisfied that the protection sequence is over, i.e. one minute after opening of the last circuit breaker.

- *Autoswitching not initiated so default closure after a pre-defined period:*

  Following the initial protection operation, no alarms have been received indicating DAR sequence initiation. Possible reasons for this are that the DAR relay is non-operational, has failed to operate or has been left locked out by the control engineer when the circuit was last re-energised. In such cases, the protection engineer would deem that the incident has concluded one minute after the last circuit breaker that opened because of the protection scheme operation.

4.4.1.3 Low-Level Event Identification

As described in section 4.3.1, once an incident has been identified, incident alarms grouped and the incident concluded, the protection engineer begins stage four of post-fault SCADA interpretation, namely: low-level event identification.

The elicited knowledge indicated that protection engineers interpret the grouped incident alarms to identify alarms relating directly to operation of protection scheme components, thereby filtering out those that are of no interest, e.g. intruder alarms etc. Alarms of interest are main protection operations, intertrips, DAR sequence alarms, circuit breaker open alarms etc.

Furthermore, the elicited knowledge also revealed that the protection engineer often rewords and expand the legends of alarms of interest to generate the protection event summaries. This is because the SCADA system limits the size of any alarm legend string to 25 characters, often resulting in a greatly summarised description of the protection operation to which the alarm refers. For example, the protection engineer would reword the alarm legend "3RD MAIN COMMS CHNL FLTY" into "THIRD MAIN PROTECTION COMMUNICATIONS CHANNEL FAULTY", providing a more amenable summary of the protection operation.

In addition to these low-level protection events, the protection engineer is also looking for event summaries which provide some general information about the incident. The events of interest are normally generated from an alarm pattern and are as follows:

- The number of milliseconds to complete autoswitching.
- Whether all tripped circuit breakers were closed by autoswitching.
- The elapsed time between incident start and conclusion.

4.4.1.4 High-Level Event Identification

It was clear from the knowledge transcript that the high-level events generated during a second pass of the incident alarms and low-level events were of most interest to the protection engineer. It is at this stage that the protection engineer used generic knowledge of the expected protection scheme operation to identify whether

all components operated and if there are any potential anomalies requiring further investigation.

The generic nature of the elicited knowledge was particularly apparent in the captured high-level event knowledge. For example, as already stated, the protection engineer expects at least two main protection operations in response to a disturbance. If only one protection low-level event is identified then a high-level event is generated indicating failure of the other expected protection as illustrated in the rule below:

RULE: 2nd_main_failed_to_operate

IF Low-level event with event summary "1st Main Protection Operated ON"

AND

NOT Low-level event with event summary "2nd Main Protection Operated ON"

THEN

Create a high-level event for "Failed 2nd Main Protection Operation"

Similar rules were also captured for the failure of 1st main protection, no protection operation at a circuit end and no initiation of DAR.

In addition to rules capturing protection failure, additional rules were captured to generate high-level events indicating successful protection operation. Rules were also captured to group high-level events related to successful protection and intertrip operations into a single concise event summary as illustrated on the next page.

RULE: 1st_2nd_intertrips_at_both_ends

IF High-level event with event summary "1st and 2nd Intertrip received at substation x on circuit y"

AND High-level event with event summary "1st and 2nd Intertrip received at substation y on circuit x".

THEN

Create a high-level event for "1st and 2nd intertrips received at both ends"

4.4.2 Topology

As is often the case with experienced protection engineers, the knowledge of network topology can often be recalled from previous experience without referring to the actual network topology. This is fine as long as the network topology has not changed. However, the elicited domain knowledge revealed that the network topology did not change on a regular basis and that the topology information explicit or inferred from, within the SCADA alarms could actually be used thereby limiting the use of the actual topology.

To demonstrate how the protection engineer derives the required topology information from the utility's SCADA alarms, the alarm format is illustrated in Figure 4-2 along with example alarms in Figure 4-3 for two and three ended feeders.

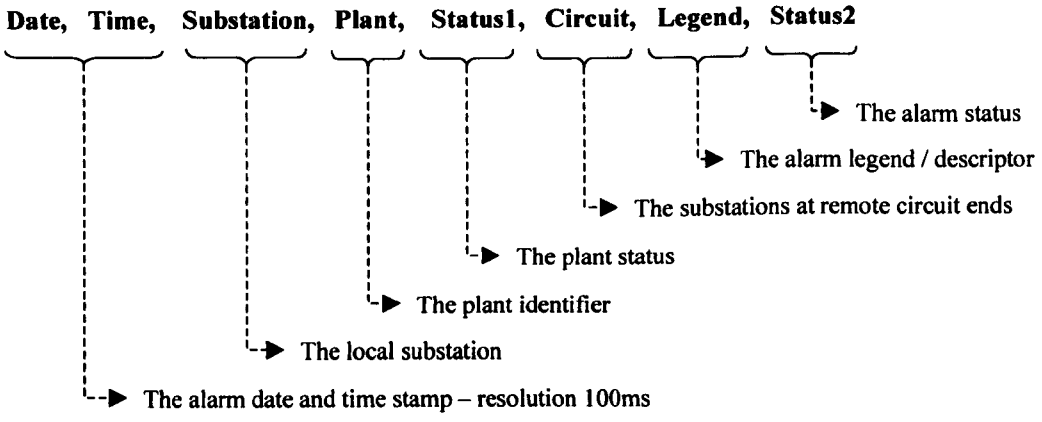


Figure 4-2 Alarm format of ScottishPower PowerSystems SCADA system

Protection alarms for two ended feeder: SUBA / SUBB

06/08/03, 08:22:07.70, SUBA, , , SUBB, SECOND MAIN PROT OPTD, ON
06/08/03, 08:22:07.72, SUBB, , , SUBA, FIRST MAIN PROT OPTD , ON

Protection alarm for three ended feeder: SUBA / SUBC / SUBD

06/08/03, 08:22:07.69, SUBA, , , SUBC / SUBD, SECOND MAIN PROT OPTD, ON

Plant alarms:

06/08/03, 08:22:07.88, SUBA, CB1, OPEN, , ,
06/08/03, 08:22:08.12, SUBB, CB1, OPEN, , ,

Figure 4-3 Example alarms for 2-ended circuits, 3-ended circuits and plant

The substation and circuit alarm fields are the principal source of topology information. As is evident from Figure 4-3, all alarms at least contain an entry in the substation field specifying the substation from which the alarm emanated. In the case of protection alarms, additional topology information can be gained from the circuit field containing the substations at the remote feeder end monitored by the protection.

Given the topology information explicit in protection alarms, it is possible for the protection engineer to identify with certainty the feeder on which a disturbance occurred. However once an incident start has been identified, the protection engineer can find incident alarm grouping (stage two of disturbance alarm interpretation) problematic without accurate topology information, particularly when the alarms relate to large numbers of simultaneous or overlapping incidents.

Alarms containing circuit field entries are not a problem and can easily be compared against the incident circuit information and grouped under the incident. Plant alarms, which do not contain circuit field entries, are more of a problem and the protection engineer must infer topology by considering the plant substation field and the timings of the protection and plant alarms.

When the alarms relate to a unique incident, or there is no overlap with other incidents, then it is assumed, with reasonable certainty that no plant operations will occur during the incident other than those due to protection operations. It is, therefore, possible to group all plant alarms occurring within a pre-defined time frame following a protection alarm under the incident start. However, when the alarms relate to simultaneous or overlapping incidents on feeders from a common substation, it is impossible to identify which feeder the plant alarms from the common substation relate to based purely on the alarm time stamps. It is at this point that the protection engineer turns to a hard-copy of network topology. However, protection engineers have indicated that such scenarios are infrequent and that they only very rarely need to turn to hard copies of network topology. Nevertheless, this problem will be illustrated in the case study in section 4.6 of this chapter.

It should be noted that effective design of a utility's SCADA alarms is critical if alarm time stamps and fields are going to provide enough information to infer topology. When inferring topology, the circuit field plays a key role as it provides a

means of accurately identifying where in the network an alarm emanated from. However SCADA alarm formats vary across the utilities, with the format largely being dictated by the circuit configurations utilised within the power system being monitored. This is apparent when comparing SCADA systems used on transmission networks with those used on distribution networks.

Transmission networks are interconnected and benefit from clearly defined circuits where the substations at both circuit ends are known, making it easy to specify the circuit fields in a SCADA alarm. Distribution networks, on the other hand, are most often radial networks, with a large number of substations on each circuit making it difficult to accurately specify the alarm circuit fields. In such cases, inferring topology from distribution alarms would, therefore, be more difficult.

4.5 Telemetry Processor

Having captured the domain and topology knowledge used by protection engineers during the disturbance alarm interpretation task, development of an intelligent alarm processor to automate the task could proceed. The subsequent research and development program resulted in an alarm processor entitled the Telemetry Processor, which has been implemented within the sponsoring utility as an online aid to protection engineers. The Telemetry Processor interprets alarms using the elicited domain knowledge and provides incident and event information over the corporate Intranet a matter of minutes after a disturbance has concluded.

The following sections outline the design choices made during development, provide a description of the novel alarm processing reasoning architecture developed and a discussion concerning the implementation of the system within the sponsoring utility.

4.5.1 Design Choices

The initial requirements and knowledge capture stages of development for the Telemetry Processor have already been discussed. The next stage was to decide on the most appropriate means for realising the protection engineer's requirements

based on the practical alarm processing problems and elicited knowledge. The design choices made and the rationale behind them is presented below.

Implementation as a Knowledge Based System

It was evident from the elicited domain knowledge that protection engineers do not use models or logic formulae to interpret SCADA alarms, instead they employ rules. This combined with the already extensive use of KBS in alarm processing and the recognised benefits of the technique made implementation of the Telemetry Processor as a KBS the obvious choice.

Multiple inference engines

Early in the design process it was decided that the protection engineer's manual approach to alarm interpretation would be mirrored as far as possible in the reasoning methodology developed for the Telemetry Processor. This would not only ensure that the same incident and event information is generated as in the manual approach but also has the added benefit of transparency, meaning that the protection engineer can fully understand the reasoning process: this provide for greater confidence in the results.

By considering the alarm data-sets used by the protection engineer at each stage of alarm interpretation, it was clear that a distinction could be drawn between the alarms interpreted for incident starts and the grouped incident alarms interpreted for incident conclusion and event identification. A single inference engine could be devoted to interpreting the incoming alarm stream for incident starts with an additional inference engine created for *each* identified incident start to handle incident conclusion and event identification. To clarify the concept, consider a storm scenario where three incidents have occurred on different feeders within less than a minute of each other.

One core inference engine would be operating for the entire execution lifetime of the Telemetry Processor, and would interpret the incoming storm alarms for patterns indicating the beginning of an incident. When an incident start is identified, the core

inference engine would spawn another inference engine dedicated to handling interpretation of alarms grouped as being part of that incident. These additional incident inference engines would be temporary, only remaining in memory and using computer processing power until the incident inference engine has concluded the incident, interpreted its associated grouped alarms for low and high level events and output the conclusions. Therefore, for that one-minute period during the storm, three separate, and temporary, inference engines would exist in addition to the permanent inference engine interpreting the incoming alarm stream.

Consideration of the software problems such an approach may present, suggested that during major network-wide disturbances, with multiple faults occurring close enough in time to have overlapping protection operating sequences, enough separate inference engines would be generated to have an adverse effect on Telemetry Processor performance. However, through evaluation of this risk and discussions with protection engineers, the likelihood of more than five overlapping feeder faults was considered negligible and this, together with the significant memory and processor capabilities of desktop PCs and servers, suggested that the reasoning approach would not pose any performance issues. Nevertheless, the performance of the implemented system was assessed using the worst storm on record to hit ScottishPower and the results will be discussed later in section 6.4.2

Use of expert system shells

The use of expert system shells within alarm processors has been on the whole avoided within control room alarm processors due to their perceived inability to deal with the practical problems of real-time alarm processing. However these problems are not an issue with the Telemetry Processor and the use of expert system shells was seen as an efficient means of accelerating development.

After reviewing a number of expert system shells, the Java Expert System Shell (JESS) [16] was chosen as the most suitable. This was due to its capability for controlling the JESS inference engine through the use of the object-oriented software language JAVA [17]. This was considered a significant advantage, as it would allow the effective control of the multiple inference engines.

Generic rules

Given the requirement for low-maintenance, the method of structuring and representing the elicited domain knowledge within the Telemetry Processor's rulebases required careful consideration. If the number of rules required to represent the domain knowledge could be kept as low as possible, whilst not overcomplicating each individual rule, then the maintenance requirements could be reduced.

Analysis of the knowledge transcripts revealed that the vast majority of domain knowledge was generic and could be applied to any feeder protection scheme within the utility's transmission system. This was an important finding, as it enabled generic rules to be created which used variables in place of specific substation, circuit and plant field entries. This meant that one generic rule could be used to identify a particular incident start, incident conclusion or event on any feeder, rather than having a specific rule for each transmission feeder protection scheme. For example, the generic rule for identifying operation of first main protection on any circuit, can be found in Appendix A.3 – Rule LE_1.

No explicit topology database

The Telemetry Processor will require knowledge of network topology so associations between alarms and circuits can be made. The normal source of network topology would either be through use of the EMS topology database or by creation of topology database separate from the EMS. Neither of these sources is suitable for the Telemetry Processor due to the protection engineers' wish to avoid integration with the EMS and the maintenance overheads associated with a separate topology database. An alternative had to be found.

Given the quality of topology information present in the ScottishPower SCADA alarms, and the fact that no operational decisions will be based on the Telemetry Processor output, it was decided that where topology was not explicit in the alarms, it should be inferred in a similar manner to that followed by protection engineers and described in section 4.4.2. This would eliminate the need for a topology database.

There is, however, a downside to this approach when it comes to plant alarms generated by simultaneous incidents occurring on circuits from a common substation. The problem is that of identifying, without any reference to a topology, which feeder the plant alarms from the common substation relate to based purely on the alarm time stamps. This problem has been mentioned earlier in section 4.4.2 and will be illustrated and explored further during the later case study and performance evaluation.

4.5.2 Reasoning Architecture

A reasoning architecture unique to the Telemetry Processor has been developed to mirror the protection engineers' multi-staged approach to alarm interpretation. The architecture is implemented within the Telemetry Processor as two separate layers as illustrated in Figure 4-4 and summarised below:

- An algorithmic **control layer** written in the JAVA language [17] which controls the overall reasoning process, manages the transition between each stage in the reasoning process and performs incident alarm grouping.
- An **inference layer** containing a number of inference engines implemented using the Java Expert System Shell (JESS) [16], each responsible for different aspects of the alarm interpretation process.

The following sections discuss each stage of the reasoning architecture in detail and how the interaction of both layers mirrors, in an online environment, the multi-staged approach to disturbance alarm identification adopted by protection engineers. Note that the alarm interpretation performance of the architecture within the Telemetry Processor will be evaluated in section 4.6.

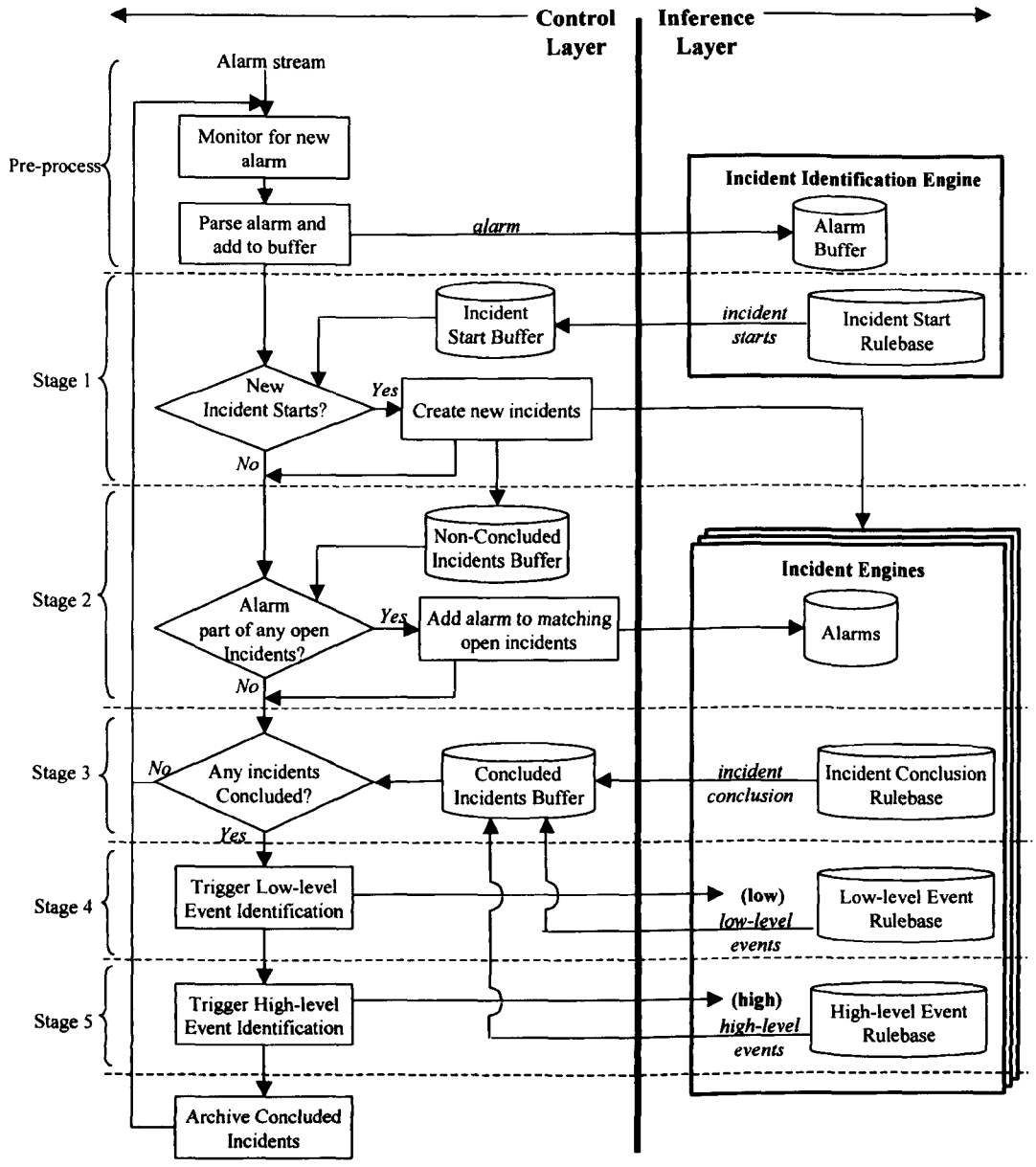


Figure 4-4 Telemetry Processor reasoning architecture

4.5.2.1 Pre-processing

Although not a core interpretation stage, the pre-processing stage is nonetheless essential to the overall reasoning process. It is this stage which is responsible for the monitoring of the SCADA database for retrieved alarms and the parsing of these alarms prior to alarm interpretation – this is handled by the control layer.

Upon finishing alarm parsing, the parsed alarms are passed over to the inference layer where they are added to an alarm buffer within a JESS inference engine

devoted to incident identification. The pre-processing stage maintains the alarm buffer to ensure that the time period spanned by the parsed alarms does not exceed one minute. This is necessary so as to limit the Telemetry Processors' physical memory consumption thereby ensuring continuous online operation. The one minute limit spans a large enough time period to identify incident starts during stage one.

When alarm buffer maintenance has been completed execution control switches to the inference layer and Stage 1 of the reasoning process is initiated.

4.5.2.2 Stage 1 – Incident Start Identification

When parsed alarms are asserted to the incident identification inference engine, the JESS inference algorithm begins trying to pattern match the parsed alarms in the buffer against the Incident Start Rulebase.

The Incident Start Rulebase contains one generic rule derived from the elicited domain knowledge described in section 4.4.1 and illustrated in simplified natural language in Figure 4-3. The JESS representation of this rule can be found in Appendix A.1. When alarm patterns are found in the buffer matching the incident start rule, the inference engine will fire the rule for each matching pattern creating Incident Identifiers which are in turn added to an Incident Start buffer in the control layer. Each Incident Identifier records the incident start time, incident feeder and the earliest protection, trip or intertrip alarm. Only when rule firing has ceased will execution control be handed back to the control layer.

When control layer execution resumes the Incident Start buffer is checked for any new Incident Identifiers. If no new incident starts are indicated, stage two of the reasoning process begins immediately. However, if incident starts have been identified, commencement of stage two is delayed until non-concluded incidents are created to represent the incidents indicated by each incident start.

An individual JESS inference engine is created in the inference layer for each non-concluded incident. It is to these incident inference engines that grouped incident alarms will be added during stage two and interpretation of the incident alarms for incident conclusion will be conducted during stages four to five. Each incident

inference engine is transient and only exists until incident alarm interpretation has concluded.

4.5.2.3 Stage 2 – Incident Alarm Grouping

Every new alarm, which has been received, parsed and added to the Incident Identification Engine is now checked to see if the alarm is part of a non-concluded incident, regardless of whether the alarm has indicated an incident start. There are two criteria which must be met in order for an alarm to be matched to a non-concluded incident:

- The alarm must occur on or after the incident start time.
- The alarm must relate to the incident feeder.

A simple comparison of each parsed alarm's date and time field against the incident start date and time is sufficient to determine if the first criterion has been met. To check the second criterion a topology check must be performed.

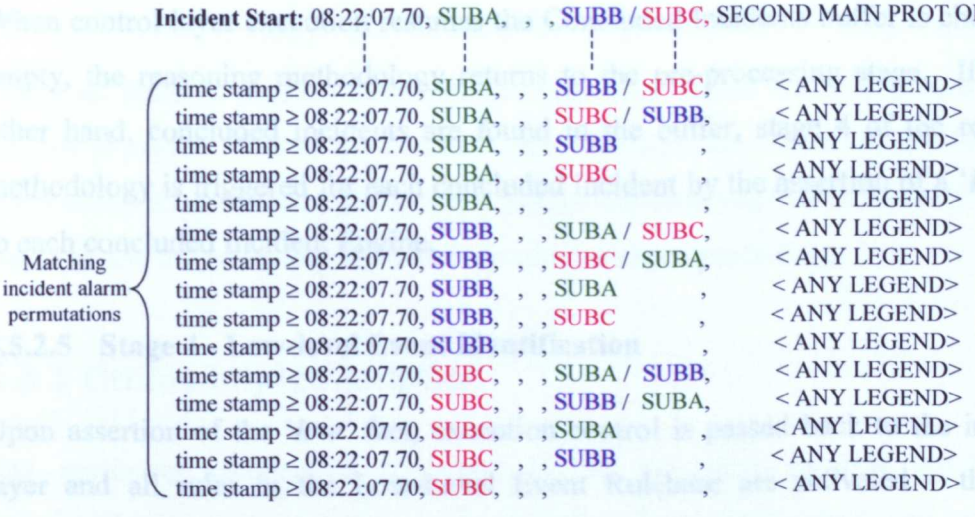


Figure 4-5 Permutations for matching alarms against an incident start

Traditionally it is at this point that the EMS topology or an application specific topology database would be checked to see which feeder the alarm related to. However, the design choice to reduce maintenance overhead by eliminating the topology database means that a topology inference algorithm had to be developed to

determine the likelihood that an alarm relates to a non-concluded incident. This algorithm checks whether an alarm relates to an incident by checking each permutation of the alarm substation and circuit against the incident start feeder. The possible permutations are indicated in Figure 4-5.

When each alarm has been compared against the non-concluded incidents and any matching incident alarms grouped, execution control is handed from the control layer to the inference layer. It is at this point that stage 3 of the process commences.

4.5.2.4 Stage 3 – Incident Conclusion Identification

The assertion of matching alarms triggers each inference engine to begin trying to pattern match its grouped incident alarms against the rules in its Incident Conclusion Rulebase - the JESS representation of the incident conclusion rules can be found in Appendix A.2. When an alarm pattern is found indicating that an incident should be concluded, the appropriate rule will fire and create an incident summary indicating how the incident concluded. The incident summary and grouped incident alarms will then be added to the Concluded Incidents buffer in the control layer.

When control layer execution resumes the Concluded Incidents buffer is checked. If empty, the reasoning methodology returns to the pre-processing stage. If, on the other hand, concluded incidents are found in the buffer, stage 4 of the reasoning methodology is triggered for each concluded incident by the assertion of a ***'low'*** fact to each concluded Incident Engine.

4.5.2.5 Stage 4 - Low-level Event Identification

Upon assertion of the ***'low'*** fact, execution control is passed back to the inference layer and all rules in the Low-Level Event Rulebase are activated – the JESS representation of the low-level event rules can be found in Appendix A.3. Each incident engine begins pattern matching the low-level event rules against the incident alarms.

Any rules which find a matching alarm pattern fire and generate low-level events which are added to the incident in the Concluded Incidents buffer. Each low-level

event contains the event date and time, an event summary and a record of the alarms which indicated the event.

When all incident engines have finished low-level event identification, execution control is passed back to the control layer and stage five of the reasoning methodology is triggered by assertion of a ***'high'*** fact to each concluded Incident Engine.

4.5.2.6 Stage 5 - High-level Event Identification

Upon assertion of the 'high' fact, execution control is passed back to the inference layer and all rules in the High-Level Event Rulebase are activated – the JESS representation of the high-level event facts can be found in Appendix A.4. Each incident engine begins pattern matching the high-level event rules against the incident alarms and low-level events

Any rules which find a matching alarm or low-level event pattern fire and generate high-level events which are added to the incident in the Concluded Incidents buffer. Each high-level event contains the event date and time, an event summary and a record of the alarms and low-level events which indicated the event.

When all incident engines have finished high-level event identification, execution control is passed back to the control layer. The incident, low-level and high-level event summaries and grouped incident alarms for each concluded incident are then archived and the reasoning methodology returns to pre-processing.

4.5.3 Online Implementation

The reasoning architecture described in the preceding sections provides the functionality necessary to interpret the SCADA alarms for incidents and events. To realise this functionality and provide protection engineers with easy and timely access to generated incident and event information an online facility had to be established within ScottishPower PowerSystems. The installation configuration is illustrated in Figure 4-6.

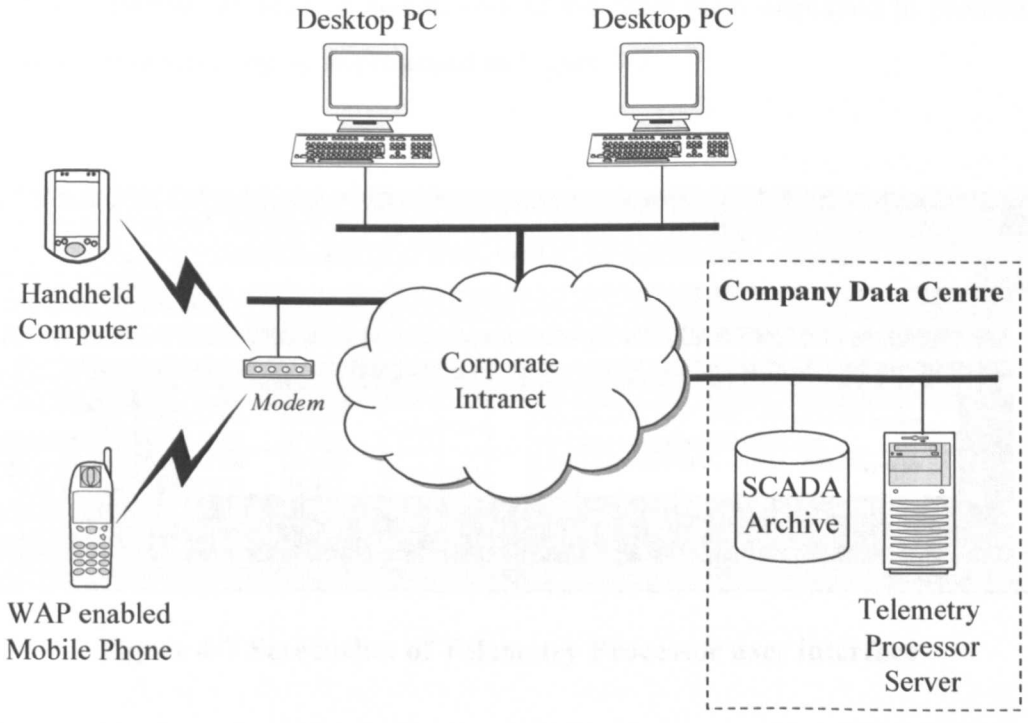


Figure 4-6 Telemetry Processor installation configuration

The Telemetry Processor software resides on a server at the company's data centre and is connected to the SCADA archive via the corporate Intranet. To start the software one of the system administrators responsible for the company's data centre, will enter a number of parameters specifying the network location of the SCADA archive and the interval the Telemetry Processor should wait between each subsequent check of the SCADA archive for new alarms. The interval is normally between one second and one minute and is at the discretion of the system administrator. The decision on what interval is appropriate is often based on how heavily loaded the IT network is and how many other software systems are accessing the SCADA archive.

Once started, the Telemetry Processor will run continuously, without user intervention. At regular intervals, the software queries the archive and retrieves new alarms which are then processed using the reasoning architecture. The generated incident and event information is archived in a database on the server.

To enable protection engineers to access and view the generated information at their convenience a web-based user interface was created. This interface enables engineers from across the company to view incident and event information via web-browsers

over the corporate Intranet. A screen shot of the front page displayed to protection engineers' following log on is presented in Figure 4-7.

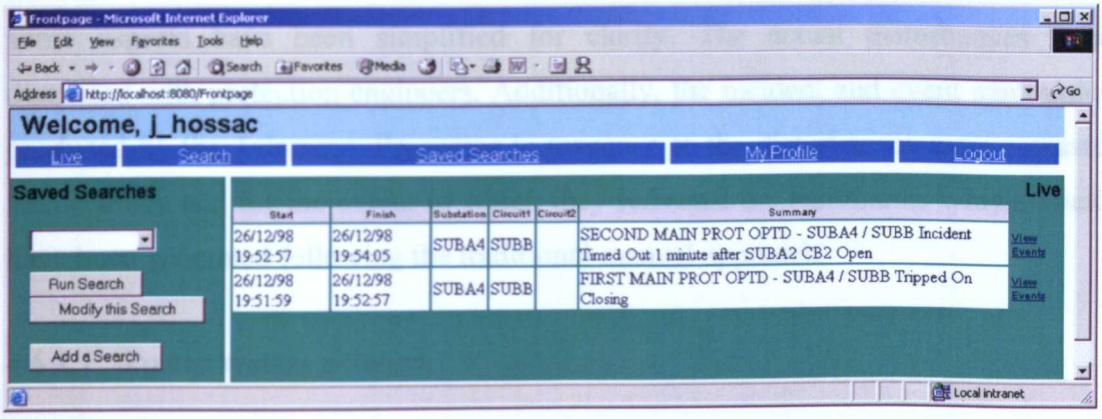


Figure 4-7 Screenshot of Telemetry Processor user interface

Using this interface, the protection engineer can perform a number of functions:

- View the last fifty incidents identified and archived by the system.
- Select a particular incident and view the related events.
- Perform a search for incidents occurring on a specific circuit or on circuits from a particular substation over a given time period.
- Save regular search criteria so the searches can be re-run when desired.
- Create and manage their user profile.

4.6 Telemetry Processor Case Study

Prior to industrial rollout of the Telemetry Processor facility to the sponsoring utility, an extensive off-line testing program was conducted using a variety of case studies based on actual power system disturbances. Of the case studies used during this testing program, one case study in particular provided an excellent illustration of the Telemetry Processor's alarm interpretation capabilities. The remaining sections will present this case study, the Telemetry Processor's output and discuss the reasoning approach. A discussion on the performance of the Telemetry Processor will conclude the section.

4.6.1 Case Study

The case study presented here is closely based on actual disturbances which occurred on the ScottishPower's network during the work described in this thesis, although some aspects have been simplified for clarity. The actual disturbances were discussed with protection engineers. Additionally, the incident and event summaries produced by the Telemetry Processor in response to the disturbances were assessed for accuracy and for the extent to which they reflected the summaries which would have been generated following the traditional manual approach.

4.6.1.1 Power system network

The portion of the transmission network in which the disturbances took place is shown in Figure 4-8. The actual disturbances relate to a double circuit fault caused by double, almost simultaneous, faults close to the busbars at SUBA2.

The protection schemes on each feeder are both augmented with DAR and are illustrated in Figure 4-9 and 4-10 where the following legend is used:

MP1: First Main Protection (Unit – type LFCB)

MP2: Second Main Protection (3 zone Micromho – type SHNB)

INT1: First Intertrip (2 way scheme – Fibre Optics and Microwave)

INT2: Second Intertrip (2 way scheme – British Telecom Pilots)

TR1 & TR2: Trip Relay 1 and Trip Relay 2

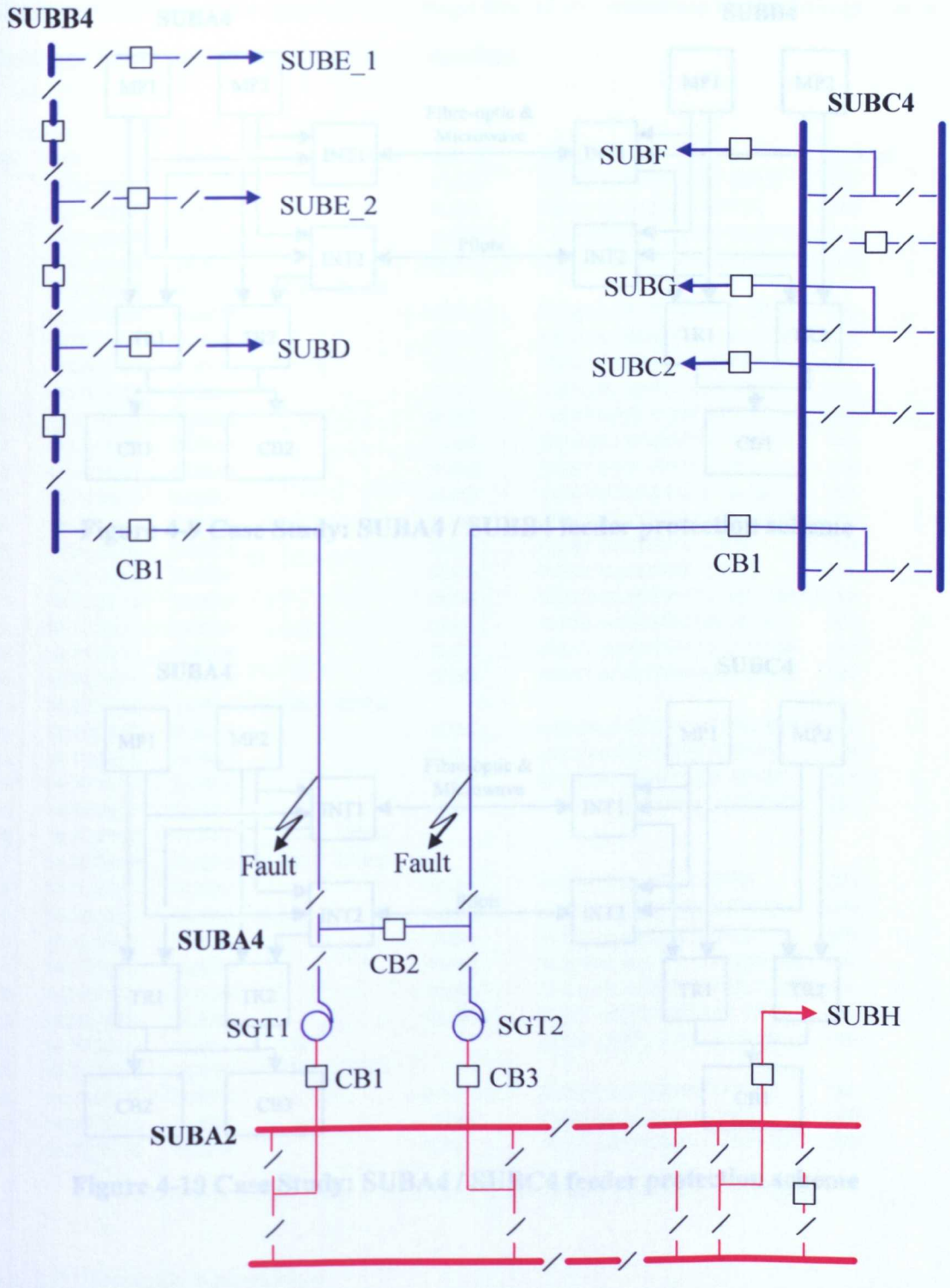


Figure 4-8 Case Study: network diagram

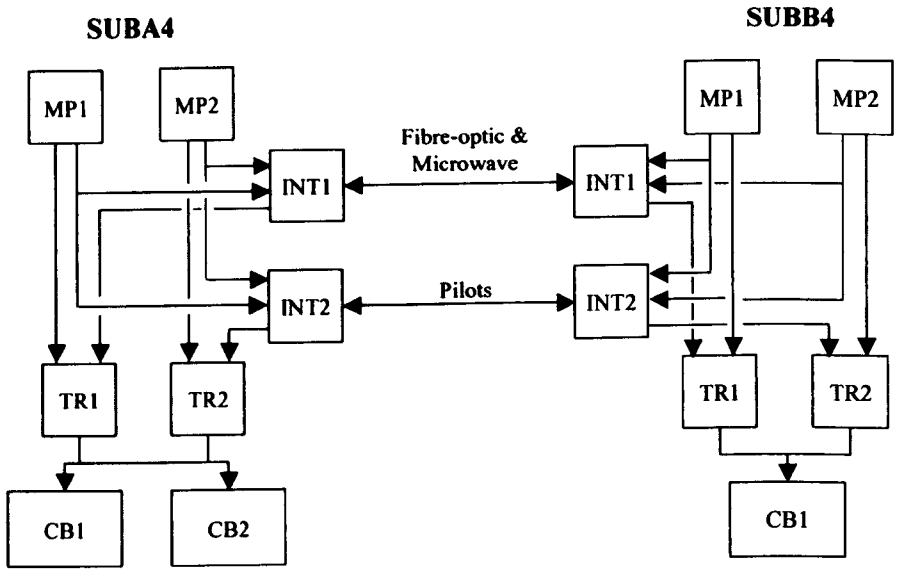


Figure 4-9 Case Study: SUBA4 / SUBB4 feeder protection scheme

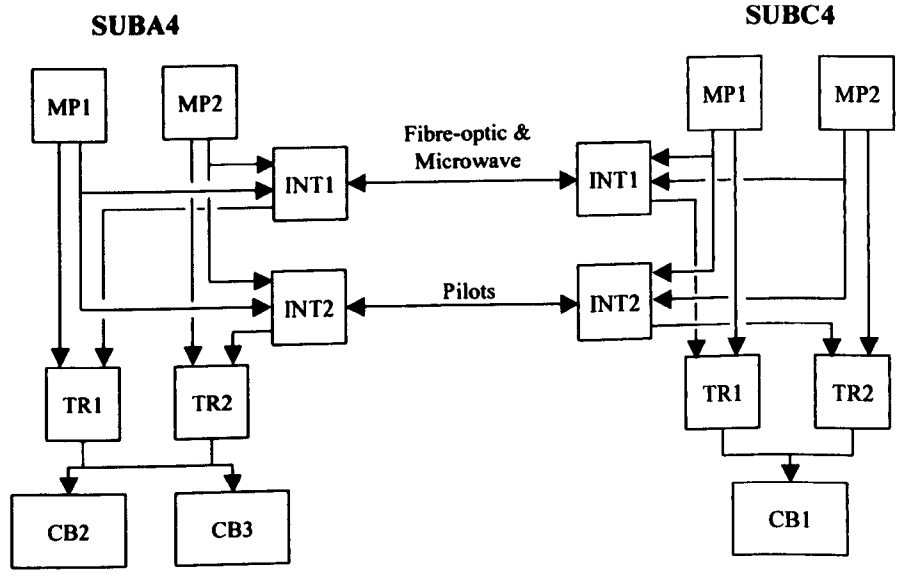


Figure 4-10 Case Study: SUBA4 / SUBC4 feeder protection scheme

4.6.1.2 SCADA Alarms

The case study alarms are presented in Table 4-2. It should be noted that over 110 alarms were received by the Telemetry Processor during the disturbances and only those directly related to the disturbances are presented. In any case, the omitted

alarms would have been ignored after stage two of the reasoning methodology due to them not being part of any disturbance incident.

No.	Time	SubStation	Plant Status1	Circuit	Legend	Status2
1	14:20:38:97	SUBC4		SUBA	SECOND MAIN PROT OPTD	ON
2	14:20:38:98	SUBC4		SUBA	FIRST MAIN PROT OPTD	ON
3	14:20:38:98	SUBC4		SUBA	TRIP RELAYS TO BE RESET-E	ON
4	14:20:39:02	SUBC4		SUBA	SECOND MAIN PROT OPTD	OFF
5	14:20:39:02	SUBC4	CB1 OPEN			
6	14:20:39:03	SUBC4		SUBA	FIRST INTERTRIP REC OPTD	ON
7	14:20:39:05	SUBC4		SUBA	SECOND INTERTRIP REC OPTD	ON
8	14:20:39:05	SUBC4		SUBA	AUTO SWITCHING IN PROG	ON
9	14:20:39:07	SUBA4		SUBC	SECOND MAIN PROT OPTD	ON
10	14:20:39:07	SUBA4		SUBC	FIRST MAIN PROT OPTD	ON
11	14:20:39:07	SUBA4		SUBB	SECOND MAIN PROT OPTD	ON
12	14:20:39.07	SUBA4		SUBB	FIRST MAIN PROT OPTD	ON
13	14:20:39:08	SUBA4		SUBB	TRIP RELAYS TO BE RESET-E	ON
14	14:20:39:08	SUBA4		SUBC	TRIP RELAYS TO BE RESET-E	ON
15	14:20:39:09	SUBC4		SUBA	FIRST MAIN PROT OPTD	OFF
16	14:20:39.09	SUBB4		SUBA	FIRST MAIN PROT OPTD	ON
17	14:20:39.10	SUBA4		SUBB	SECOND INTERTRIP REC OPTD	ON
18	14:20:39:10	SUBA4		CB2	AUTO SWITCHING IN PROG	ON
19	14:20:39:11	SUBA4		SUBC	FIRST INTERTRIP REC OPTD	ON
20	14:20:39:12	SUBA4		SUBB	FIRST INTERTRIP REC OPTD	ON
21	14:20:39:13	SUBA4	CB2 OPEN			
22	14:20:39:13	SUBA4		SUBC	SECOND INTERTRIP REC OPTD	ON
23	14:20:39:15	SUBA4		SUBC	SECOND MAIN PROT OPTD	OFF
24	14:20:39:16	SUBA4		SUBB	SECOND MAIN PROT OPTD	OFF
25	14:20:39:16	SUBA4		CB2	AUTO SWITCHING COMPLETE	ON
26	14:20:39:16	SUBA2	CB1 OPEN			
27	14:20:39:16	SUBA2	CB3 OPEN			
28	14:20:39:17	SUBA4		SUBB	FIRST MAIN PROT OPTD	OFF
29	14:20:39:17	SUBA4		CB2	AUTO SWITCHING IN PROG	OFF
30	14:20:39:18	SUBA4		SUBC	FIRST MAIN PROT OPTD	OFF
31	14:20:39:21	SUBB4		SUBA	TRIP RELAYS TO BE RESET-E	ON
32	14:20:39:21	SUBB4		SUBA	SECOND INTERTRIP REC OPTD	ON
33	14:20:39:21	SUBB4		SUBA	FIRST INTERTRIP REC OPTD	ON
34	14:20:39:23	SUBB4	CB1		AUTO SWITCHING IN PROG	ON
35	14:20:39:24	SUBB4	CB1 OPEN			
36	14:20:39:24	SUBA4		SUBC	AUTOSWITCHING IN PROG	ON
37	14:20:39:25	SUBB4		SUBA	FIRST MAIN PROT OPTD	OFF
38	14:20:39:28	SUBA4		CB2	AUTO SWITCHING COMPLETE	OFF

Table 4-2 Telemetry Processor Case Study: SCADA alarms

4.6.1.3 Domain Knowledge

The JESS rules derived from the elicited domain knowledge and used by the Telemetry Processor to interpret the alarms in Table 4-2 have been included in Appendix A. The rules for incident start, incident conclusion, low-level event and high-level event identification are clearly distinguished.

4.6.1.4 Telemetry Processor Output

The incident and event summaries produced by the Telemetry Processor are presented in Tables 4-3 and 4-4.

Incident			
A	**START**	**14:20:38:97**	**SECOND MAIN PROT OPTD – SUBC4 / SUBA Autoswitching Sequence Complete**
	FINISH	**14:20:39:28**	
Low-Level Events			
1	14:20:38:97	2nd Main Protection Operated ON at SUBC4	
2	14:20:38:98	1st Main Protection Operated ON at SUBC4	
3	14:20:39:02	2nd Main Protection Operated OFF at SUBC4	
4	14:20:39:02	SUBC4 Circuit Breaker CB1 OPEN	
5	14:20:39:03	1st Intertrip Received ON at SUBC4 from SUBA	
6	14:20:39:05	2nd Intertrip Received ON at SUBC4 from SUBA	
7	14:20:39:05	Autoswitching in Progress at SUBC4 SUBA	
8	14:20:39:07	2nd Main Protection Operated ON at SUBA4	
9	14:20:39:07	1st Main Protection Operated ON at SUBA4	
10	14:20:39:09	1st Main Protection Operated OFF at SUBC4	
11	14:20:39:10	Autoswitching in Progress at SUBA4 CB2	
12	14:20:39:11	1st Intertrip Received ON at SUBA4 from SUBC	
13	14:20:39:13	SUBA4 Circuit Breaker CB2 OPEN	
14	14:20:39:13	2nd Intertrip Received ON at SUBA4 from SUBC	
15	14:20:39:15	2nd Main Protection Operated OFF at SUBA4	
16	14:20:39:16	SUBA2 Circuit Breaker CB1 OPEN	
17	14:20:39:16	SUBA2 Circuit Breaker CB3 OPEN	
18	14:20:39:18	1st Main Protection Operated OFF at SUBA4	
19	14:20:39:24	Autoswitching in Progress at SUBA4 SUBC	
20	14:20:39:28	Autoswitching Complete at SUBA4 CB2	
21	14:20:39:28	All tripped circuit breakers did NOT close	
22	14:20:39:28	SUBC4 / SUBA circuit was not restored by end of incident. Time elapsed = 0m 0s 260ms	
23	14:20:39:28	Autoswitching Sequence at SUBA4 CB2 took 0m 0s 180ms	
High-Level Events			
24	14:20:38:98	1st and 2nd Main Protection operated successfully at SUBC4 > SUBA	
25	14:20:39:03	1st and 2nd Intertrips received at both ends	
26	14:20:39:07	1st and 2nd Main Protection operated successfully at SUBA4 > SUBC	

Table 4-3 Telemetry Processor Case Study: Incident A

Incident			
B	START	14:20:39:07	SECOND MAIN PROT OPTD - SUBA4 / SUBB Autoswitching Sequence Complete
	FINISH	14:20:39:38	
Low-Level Events			
1	14:20:39:07	2nd Main Protection Operated ON at SUBA4	
2	14:20:39:07	1st Main Protection Operated ON at SUBA4	
3	14:20:39:09	1st Main Protection Operated ON at SUBB4	
4	14:20:39:10	2nd Intertrip Received ON at SUBA4 from SUBB	
5	14:20:39:10	Autoswitching in Progress at SUBA4 CB2	
6	14:20:39.12	1st Intertrip Received ON at SUBA4 from SUBB	
7	14:20:39:13	SUBA4 Circuit Breaker CB2 OPEN	
8	14:20:39:16	2nd Main Protection Operated OFF at SUBA4	
9	14:20:39:16	SUBA2 Circuit Breaker CB1 OPEN	
10	14:20:39:16	SUBA2 Circuit Breaker CB3 OPEN	
11	14:20:39:17	1st Main Protection Operated OFF at SUBA4	
12	14:20:39:21	2nd Intertrip Received ON at SUBB4 from SUBA	
13	14:20:39:21	1st Intertrip Received ON at SUBB4 from SUBA	
14	14:20:39:23	Autoswitching in Progress at SUBB4 CB1	
15	14:20:39:24	SUBB4 Circuit Breaker CB1 OPEN	
16	14:20:39:25	1st Main Protection Operated OFF at SUBB	
17	14:20:39:28	Autoswitching Complete at SUBA4 CB2	
18	14:20:39:28	All tripped circuit breakers did NOT close	
19	14:20:39:28	SUBA4 / SUBB circuit was not restored by end of incident. Time elapsed = 0m 0s 150ms	
20	14:20:39:28	Autoswitching Sequence at SUBA4 CB2 took 0m 0s 180ms	
High-Level Events			
21	14:20:39:07	1st and 2nd Main Protection operated successfully at SUBA4 > SUBB	
22	14:20:39:09	1st Main Protection operated successfully at SUBB4 > SUBA	
23	14:20:39:12	1st and 2nd Intertrips received at both ends	
24	14:20:39:28	2nd Main Protection at SUBB4 > SUBA failed to operate	

Table 4-4 Telemetry Processor Case Study: Incident B

4.6.1.5 Telemetry Processor reasoning

It is clear that the Telemetry Processor generated two incidents as expected: one for the operation of each protection scheme. The alarm interpretation reasoning conducted to identify these incidents, group each incidents alarms and interpret each set of incident alarms for events is described in detail below.

The second main protection (MP2) at SUBC4 is the first protection to detect a fault on the SUBC4 / SUBA feeder and generate an alarm at 14:20:38.97. Although this is the first indication that an incident has occurred on the feeder, the Incident Identification inference engine has not received all the parsed alarms required to pattern match against the incident start rule IS_1 in Appendix A.1. It is not until alarm 5 is asserted that the inference engine will find a pattern match (alarms 1 and 5) and fire the rule. On rule firing an incident identifier will be created using the protection alarm time stamp as the incident start time and its substation and circuit identifiers as the incident feeder.

Stage one of the reasoning methodology continues with execution control being passed back to the control layer where a new non-concluded incident and an incident inference engine is created for the 14:20:38.97 SUBA4 / SUBC incident start.

During the creation of Incident A, all alarms received by the Telemetry Processor between receipt of alarms 1 and 5 will be matched against the newly created non-concluded incident by the topology inference algorithm. This process will ensure any alarms occurring between the two incident triggering alarms that are related to the incident are identified and grouped as such. At this stage in the reasoning one non-concluded incident exists with the following alarm grouping: Incident A (1-5).

With no additional incident starts or concluded incidents, the control layer continues with pre-processing of received alarms and stages one to three until alarm 21 is asserted to the Incident Identification inference engine. On assertion of this alarm, the inference engine finds an alarm pattern in alarms 11 and 21 indicating an incident start and the incident start rule IS_1 fires generating Incident B. At this stage in the reasoning two non-concluded incidents exist with the following alarm groupings: Incident A (1-10,14,15,18,19) and Incident B (11,21).

During creation of Incident B, and in exactly the same manner as during creation of Incident A, all alarms between receipt of alarms 11 and 21 will then be matched against the newly created non-concluded Incident B by the topology inference algorithm. This process results in the following alarm grouping: Incident B (11-13,16-18,20,21).

The current alarm being processed, alarm 21, then enters stage 2 of reasoning where it is matched against the other non-concluded incidents, in this case Incident A, using the topology inference algorithm. This results in the following alarm groupings: Incident A (1-10,14,15,18,19, 21) and Incident B (11-13,16-18,20,21).

Comparison of the alarm groupings for Incidents A and B indicate that alarm 18 and 21 have been grouped by the topology inference algorithm as being common to both incidents. This is correct given that the topology inference algorithm has no means of identifying from the alarm fields which circuit the CB2 alarm relates to. Given any uncertainty as to the correct incident alarm assignment, the topology inference algorithm places each alarm in both incidents since SUBA is a feeder end common to both incidents.

The reasoning methodology continues with each new alarm being parsed, asserted to the Incident Identification inference engine and then matched against the two non-concluded incidents using the topology inference algorithm.

When the topology inference algorithm has identified an alarm as matching a non-concluded incident, the alarm is added to the inference engine associated with the incident. Upon assertion of this alarm, the incident inference engine will begin searching the grouped incident alarms trying to pattern match sub-sets of the incident alarms against the incident conclusion rules in the Incident Conclusion rulebase. Until an alarm pattern is found indicating incident conclusion and the incident is concluded, stages 1 to 3 continue.

In this case study, DAR relays have initiated autoswitching at each feeder end for both incidents; this is indicated by the 'AUTO SWITCHING IN PROGRESS ON' alarms 8 and 18 for Incident A and alarms 18 and 34 for Incident B. An extract from the elicited domain knowledge documented in the knowledge transcript reveals that for a successful autoswitching sequence:

- *"...The incident is concluded when an alarm is received indicating completion of the autoswitching sequence."*

This knowledge is represented in incident conclusion rule IC_1 in Appendix A.2 which is present in the Incident Conclusion rulebases of both incidents A and B.

Using rule IC_1, both incident inference engines find a pattern match when alarm 38, '14:20:39.28 SUBA4 CB2 AUTO SWITCHING COMPLETE OFF', is asserted to each incident inference engine. In each case, the rule fires and generates an incident summary indicating the first alarm and the manner of incident conclusion:

- 14:20:38.91 SECOND MAIN PROT OPTD – SUBC4 / SUBA
  Autoswitching Sequence Complete
- 14:20:39.07 SECOND MAIN PROT OPTD – SUBA4 / SUBB
  Autoswitching Sequence Complete

Each incident is then concluded, stored in a concluded incidents buffer and removed from the non-concluded incident buffer. The control layer now recognises the conclusion of both incidents and moves to stage 4 of alarm analysis: Low-Level Event Identification.

Low-level event identification is triggered for both incidents by the assertion of a 'low' fact to each inference engine. This activates the low-level event rules which begin pattern matching each set of grouped incident alarms against rules LE_1 to LE_15 in Appendix A.3. As rules fire, the appropriate low-level event summaries are generated and added to the concluded incident.

The simplest forms of low-level events are those generated from one alarm and are basically rewordings of the alarm legends. Event 2 of Incident A and events 2 and 3 of Incident B are good examples where, using rule LE_1 in Appendix A.3, each incident inference engines has identified the 'FIRST MAIN PROT OPTD ON' alarms and has generated an event summary for each of the form "First Main Protection Operated ON at SUBC4". Note that the same rule has been used to generate all three events despite the alarms emanating from different circuit ends. This is possible due to the rule using generic wildcards for the substation and circuit alarm fields.

Low-level events are also generated from patterns of alarms and provide summarised information on what happened during the incident. For example, both incident inference engines used rule LE-13 in Appendix A.3 to determine how long the autoswitching sequence took and generate event summaries. In both cases the elapsed time between the 'AUTO SWITCHING IN PROG ON' and 'AUTO SWITCHING COMPLETE OFF' alarms was calculated and event 23 of Incident A and 20 of Incident B generated indicating completion of the autoswitching sequence in 180ms.

Although simple in nature, these low-level events have already provided useful information since the absence of a "First Main Protection Operated ON at SUBA4" event in Incident A may indicate to the protection engineer problems with the protection scheme on the SUBA4 / SUBC circuit.

When each incident inference engine has finished firing low-level events, execution is passed back to the control layer and stage five of the reasoning process begins: High-Level Event Identification.

High-level event identification is triggered for both incidents by the assertion of a 'high' fact to each inference engine. This activates the high-level event rules which begin pattern matching each set of grouped incident alarms and low-level events against rules HE_1 to HE_7 in Appendix A.4. As rules fire, the appropriate high-level event summaries are generated and added to the concluded incident. One such high-level event is event 24 of Incident B: "2nd Main Protection at SUBB4 > SUBA failed to operate". This event was generated when rule HE_4 triggered on assertion of the 'high' fact due to there being a "1st Main Protection Operated ON at SUBB4" low-level event and there not being a corresponding "2nd Main Protection Operated ON at SUBB4" low-level event.

Having completed stage five for both concluded incidents, the reasoning methodology archives the concluded incidents and events, which are then available for viewing by the protection engineer using the web-based interface. For a summary of the complete output, refer back to previous tables 4-3 and 4-4.

4.6.2 Performance Evaluation

At the time of writing this thesis, the Telemetry Processor facility at the sponsoring utility has been operating in an online mode for a number of months. A number of incidents have occurred over this period and the alarms obtained from the SCADA archive have been interpreted and incidents identified successfully.

Before the Telemetry Processor could be installed at the sponsoring utility a rigorous testing program was conducted where the systems performance was tested using alarm data from the most adverse storms and disturbances to affect the sponsoring utility's network in recent years.

To ascertain the reasoning efficiency, speed and diagnostic capabilities of the new technique 15,500 alarms generated during an actual storm were input offline. On a 2.2. GHz Pentium processor with 256MB RAM the Telemetry Processor took 9 minutes and 50 seconds to parse and interpret all alarms successfully identifying 110 incidents. This is a significant time saving when compared with the ten man-days spent performing manual analysis of the same data.

The online performance was evaluated by simulating the real-time feed of alarms. It was found that the incident reports were available on the corporate Intranet and accessible via the web user interface within a minute of the incidents concluding. This rapid alarm interpretation and provision of incident and event information, enables protection engineers to monitor, in near-real time, the performance of protection schemes during storms from anywhere in the company. Any protection problems which have been indicated by high-level events and which may inhibit continued network operations can be identified and a control engineer informed.

At the requirements capture stage of development, the protection engineers had requested that the Telemetry Processor be designed in such a way as to limit its maintenance requirements. Consequently, given the quality of topology information available in the utility's SCADA alarms, the design choice was taken to avoid an explicit representation of network topology. As an alternative, a topology inference algorithm was developed to infer topology from SCADA alarms in a similar manner to that adopted by protection engineers.

The topology inference algorithm has proved extremely efficient and the need for maintenance of a topology database has been eliminated entirely. Only simultaneous incidents occurring on circuits from the same substation, as in the case study, pose a problem due to the algorithm not being able to distinguish between plant at the same circuit end but on different feeders. Historical analysis of disturbance data revealed that this only occurred in 4% of incidents and resulted in incorrect assignment of additional plant alarms to simultaneous incidents. This level of accuracy was more than acceptable to the protection engineers and the elimination of any topology maintenance overhead was welcomed.

The protection engineers, in conjunction with the author, have been exploring the possibility of achieving 100% accuracy by enhancing the quality of topology information in the SCADA alarms. The sponsoring utility is initiating an EMS replacement program and it was quickly recognised that this would be the ideal opportunity to improve the SCADA system by adding feeder information to alarms containing only a substation identifier. If such additional topology information could be included, the Telemetry Processor would achieve 100% accuracy with no topology maintenance overhead.

The extensive testing and knowledge validation program confirmed that all the protection engineers' requirements had been met. The new reasoning technique can identify incidents, group incident alarms and identify the low- and high-level events of interest to protection engineers. Furthermore, the use of multiple inference engines combined with a separate controlling mechanism enables the protection engineers' offline approach to be mirrored in an online environment.

4.7 Chapter Summary

This chapter has described an intelligent alarm processor, the Telemetry Processor for the online protection engineering interpretation of SCADA alarms. The protection engineers requirements for an alarm processor have been presented and the suitability of existing alarm processors for meeting these requirements discussed.

The new reasoning methodology developed to mirror the protection engineers approach to alarm interpretation in an online alarm processor is presented and demonstrated using a case study. A particular feature of the Telemetry Processor is the topology inference algorithm used to infer topology from the utility's SCADA alarms thereby eliminating the need for a topology database and the associated maintenance overheads.

The next chapter introduces Multi-Agent Systems (MAS), a technology which will later be demonstrated as an effective means of integrating the online Telemetry Processor with other power system data interpretation systems to assist protection engineers with the entire post-fault disturbance analysis process.

4.8 References

[1]. W.R. Prince, B.F. Wollenberg, D.B. Bertagnolli, "Survey on Excessive Alarms", IEEE Transactions on Power Systems, v4, n3, August 1989, pp 950-956.

[2]. B.F. Wollenberg, "Feasibility Study for an Energy Management System Intelligent Alarm Processor", IEEE Transactions on Power Systems, v.PWRS-1, n2, May 1986, pp 241-247.

[3]. D.S. Kirshen, B.F. Wollenberg, "Intelligent Alarm Processing in Power Systems", *Proceedings of the IEEE*, v80, n5, May 1992, pp 663-672.

[4]. R. Khosla, T. Dillon, "Learning Knowledge and Strategy of a Neuro-Expert System Architecture in Alarm Processing", *IEEE Transactions on Power Systems*, v12, n4, November 1997, pp 1610-1618.

[5]. M.A.P. Rodrigues, J.C.S. Souza, M.Th. Shilling, "Building Local Neural Classifiers for Alarm Handling and Fault Location in Electrical Power Systems", in Proc. 1999 Intelligent Systems Applications in Power (ISAP), 1999.

[6]. J. Jung, C-C. Liu, M. Hong, M. Gallanti, G. Tornielli, "Multiple Hypotheses and Their Credibility in On-Line Fault Diagnosis", *IEEE Transactions on Power Delivery*, v16, n2, April 2001.

[7]. G. Shreiber, et al, 'Knowledge Engineering and Management: The CommonKADS Methodology', MIT Press, 1999.

[8]. CLIPS Expert System Shell, http://www.ghg.net/clips/CLIPS.html

[9]. G2 from Gensym, http://www.gensym.com/manufacturing/g2_overview.html

[10]. Z.A. Vale, C. Ramos, L. Faria, J. Santos, M. Fernandes, C. Rosado, A. Marques, "Knowledge-Based Systems for Power System Control Centers: Is Knowledge The Problem", in Proc. 1997 Intelligent Systems Applications in Power (ISAP), July 6-10 1997.

[11]. Z.A. Vale, A. Machado e Moura, "An Expert System with Temporal Reasoning for Alarm Processing in Power System Control Centers", IEEE Transactions on Power Systems, v8, n3, August 1993, pp 1307-1314.

[12]. D.B. Tesch, D.C. Yu, L-M. Fu, K. Vairavan, "A Knowledge-Based Alarm Processor for and Energy Management System", *IEEE Transactions on Power Systems*, v5, n1, February 1990, pp 268-275.

[13]. R.W. Bijoch, S.H. Harris, T.L. Volkmann, J.J. Bann, B.F. Wollenberg, "Development and implementation of the NSP intelligent alarm processor", IEEE Transactions on Power Systems, v6, n2, May 1991, pp 806-812.

[14]. J.R. McDonald, G.M. Burt, D.J. Young, "Alarm Processing and fault diagnosis using knowledge based systems for transmission and distribution network control", *IEEE Transactions on Power Systems*, v7, n3, pp 1292-1298, August 1992.

[15]. S.D.J. McArthur, J.R. McDonald, S.C. Bell, G.M. Burt, "An Expert System for On-line Analysis of Power System Protection Performance", *in Proc. 1994 Expert Systems conference: Applications and Innovations in Expert Systems*, 1994, pp 125-142.

[16]. E.J. Friedman-Hill, "Jess, The Java Expert System Shell", http://herzberg.ca.sandia.gov/jess/, version 6.1a5, 15th January 2003.

[17]. "Java Sun", http://java.sun.com

Chapter 5: Multi-Agent Systems

5.1 Chapter Overview

Intelligent agents are a popular research area in both Artificial Intelligence and Computer Science research fields. Although the term 'agents' has been very popular and used within a number of different definitions, the aim of this chapter is to present the most important issues of agent-based technology associated with the research described in this thesis. The main features of these agents and the multi-agent communities in which they commonly reside are introduced based on the application of multi agent systems in power engineering.

5.2 Intelligent Agents

The term 'agents' can be used to describe very different kinds of systems (biological systems, technological artefacts etc) as shown in Figure 5-1. Within the research described in this thesis, the term 'agent' will correspond to the category of task specific software components. Again within this thesis, although all agents will be categorised as task specific software components, only those which are capable of processing different types of inputs utilising different AI techniques and communicating their results in an attempt to intelligently interpret the data of a particular problem domain will be considered as 'intelligent agents'.

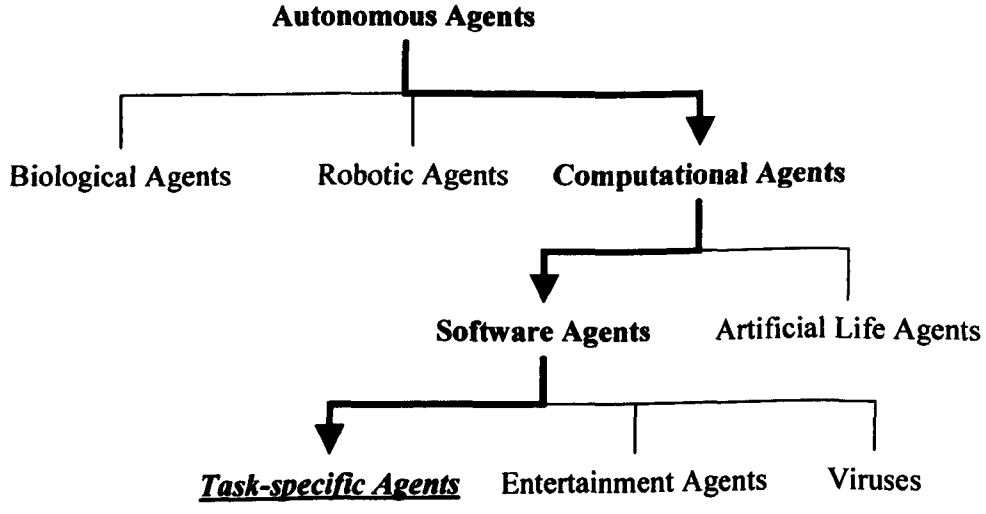


Figure 5-1 Taxonomy of Agents [1]

To be considered a software agent, Wooldridge and Jennings [2] deem that a software component should exhibit the following properties:

- *Autonomy*: an agent operates without the direct intervention of other agents or humans and has control over its actions and its internal state.
- *Responsiveness*: an agent perceives its environment and responds in a timely fashion to changes that occur in it.
- *Pro-activeness*: an agent doesn't simply react to changes in the environment, but exhibits goal-directed behaviour and takes the initiative when it considers it appropriate.
- *Social ability*: an agent interacts with other agents (if it is needed) to complete its tasks and help others to achieve their goals.

The above characteristics form the weak notion of agency while the strong notion of agency is described by properties that are more usually applied to humans (like the strong notion of AI that assigns to an intelligent action the same scope of action seen in humans):

- *Mobility*: the agents can move around an electronic network. This means that not only are robots characterised as mobile, but also an agent that is 'moving' through the Internet can be classed as mobile.
- *Veracity*: an agent will not knowingly communicate false information.
- *Benevolence*: the agents do not have conflicting goals, and therefore they will try to do what they are asked to.
- *Rationality*: The agents will not act in such a way as to prevent their goals being achieved.
- *Co-operation*: The users specify what they want to be performed on their behalf by the agent, and the agent specifies what it can do and provides results.

The software development of agent-based systems views these autonomous software agents as components of a much larger business function. The main benefit of viewing them from this perspective is that the partial software components can be integrated into a coherent and consistent software system in which they work

together to better meet the needs of the entire application (utilising autonomy, responsiveness, pro-activeness and social ability). Based on this integrated environment, this thesis describes how the functional complexity of post-fault disturbance analysis can be overcome within such architectures.

5.3 Multi-Agent Systems

Artificial intelligence research originally focussed on complicated, centralised intelligent systems with expertise in certain domains. This changed during the mid-1970s, when researchers investigating Distributed Artificial Intelligence (DAI) began to formulate some of the basic theories, architectures, and experiments that showed how interaction and sub-division of tasks could be effectively applied to problem solving [3]. Experiments showed that intelligent, rational behaviour is not an attribute of isolated components, but rather an outcome that emerges from the interaction of entities with simpler behaviours [4]. Out of this research, architectures consisting of distributed autonomous reasoning components, Multi-Agent Systems (MAS), began to emerge.

Various definitions from different disciplines have been proposed for the term 'multi-agent system'. As seen from a DAI perspective, a multi-agent system is a loosely coupled network of problem-solver entities that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity [5]. More recently, the term multi-agent system has been given a more general meaning, and it is now used for all types of systems composed of multiple autonomous components showing the following characteristics [6]:

- Each agent has incomplete capabilities to solve a problem
- There is no global system control
- Data is decentralized
- Computation is asynchronous

One of the current factors fostering MAS development is the increasing popularity of the Internet, which provides the basis for an open environment where agents interact with each other to reach their individual or shared goals. To interact in such an

environment, agents need to overcome two problems: they must be able to find each other (since agents might appear, disappear, or move at any time); and they must be able to interact [6]. As will be demonstrated later in the thesis, such an open environment is also required for decision support within the power industry.

The following sections describe the types of industrial applications suited to MAS, the methods for facilitating information discovery between agents and the components of a MAS architecture essential for interaction.

5.3.1 Applications Suited to Multi-Agent Technology

Agents are not the panacea for all industrial problems, and, like any other technology, have certain capabilities that are best used for problems whose characteristics require those capabilities. Research conducted by H. Van Dyke [7] extends the categories specified in [8] to five characteristics that indicate agents are best suited for applications that are:

- **Modular:** Agents are pro-active software components and as such are suited to applications that fall into natural modules.
- **Decentralised:** An agent autonomously monitors its own environment and takes action, as it deems appropriate. This characteristic of agents makes them particularly suited for applications that can be decomposed into stand-alone processes, each capable of performing useful tasks without continuous direction from other processes.
- **Changeable:** Agents are well suited to modular problems because they are independent software components. They are well suited to decentralised problems because they are autonomous and pro-active. These two characteristics combine to make them especially valuable when a problem is likely to change frequently, as illustrated in Figure 5-2. Modularity permits the system to be modified one piece at a time. Decentralisation minimises the impact that changing one module has on the behaviour of other modules.

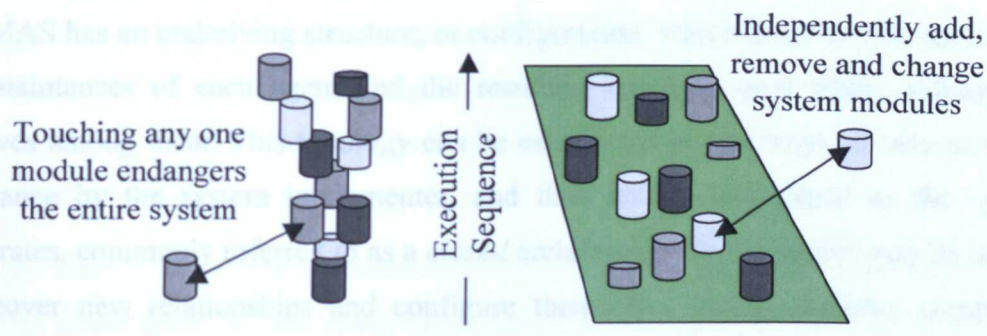


Figure 5-2 Modularity + Decentralisation → Changeability [3]

- **Ill-structured:** Agents are designed to interact with their environment rather than with other specific agents, allowing interactions with any other agent that modifies the environment. With agent-based design there is, therefore, no need to specify the individual components to be interconnected and their interfaces with one another, instead it is sufficient to merely identify the classes of the components in the system and their impact on the environment. Agents are, therefore, particularly suited to ill-structured applications where it is extremely difficult or near on impossible to determine the structure of the application in advance of design.
- **Complex:** One measure of system complexity is the number of different behaviours that must be exhibited. As is often the case, increasing the number of interacting components in a system results in a combinatorial increase in the number of different interactions between components. In traditional systems, these behaviours must be implemented at design time. Appropriately designed agent architectures can move the generation of combinatorial behaviours from design-time to run-time, drastically reducing the amount of software that must be generated and thus the cost of the system to be constructed.

5.3.2 MAS Configuration

A MAS has an underlying structure, or configuration, which describes the immediate acquaintances of each agent and the resulting topology over which information moves among them. This topology can be established in two ways. It may be set in advance by the system implementer, and thus remain unchanged as the system operates, commonly referred to as a *closed* architecture. Or the agents may be able to discover new relationships and configure themselves during runtime, commonly referred to as an *open* architecture.

Closed architectures, although robust, do not welcome changeability, since any change in the available agents, their location or abilities needs to be reflected in the acquaintance information hard coded within the agents. This problem is overcome in open architectures by providing information discovery services.

In open architectures, *utility agents*, also referred to as *middle agents* [9], provide information discovery services. These agents provide a mechanism for advertising, discovering, using, managing and updating agent services and information. Utility agents are entities to which other agents advertise their capabilities, and which are neither requesters nor providers from the standpoint of the transaction under consideration. The advantage of utility agents is that they allow a MAS to operate robustly when confronted with agent appearance and disappearance.

There are several types of agents that fall under the definition of utility agents:

- **Nameservers:** Also referred to as 'white pages', these agents provide a look-up service for agents' network addresses [10].
- **Facilitators:** Also referred to as 'yellow pages', these agents provide a look-up service for agents' abilities [10].
- **Mediators**: agents that exploit encoded knowledge to create services for a higher level of applications [11].
- **Brokers**: agents that receive requests and perform actions using services from other agents in conjunction with their own resources [12].
- **Blackboards**: repository agents that receive and hold requests for other agents to process [13].

5.3.3 Coordination

Agents are characterised by their autonomy and their ability to execute without being invoked. Given this autonomy, agents need to coordinate to ensure robust global behaviour. This coordination is sometimes refined into more specific categories of cooperation and negotiation.

Cooperation is, in general, coordination amongst non-antagonistic agents and relies on the decomposition and distribution of tasks amongst agents. There are two popular mechanisms for achieving cooperation:

- *Co-operative interaction*: This occurs when agents interact to assist each other in achieving their goals more efficiently. The co-ordination has to be built from the developer of the software, in terms of goals, roles and relationships between them.
- *Contract-based co-operation*: This approach uses one of the common auction strategies, when there is some conflict between the agents [14]:
    - *Sealed-bid auction*: each agent submits a bid without knowing the bids of the other agents. The contract is awarded to the cheapest bidder.
    - *English auction*: bids are accepted sequentially. Each new bid must be cheaper than the currently cheapest bid. The contract is awarded to the final bidder (who offered the cheapest bid).
    - *Dutch auction*: The initiator invites potential contractors to bid at a given price, which is systematically increased until a bid is received. The contract is awarded to the first bidder.

  The approach most commonly used within MAS is the contract-net protocol [15], which is based on a sealed-bid auction. The agent co-operates by committing to a goal, which makes it able to predict the actions of other agents contracted to it.

In contrast to cooperation, negotiation is coordination amongst competitive or self-interested agents. Conflicts can often arise if agents are competing for a share of a common finite resource, which they require to carry out their goals. These conflicts can be resolved with negotiation between the agents or the development of a

software management mechanism. Using the latter means behaviour rules have to be defined, while negotiation can contribute to the system's equilibrium in a dynamic fashion [16].

5.3.4 Communication

In order for agents to achieve their goals, and to facilitate coordination amongst agents, communication between the agents is a necessity. Although agent communications are achieved using the communications common to IT networks, namely TCP/IP, SMTP and HTTP, there are a number of additional considerations:

- A common Agent Communication Language (ACL) is required so any agent that receives a message can understand its intent and process it accordingly.
- A common vocabulary, or ontology, is needed so each agent can understand the information contained in each message.
- A common message content language, or syntax, is essential if messages are to be parsed correctly and understood.

5.3.4.1 Agent Communication Language (ACL)

One of the earliest and best known ACL is the Knowledge Query and Manipulation Language (KQML) [17], which was developed in the early 1990s as part of the US government's DARPA knowledge sharing effort [18]. In recent years the most active participants in agent research have been supporting the ACL developed by the Foundation for Intelligent Physical Agents (FIPA), an international non-profit organisation which aims to set general standards for agent interoperability. The research presented in this thesis also supports the FIPA ACL.

The FIPA ACL incorporates many aspects of KQML and is based on the idea that communication can best be modelled as the exchange of declarative statements. Under this paradigm, agents send, receive and reply to requests for services and information, with the intent of the message specified by a performative, such as 'inform' or 'request', describing the way in which the message content expression

should be expressed [19]. Examples of FIPA-SL performatives are presented in Table 5-1.

FIPA Performative	Summary
query-ref	The action of asking another agent for the object referred to by a referential expression.
request	The sender requests the receiver to perform some action.
inform	The sender informs the receiver that a given proposition is true.
subscribe	The act of requesting a persistent intention to notify the sender of the value of a reference, and to notify again whenever the object identified by the reference changes.
failure	The action of telling another agent that an action was attempted but the attempt failed.

Table 5-1 Example FIPA performatives [20]

FIPA SL Paramater	Meaning
:sender	Defines the agent name of the sender of the performative.
:receiver	Defines the agent name of the received of the performative.
:in-reply-to	Describes the query, which the performative is in reply to.
:reply-with	Defines whether the sender expects a reply and if so a label for the reply.
:content	Defines the content of the message; equivalently denotes the object of the action. The meaning of the content of any ACL message is intended to be interpreted by the receiver of the message.
:protocol	Defines the interaction protocol that the sending agent is employing with this ACL message.
:language	Defines the language in which the content parameter is expressed.
:ontology	Defines the ontology(s) used to give a meaning to the symbols in the content expression

Table 5-2 FIPA-SL parameters as defined in [21]

A FIPA ACL message contains a set of one or more message parameters, indicated by a ':' – these are illustrated in Table 5-2. Precisely which parameters are needed for effective agent communication will vary according to the situation; the only parameter that is mandatory in all ACL messages is the performative, although most ACL messages will also contain sender, receiver and content parameters.

5.3.4.2 Ontology

Ontologies [22][23] have been developed in order to provide a domain specific vocabulary for inter-agent communication.

In the context of knowledge sharing the term "ontology" is used as a description of the concepts and relationships that can exist for an agent, or a group of agents, in a specified formal vocabulary. This means that in order for the agents to communicate in an efficient way, they have to use a formal context of knowledge representation so that they infer the same meaning for the same concepts referenced. The set of objects and the relationships between them are represented in a logical formalism of a vocabulary. There are certain definitions associated with the names of the different entities within the problem domain (types of entities, their attributes and their properties, the entities' relations and functions and any of their possible constraints) in a human readable text describing what these names mean and certain axioms that constrain interpretation.

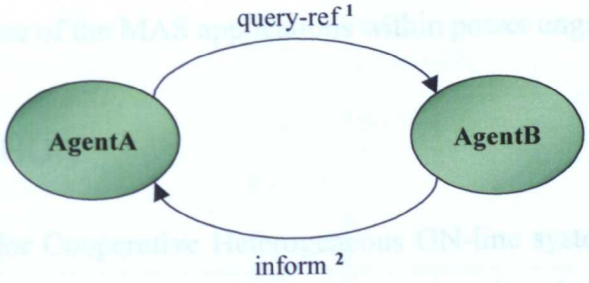
In a MAS the agents share the same vocabulary, but this doesn't mean that they share a knowledge base. Each agent might have different knowledge to that of the others, but a shared vocabulary is essential in order to achieve their communication in a coherent and consistent manner.

5.3.4.3 Message Content Language

Most ACLs do not specify a syntax for message contents, with the rationale being that different application domains may require different content languages. Nonetheless, a number of general-purpose content languages have been developed, e.g. the Knowledge Interchange Format (KIF) [24], typically used with KQML, and the FIPA Semantic Language (FIPA-SL) [25] for use with the FIPA ACL. The research presented in this thesis employs the more common FIPA-SL.

FIPA-SL provides a syntax for message content, which is based on the formalism of predicate logic. It defines some 'built-in' constants, functions and predicates, and any others used in any given content expression are assumed to be defined in the ontology referenced by the ':ontology' message parameter.

To illustrate the use of FIPA ACL and FIPA-SL during agent interactions, a simple example is presented in Figure 5-3.



query-ref message 1

```
(query-ref
   :sender     AgentA
   :receiver   AgentB
   :language   fipa-sl
   :ontology   transport
   :content    "((any ?x (is-car ?x)))")
```

inform message 2

```
(inform
   :sender     AgentB
   :receiver   AgentA
   :language   fipa-sl
   :ontology   transport
   :content    "(= (any ?x (is-car ?x))
                  (car
                     :colour        blue
                     :registration  "S460 DGD"
                     :make          renault
                     :type          clio))")
```

Figure 5-3 Illustration of FIPA ACL and FIPA-SL for message content

Figure 5-3 illustrates a simple FIPA message exchange between two agents, where AgentA is querying AgentB, enquiring whether AgentB has any knowledge relating to a car. The query is successful and AgentB replies using an 'inform' performative with message content, structured using FIPA-SL, providing knowledge on a car.

5.4 MAS Power Engineering Applications

Although the MAS approach has been applied within the power industry, the post-fault disturbance analysis arena has been overlooked, an oversight this research aimed to address. Furthermore, many of the reported industrial applications of MAS have been developed in an ad hoc fashion, following little or no rigorous design methodology and with limited specification of the requirements or design of the

agents or MAS as a whole [26]. This issue will also be addressed in chapter six of this thesis.

To provide a setting for the agent research presented in the remaining chapters of this thesis, a number of the MAS applications within power engineering are presented.

5.4.1 ARCHON

Architecture for Cooperative Heterogeneous ON-line systems (ARCHON) [27][28] was one of the first applications of agent-based technology within the power industry. ARCHON was developed during an ESPRIT project, the motivation of which was the integration of pre-existing computer tools that were not originally intended to interoperate.

ARCHON was used to integrate four pre-existing expert systems into an architecture designed to provide decision support to control engineers managing the distribution network in northern Spain. Each expert system is assigned one agent as a wrapper, which maintains a model of the capability of other agents in the system. The resulting architecture consists of four agents, each offering a different decision support function: control systems interface, blackout area identifier, an alarm analyser and a service restoration planner.

Figure 5-4 presents the common modular architecture of the ARCHON layer that wraps the different expert systems with: high-level communications manager (HLCM) for network interfacing; planning and coordination module (PCM), to establish and maintain cooperative activity; acquaintance model AM) to maintain information on the abilities of other agents; self model (SM) to represent the current state of the wrapped intelligent system and the monitor module for interfacing with the existing intelligent system code.

ARCHON is a closed MAS, where the configuration of agents has been defined when the system was developed, and each agent provided with knowledge of its acquaintances abilities, thereby providing a non-extensible and inflexible architecture. Furthermore, although ARCHON does follow a primitive speech act

protocol similar to FIPA ACL and KQML, it was developed before any of these accepted standards were mature; consequently it uses a proprietary ACL.

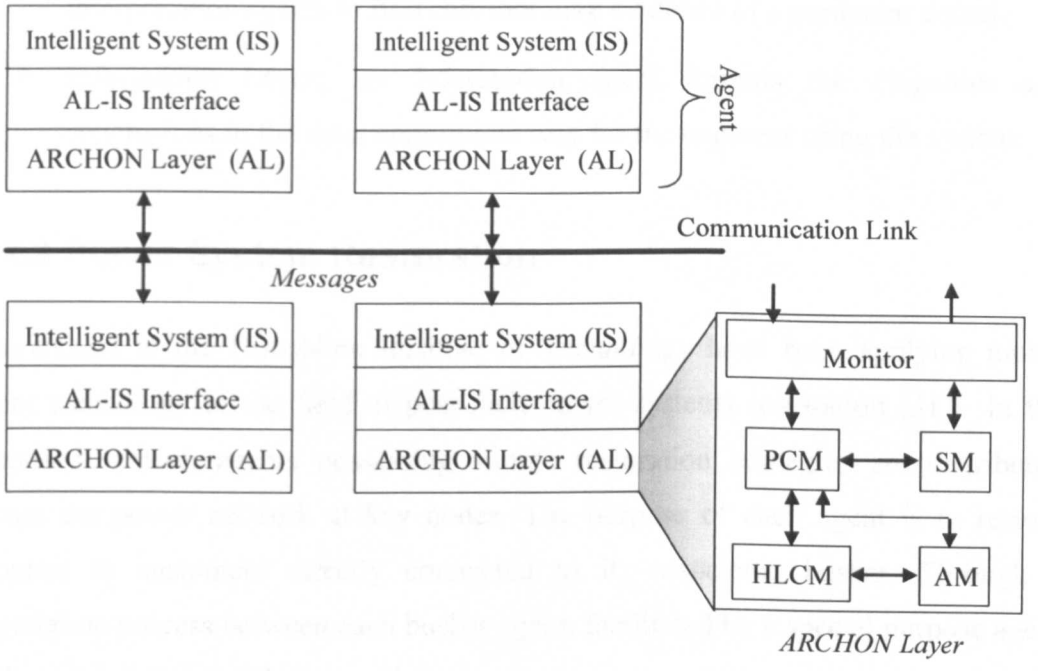


Figure 5-4 Structure of ARCHON community and ARCHON layer [29]

5.4.2 COMMAS

Another more recent application of MAS is to the condition monitoring of industrial plant, such as power transformers, gas turbines and Gas Insulated Switchgear (GIS). The COMMAS (COndition Monitoring Multi Agent System) developed by Mangina [30] provides a layered condition monitoring system, where functional modules are grouped by their overall goal. Architecturally, the condition monitoring system uses distributed agents that have no constraints on their physical location. This allows data handling agents to be on the plant or in close proximity.

Each layer within COMMAS contains a number of agents performing different functions:

- Data Layer: Agents which interpret data, extracting statistical features from the data and performing basic calculations.

- Interpretation Layer: Agents employing various AI and data interpretation techniques to interpret the data provided by agents in the data layer.
- Corroboration Layer: Agents use the different information provided by the interpretation agents to find corroborative evidence of a particular defect.
- Information Layer: An information agent formats the diagnoses and conclusions in the most appropriate way for the engineer using the system.

5.4.3 Power System Restoration

Researchers at the Hiroshima Institute of Technology have been applying multi-agent technology to the field of post-fault power systems restoration [31]. In the proposed system, agents possessing simple restoration strategies are distributed across the power network at key nodes. The purpose of each agent is to restore supplies to customers directly connected to its associated busbar. Through a negotiation process between each busbar agent, facilitated by a special purpose agent with a global view of the network, automated power system restoration is achieved. Inter-agent communications are facilitated by a KQML message scheme.

The research is still in its early stages, having not moved much beyond simulations using representative models of local distribution systems. Nevertheless, the researchers have reported promising results, demonstrating the validity and effectiveness of the proposed MAS. Research is to continue, with the performance of the MAS being improved in order for it to cope with multiple faults [32].

5.4.4 SPID

The Strategic Power Infrastructure Defence (SPID) research program [33], started in 2000 and funded by EPRI and the U.S. Department of Defence, has been developing a new concept for the defence of power systems. Using MAS technologies, the SPID architecture should be capable of assessing power system vulnerability, monitoring for hidden failures in protection schemes, and providing adaptive control to prevent catastrophic failures and cascading sequences of events. As yet, SPID has not advanced far beyond the conceptual stage and has not been implemented on an

operational network, nonetheless it is one of the most active research projects in MAS applied to power systems.

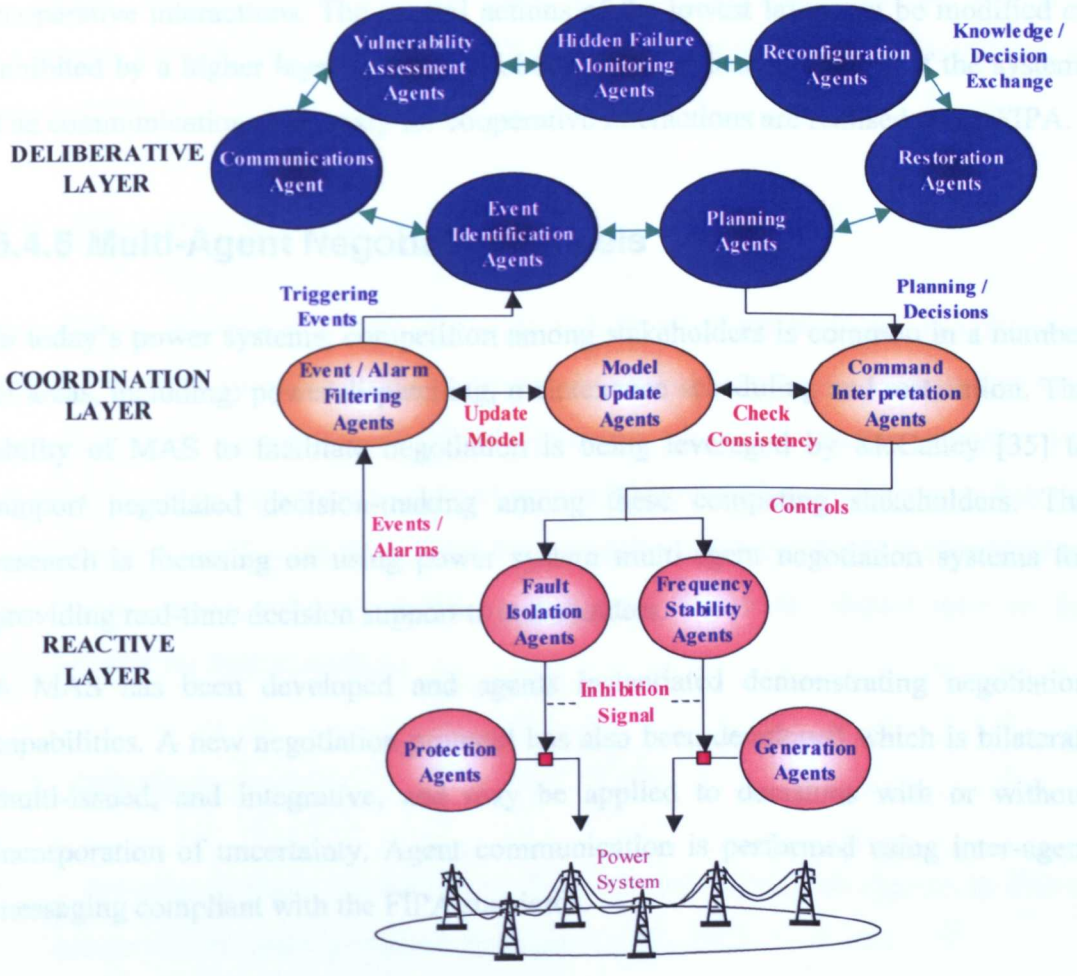


Figure 5-5 The conceptual architecture of SPID [34]

SPID will achieve power systems vulnerability assessment and self-healing network reconfiguration control using three agent layers, as illustrated in Figure 5-5:

- *Reactive Layer:* Agents perform local subsystems or components control with a fast response time.
- *Deliberative Layer:* Agents analyse, monitor and control power systems from a global point of view.
- *Coordination Layer:* Agents examine the consistency of decisions received from the deliberative layer with the current model of the power system. These

agents are also required to map the decisions from the deliberative layer into control signals that can be accepted by the agents in the reactive layer.

SPID's global goal is achieved by agents working together in the context of cooperative interactions. The control actions of the lowest layer can be modified or inhibited by a higher layer in order to obtain the coordinated control of the system. The communications necessary for cooperative interactions are realised using FIPA.

5.4.5 Multi-Agent Negotiation Models

In today's power systems, competition among stakeholders is common in a number of areas, including: power dispatching, maintenance scheduling and restoration. The ability of MAS to facilitate negotiation is being leveraged by McCalley [35] to support negotiated decision-making among these competing stakeholders. The research is focussing on using power system multi-agent negotiation systems for providing real-time decision support to stakeholders.

A MAS has been developed and agents instantiated demonstrating negotiation capabilities. A new negotiation protocol has also been developed which is bilateral, multi-issued, and integrative, and may be applied to decisions with or without incorporation of uncertainty. Agent communication is performed using inter-agent messaging compliant with the FIPA standards.

5.5 MAS and Hybrid Intelligent Systems

The foregoing discussion has introduced MAS as systems composed of multiple autonomous components coordinating their behaviour and working together to reach the overall global goal of the architecture. This description if very similar to that of hybrid intelligent system presented in chapter three. However, only if the MAS architecture consists of intelligent agents employing two or more distinct intelligent reasoning techniques can the architecture be considered as truly hybrid.

With regards hybrid intelligent systems, MAS are particularly useful since they provide a flexible and extensible platform for implementing such systems. The use

of utility agents together with a common communications vocabulary and protocols facilitates the introduction of new intelligent systems. By facilitating such introductions, new reasoning techniques can easily be introduced to provide additional problem solving and data analysis capabilities, complementing the other techniques already used within the MAS and providing an overall enhanced level of functionality.

5.6 Chapter Summary

This chapter introduced intelligent agents and described the main characteristics of MAS. Finally some of the major applications of MAS in power engineering have been described. The motivation of this overview of multi-agent technologies was twofold. Firstly to represent the general aspects of MAS and secondly to establish the agent terminology for use throughout the remaining chapters of this thesis. Most of the agents' issues are not covered in detail and the reader should refer to the bibliography for further reading.

5.7 Bibliography

- C. Rehantz (Editor), "Autonomous Systems and Intelligent Agents in Power System Control and Operation", Springer-Verlag, 2003.
- G. Weiss, "Multiagent Systems: A Modern Approach, MIT Press, 1999.
- M.A. Hugns, "Readings in Agents", Morgan Kaufmann Publishers, 1998.
- J. Bradshaw, "Software Agents", MIT Press, 1997.
- R. Khosla, T. Dillon, "Engineering Intelligent Hybrid Multi-Agent Systems", Kluwer Academic Publishers, 1997.
- A. Bond, L. Gasser (Editors), "Readings in Distributed Artificial Intelligence", Morgan Kaufmann Publishers, 1988.

5.8 References

[1]. S. Franklin, A. Graesser, "Is it an Agent, or just a Program: A Taxonomy for Autonomous Agents", Proc. of the Third International Workshop on Agent Theories, Architectures and Languages", Hiedelberg, Germany, 1996.

[2]. M. Wooldridge, N.R. Jennings, "Intelligent Agents: Theory and Practice", Knowledge Engineering Review, v10, n2, 1995.

[3]. L. Gasser, "Foreword In: Readings in Agents", M.N. Huhns, and M.P. Singh (Editors), Morgan Kaufmann Publishers, pages v-vi, 1998.

[4]. E.H. Durfee, "What Your Computer Really Needs to Know, You Learned in Kindergarten", Proceedings of the Tenth National Conference on Artificial Intelligence, pages 858-864, July 1992

[5]. E.H. Durfee, V.R. Lesser, D.D. Corkill, "Trends in Cooperative Distributed Problem Solving", IEEE Transactions on Knowledge and Data Engineering, v1, n1, pages 63-83, March 1989.

[6]. N.R. Jennings, K. Sycara, M. Wooldridge, "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems Journal, N.R. Jennings, K. Sycara, M. Wooldridge (Editors), Kluwer Academic Publishers, v1, n1, pages 7-38, 1998.

[7]. H. Van Dyke Parunak, "Practical and Industrial Applications of Agent-Based Systems", Environmental Research Institute of Michigan (ERIM), 1998.

[8]. N. Jennings, "Applying Agent Technology", Plenary presentation at PAAM'96, 1996.

[9]. K. Decker, K. Sycara, M. Williamson, "Middle-Agents for the Internet", Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI-97), January, 1997.

[10]. J.C. Collis, D.T. Ndumu, H.S. Nwana, L.C. Lee, "ZEUS agent building tool-kit", BT Technology Journal, 16(3), p 60-68, 1998.

[11]. G. Wiederhold, "Mediators in the Architecture of Future Information Systems", IEEE Computer, March 1992, pages 38-49.

[12]. K. Decker, M. Williamson, K. Sycara, "Matchmaking and Brokering", Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), December, 1996.

[13]. P.R. Cohen, A. Cheyer, M. Wang, S.C. Baeg, "An open agent architecture", Proceedings of the AAAI Spring Symposium. 1994.

[14]. L. Thomas, "Games, Theory and Applications", Ellis Horwood Series in Mathematics and Its Applications, G.M. Bell (Editor), Prentice Hall Europe, 1984.

[15]. R. Smith, "The Contract Net Protocol: High Level Communication and Distributed Problem Solver", Readings in Distributed Artificial Intelligence, A.Bond and L. Gasser (Editors), Morgan Kaufmann Publishing, pp 357-366, 1988.

[16]. G. Zlotkin, S. Rosenschein, "A Domain Theory for Task Oriented Negotiation, Proceedings of IJCAI'93, pp 416-422, 1993.

[17]. T. Finin, Y. Labrou, J. Mayfield, "KQML as an agent communication language", Software Agents, J. Bradshaw (Editor), MIT Press, Cambridge, 1997.

[18]. R.S. Patil, et al., "The DARPA knowledge sharing effort: Progress report", Proceedings of Knowledge Representation and Reasoning, pp 777-788, 1992.

[19]. S. Cranefield, M. Purvis, "Referencing Objects in FIPA SL: An Analysis and Proposal", Proceedings of the 2nd International Workshop on Challenges in Open Agent Environments, AAMAS 2003.

[20]. "FIPA Communicative Act Library Specification", XC00037H, [Online], Available: http://www.fipa.org/repository/index.html

[21]. "FIPA ACL Message Structure Specification", SC00061G, [Online], Available: http://www.fipa.org/repository/index.html

[22]. T.R. Gruber, "A Translation Approach to Portable Ontologies", Knowledge Acquisition, v5, n2, pp 199-220, 1993.

[23]. T.R. Griber, "Towards principles for the design of ontologies used for knowledge sharing", presented at The Padua Workshop on Formal Ontology, 1993.

[24]. M.R. Genesereth, "Knowledge Interchange Format", Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, pp 589-600, 1991.

[25]. "FIPA SL Content Language Specification", XC00008G, [Online], Available: http://www.fipa.org/repository/index.html

[26]. M. Luck, P. McBurney, C. Preist, C. Guilfoyle, "Agent Technology Roadmap", [Online], Available, http://www.AgentLink.org, October 2002.

[27]. N.R. Jennings, T. Wittig, "ARCHON: Theory and Practice", Distributed Artificial Intelligence: Theory and Practice, N.M. Avouris (Editor), Kluwer Academic Press, 1992, pp 179-195.

[28]. L.Z. Varga, N.R. Jennings, D. Cockburn, "Integrating Intelligent Systems into a Cooperating Community for Electricity Distribution Management", Expert Systems with Applications, v7, n4, 1994, pp 563-579.

[29]. D. Cockburn, N.R. Jennings, "ARCHON: A Distributed Artificial Intelligence System for Industrial Applications", Foundations of Distributed Artificial Intelligence, G.M.P. O'Hare and N.R. Jennings (Editors), Wiley, pp 319-344, 1996.

[30]. E. Mangina, S.D.J. McArthur, J.R. McDonald, "COMMAS (COndition Monitoring Multi Agent System)", Journal of Autonomous Agents and Multi-Agent Systems, v4, n3, pp 279-281, September 2001.

[31]. T. Nagata, H. Sasaki, "A Multi-Agent Approach to Power System Restoration", IEEE Transactions on Power Systems, v17, n2, pp 457-462, May 2002.

[32]. T. Nagata, H. Sasaki, "A Multi-Agent Approach to Power System Restoration", Autonomous Systems and Intelligent Agents in Power System Control and Operation, Christian Rehantz (Editor), Springer-Verlag, 2003, pp 101-114.

[33]. C-C. Liu, H. Li, Y. Zoka, "New Applications of Multi-Agent System Technologies to Power Systems", Autonomous Systems and Intelligent Agents in Power System Control and Operation, Christian Rehantz (Editor), Springer-Verlag, 2003, pp 247-277.

[34]. C-C. Liu, J. Jung, "Multi-Agent Systems and Their Applications in a Competitive Industry Environment", ISAP Plenary Session Presentation, 2001.

[35]. V. Vishwanathan, J. McCalley, V. Honovar, "A Multiagent System Infrastructure and Negotiation Framework for Electric Power Systems", IEEE Porto Power Tech Conference, Porto, Portugal, 10th-13th September 2001.

Chapter 6: A Methodology for the Specification of MAS for Power Engineering Decision Support

6.1 Chapter Overview

Research conducted by Sycara in 1998 [1] indicated that there were few industrial strength applications of MAS technology. Sycara attributed this to the lack of proven methodologies enabling designers to clearly structure applications as MAS and the absence of tool-kits to facilitate their implementation. In the six years since, a range of MAS design methodologies and industrial strength toolkits have been developed, providing the necessary tools to increase the profile of MAS within industry.

Although numerous methodologies have been proposed, few have been evaluated in online, near real-time operational environments and none have been developed in the field of decision support for power engineering. This chapter addresses this issue, by presenting a new methodology for the specification of MAS for power engineering decision support. Before, presenting the new methodology, the characteristics of decision support within the power industry are described and the suitability of existing MAS design methodologies for specifying such systems assessed.

6.2 Power Engineering Decision Support

As monitoring technologies have evolved from simple data capture devices to more advanced devices recording the condition of the circuits and plant, which constitute the network, the volumes and complexity of data available to engineers has increased. Combined with experience and knowledge, engineers use this data as a basis for decision-making.

As described in the earlier chapters of this thesis, a range of software tools are commonly available to assist in this decision making process. These decision support tools can be characterised by a number of common features:

- **Distributed Systems:** Data gathering devices are often located on, or in close proximity to plant, leading to a large number of hardware and software systems distributed over a significant geographical area. Additionally, software systems for retrieving, visualizing and interpreting the data are often located in offices remote from the data capture devices.

- **Heterogeneous Data:** Communicated decision support data is heterogeneous. The data types can range from the simple textual representation of plant status found in SCADA alarms to the large data files generated by DFRs. The differing data sizes also mean that data is not always immediately available with data retrieval sometimes taking several minutes.
- **Online, Near Real-time Operational Environment:** The majority of existing DSS offer support in 'live', or online, operational environments by interpreting monitoring data in near real-time. In such an environment, the problems of unpredictable data volumes and intermittent communications to remote devices need to be managed if timely support is to be provided.
- **Legacy Systems:** Legacy systems are a common feature and are often integral parts of complex data management and decision support schemes. It can be prohibitively expensive to redesign these existing software systems in line with new technologies. As a result there may be several generations of software managing the retrieval and analysis of data from different devices.
- **System Turnover:** New monitoring technologies, and their associated proprietary data gathering and visualization systems, are introduced relatively frequently leading to a high turnover of software and hardware systems.

As described in section 5.3.1 of chapter five, MAS are particularly suited for applications which are modular, decentralised, changeable, ill-structured and complex. The common features of power engineering decision support exhibit all these characteristics, indicating the suitability of MAS for enhancing decision support within the power industry. A key element in achieving this is the creation of a methodology for specifying MAS, which considers all the characteristics of the decision support systems.

6.3 Methodologies for MAS Specification

A number of methodologies have been proposed by the agent research community [2] for specifying MAS. Many, such as the Styx methodology, are only in the prototype stage [3]. Of the more developed methodologies, some are closely linked

to a particular MAS development tool-kit such as the Zeus tool-kit [4], others are focussed on the development of MAS employing particular types of agents [5]. All of the developed methodologies either extend traditional software design methodologies such as the Unified Modelling Language (UML) [6] or knowledge engineering methodologies such as CommonKADS [7].

The following sections review four of the most developed methodologies, highlighting the key features of each. A discussion on their suitability for developing MAS for power engineering decision support is then presented.

6.3.1 MAS-CommonKADS

The CommonKADS knowledge engineering methodology is traditionally used to capture and structure the knowledge required within a centralised monolithic Knowledge Based System (KBS) [8]. A description of the CommonKADS methodology was presented earlier in chapter three. For the purposes of this discussion merely remember that CommonKADS requires construction of six models capturing the salient features of the KBS and the organisation in which it will reside, namely: organisation, task, agent, communication, expertise and design models.

In MAS where the software agents are considered 'intelligent', the CommonKADS methodology is suitable for acquiring the agent knowledge. However, the main restrictions for the direct application of CommonKADS to MAS come from the CommonKADS Communication Model. The Communication Model is unable to represent agent cooperation and interactions for a number of reasons [7]:

- The Communication Model deals mostly with human-computer interaction and is very restrictive for computer-computer interaction.
- The primitives of a protocol for complex interactions are not considered.
- The Communication Model does not address multi-agent transactions.

To overcome these limitations the MAS-CommonKADS [7] methodology has been developed which extends CommonKADS with an additional Coordination Model.

The methodology consists of seven stages that are applied iteratively until a design model is obtained for the MAS. The MAS-CommonKADS methodology is illustrated in Figure 6-1.

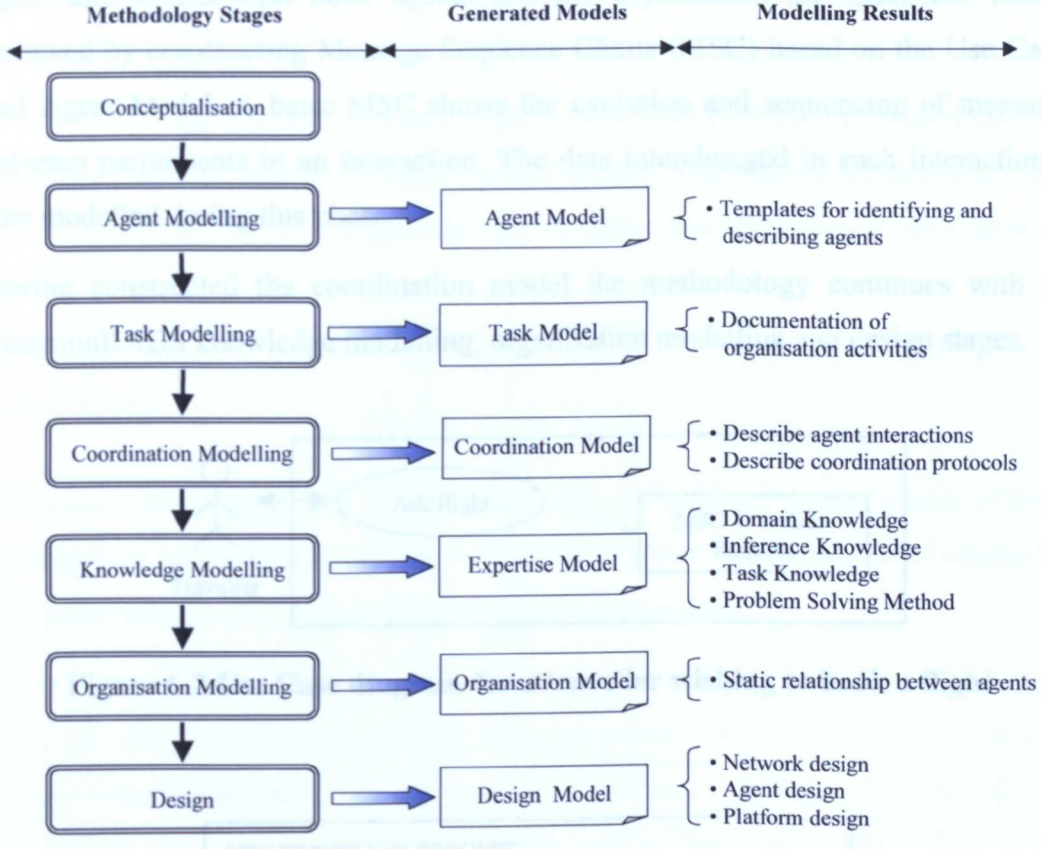


Figure 6-1 MAS-CommonKADS methodology

Application of the methodology begins with a conceptualisation phase where a knowledge elicitation process is conducted to obtain a preliminary description of the problem the MAS is being designed to solve. During this process UML [9] Use Cases are identified which capture the informal requirements and enable later testing of the system.

The methodology then moves onto the analysis phase beginning with agent modelling where the problem description and Use Cases are analysed to identify the agents required within the MAS. The activities of the organisation are then documented during task modelling where tasks are decomposed following a top-

down approach into a task hierarchy. This documentation serves for supporting the maintenance and management of changes in the organisation.

The next stage is coordination modelling where the social and distributed nature of agents within a MAS are modelled. During coordination modelling the required agent interactions with other agents and the organisation are specified. This is achieved by constructing Message Sequence Charts (MSC) based on the Use Cases and Agent Model. A basic MSC shows the evolution and sequencing of messages between participants in an interaction. The data interchanged in each interaction is also modelled during this phase.

Having constructed the coordination model the methodology continues with the CommonKADS knowledge modelling, organisation modelling and design stages.

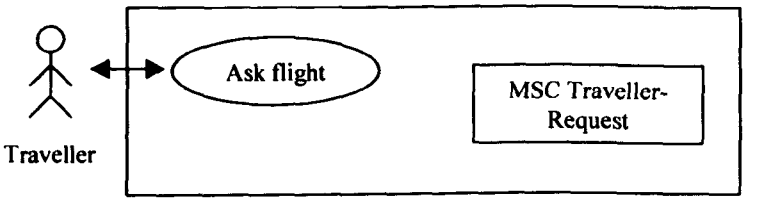


Figure 6-2 Use Case diagram for a traveller wishing to book a flight

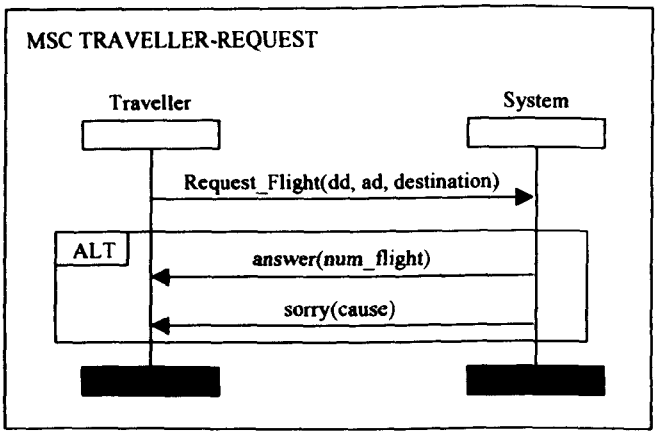


Figure 6-3 MSC for traveller requesting a flight.

To illustrate the Use Cases and Message Sequence Charts used during conceptualisation and coordination modelling the case study used in [7] will be briefly described. The problem consists of building a system that is consulted by a

user (Traveller) for booking a flight, and answers with the flight number (num-flight) of the cheapest available flight with the lowest probability of delay.

Using UML, the interaction between the Traveller and the system can be represented by the Use Case diagram in Figure 6-2. This diagram shows that the Traveller uses the 'Ask Flight' system function. The interactions in this Use Case diagram are formalised using the MSC indicated in Figure 6-3.

In Figure 6-3, the Traveller requests flight details from the System, by providing the departure date (dd), arrival date (ad) and destination. The System can then reply with two alternatives (ALT): a flight number, or sorry and the cause if no flights are available.

6.3.2 Gaia

The Gaia methodology [10] only addresses the analysis and design phases of MAS development assuming that the conceptualisation phase has already been conducted. The main models used in Gaia are presented in Figure 6-4.

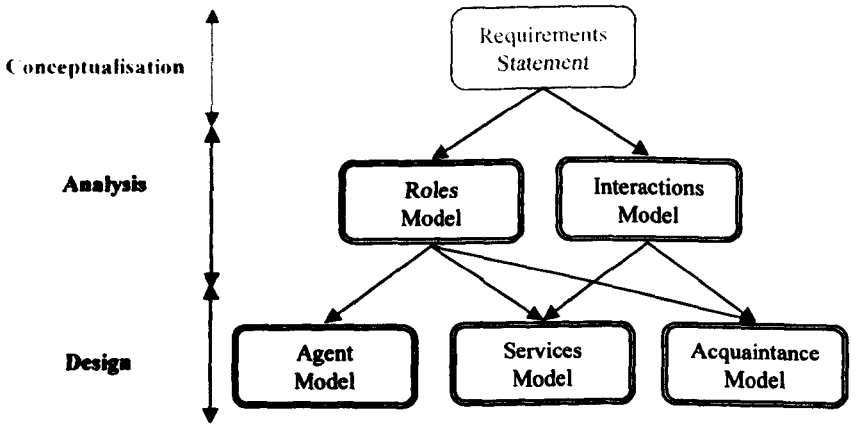


Figure 6-4 Relationships between Gaia models

The objective of the analysis phase is to develop an understanding of the MAS and its organisational structure through creation of a Roles Model and an Interactions Model. The aim of the design phase is to transform the analysis models into a sufficiently low level of abstraction that traditional software design techniques may

be applied in order to implement the agents. How an agent actually realises its functionality is considered beyond the scope of Gaia.

The organisation is viewed as a collection of related roles that take part in systematic patterns of interactions with other roles. A role can be viewed as an abstract description of an agent's expected function and is defined by four attributes:

- Responsibilities: These determine the functionality of the role and are split into *liveness properties* and *safety properties*. Liveness properties describe the desired states a role must achieve. Safety properties describe the states an agent must ensure are always maintained.
- Permissions: These identify the resources that are available to a role in order to realise its responsibilities.
- Activities: The computations associated with the role that may be carried out without interacting with other roles.
- Protocols: These define role interactions.

Having generated a preliminary Roles Model, an Interactions Model is created which identifies and documents the associated agent interactions in the form of protocol definitions, one for each type of inter-role interaction. At this stage attention is focussed on the essential nature and purpose of the interaction, rather than on the precise ordering of particular message exchanges.

Role Schema:	*name of role*
Description:	*short description of the role*
Protocols and Activities:	*protocols and activities in which the role takes part*
Permissions:	*'rights' associated with the role*
Responsibilities:	
Liveness:	*liveness responsibilities*
Safety:	*safety responsibilities*

Figure 6-5 Gaia template for a Role Schema

The analysis process continues with iteration of the Roles Modelling and interactions modelling until an elaborated roles model is realised which documents the key roles occurring in the system, their permissions and responsibilities, together with the protocols and activities in which they participate. Each role is documented using a Role Schema as illustrated in Figure 6-5.

The methodology continues with the design phase in which three separate models are developed to assist with implementation of the MAS using traditional software design techniques. The models are:

- Agent Model: Identifies the agent types that will make up the system. An agent type is best thought of as a set of agent roles. There may be a one-to-one correspondence between roles and agent types. However, this need not be the case. A designer can choose to package a number of closely related roles in the same agent type for the purpose of convenience or efficiency.
- Services Model: Identifies the main services, or functions, required to realise a role. The services that an agent will perform are derived from the list of protocols, activities, responsibilities and the properties of a role.
- Acquaintance Model: Defines the communication links that exist between agent types. It does not define what messages are sent or when, it simply indicates that a communications path exists. The model is derived from the roles, protocols and agent models.

The Gaia methodology assumes that the MAS is closed and the systems' organisational structure is static, i.e. inter-agent relationships and agent abilities do not change at run-time.

6.3.3 DESIRE

DESIRE (DEsign and Specification of Interacting REasoning components) is a compositional modelling framework originally conceived as a means of specifying complex software systems [11]. The authors of DESIRE consider that it is suited to the specification of MAS due to its philosophy of viewing the complex system as a series of interacting, task based, hierarchically structured components. This view has been affirmed by using DESIRE to create a formal specification of the existing ARCHON multi-agent system (*outlined in section 5.4.1*) [12].

DESIRE does not provide a detailed methodology for the entire MAS development cycle but instead provides a framework for supporting its specification. It is assumed that knowledge acquisition and requirements capture have already been conducted and that the required agents have already been identified based on high-level functional requirements.

To identify the necessary tasks, the compositional modelling process begins with task decomposition. This involves decomposition of the overall system task into a set of composed and primitive tasks documented in a task hierarchy. In contrast to primitive tasks, composed tasks are tasks for which subtasks are identified. An example of the task hierarchy for ARCHON is presented in Figure 6-6. The tasks are delegated to agents by deciding which agents would best perform which tasks.

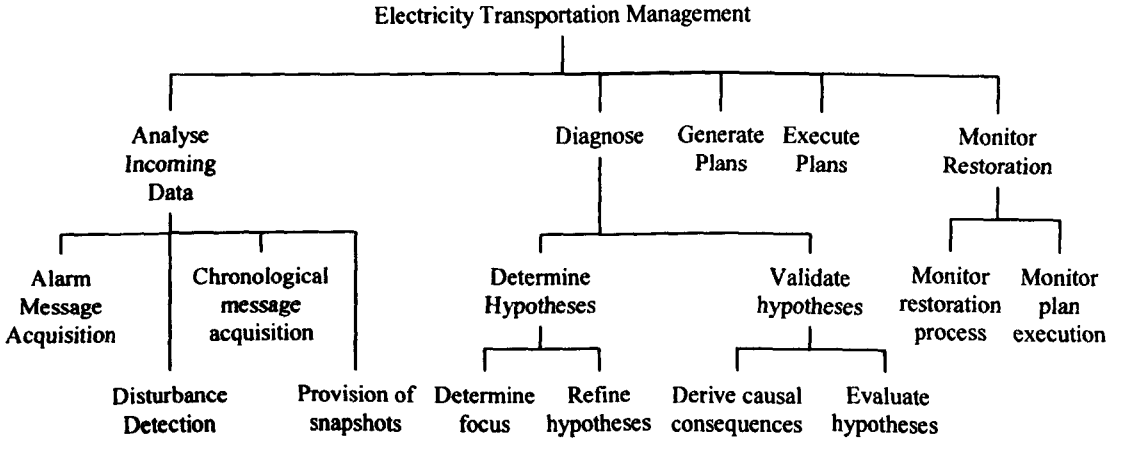


Figure 6-6 ARCHON Task Hierarchy

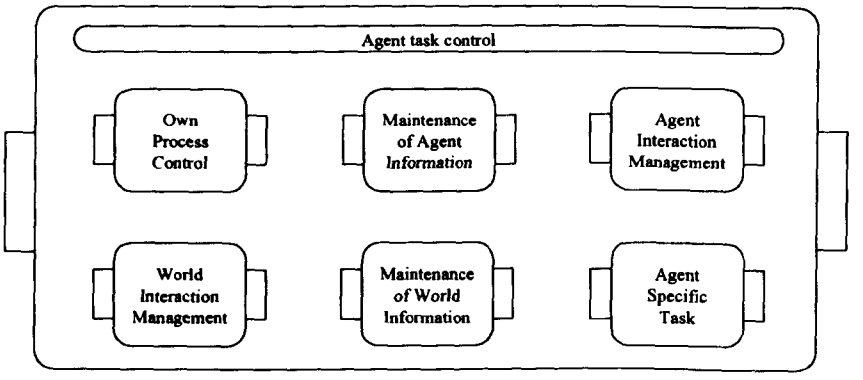


Figure 6-7 DESIRE generic compositional model for the weak agent notion

Central to the DESIRE framework is a library of generic models which can be modified or refined to produce compositional models of the agents and their assigned tasks. These generic models represent different classes of agents, e.g. Belief Desire Intention (BDI) and the weak notion of agency. A generic model for agents adopting the weak notion of agency is presented in Figure 6-7.

The model inputs and outputs are represented diagrammatically as rectangular blocks on the left and right of the model respectively. The tasks performed by the agents are themselves represented as compositional models which can, in turn, be repeatedly decomposed until the associated primitive tasks are reached. The agent tasks within the weak agent model not only relate to the 'Agent Specific Task' assigned to the agent but also the tasks required to support the weak notion of agency:

- 'Own Process Control' supports *autonomy* and *pro-activeness.*
- 'Agent Interaction Management' and 'Maintenance of Agent Information' supports *social abilities*, *reactiveness* and *pro-activeness* with respect to other agents.
- 'World Interaction Management' and 'Maintenance of World Information' supports *reactiveness* and *pro-activeness* with respect to the external world.

DESIRE also recognises that within agents components can either be autonomous or controlled. Where control is required the agent's task control knowledge specifies

when and how components are to be activated (and whether activation is continuous or only for a given period).

The domain task assigned to the agent is illustrated by the 'Agent Specific Task' component of the agent model and can be further decomposed into component models. Figure 6-8 illustrates the decomposition of the 'Diagnose' agent specific task within the Alarm Analysis Agent (AAA) of ARCHON into a component model.

The information exchanges between components are modelled as *Information Links,* as illustrated in Figure 6-8, which relate output of one component to input of another.

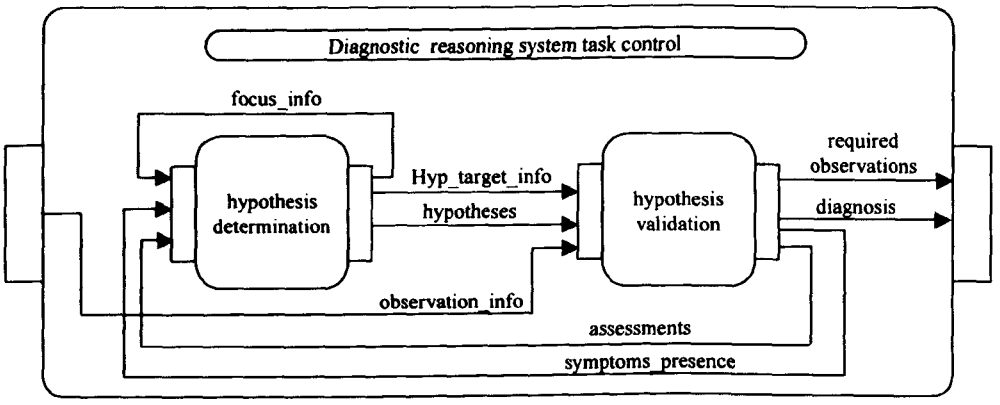


Figure 6-8 Generic task model of the diagnosis task

The required information links are determined by using the task hierarchy and considering the exchange of information between tasks. Information links are formally specified within DESIRE and an example specification for the Hyp_target_info link is as follows:

```
link        hyp_target_info: object-object
  domain        hypothesis_determination
        output          hyp_target_info
  codomain      hypothesis_validation
        input           target_info
  links         (hyp_target_info, target_info) :<<true, true>>
endlink
```

This link transfers hypothesis information determined in the hypothesis_determination component (the domain of the link) to input of the component hypothesis_validation (the codomain of the link). The links specify the relationships between the names used by the components sending and receiving the information. Different names are used for the same information to allow each component to specify information in its own language, independent of other components.

At the end of the specification process a developer using the DESIRE framework should have generated a formal specification of the MAS. This specification will consist of a large number of models representing the components within the MAS at various levels of abstraction. It will also specify the task control knowledge and required information links and is intended to be used to implement the MAS.

6.3.4 MaSE

The Multiagent Systems Engineering (MaSE) methodology [13][14] was developed to guide a multi-agent system developer from an initial systems specification to a set of formal design models. The methodology is illustrated in Figure 6-9 and has an analysis phase and a design phase each consisting of four steps. These steps are applied iteratively until a complete set of design models is realised.

The analysis phase aims to describe the system requirements through a set of roles with assigned tasks. The roles are similar in concept to those defined in Gaia. The first step in analysis is *Capturing Goals* where the system-level objectives, or goals, are identified by distilling the essence of the initial system requirements. Goals are used at this stage since they are less likely to change than the detailed tasks and interactions involved in achieving them. The goals are then analysed and structured into a goal hierarchy diagram which can be used during the design phase.

The next analysis step is *Applying Use Cases* where UML Use Cases, similar to those adopted in MAS-CommonKADS, are compiled based on the system requirements. A similar concept to the Message Sequence Chart (MSC) used in MAS-CommonKADS, namely the sequence diagram, is used to determine the minimum set of messages that must be passed between roles.

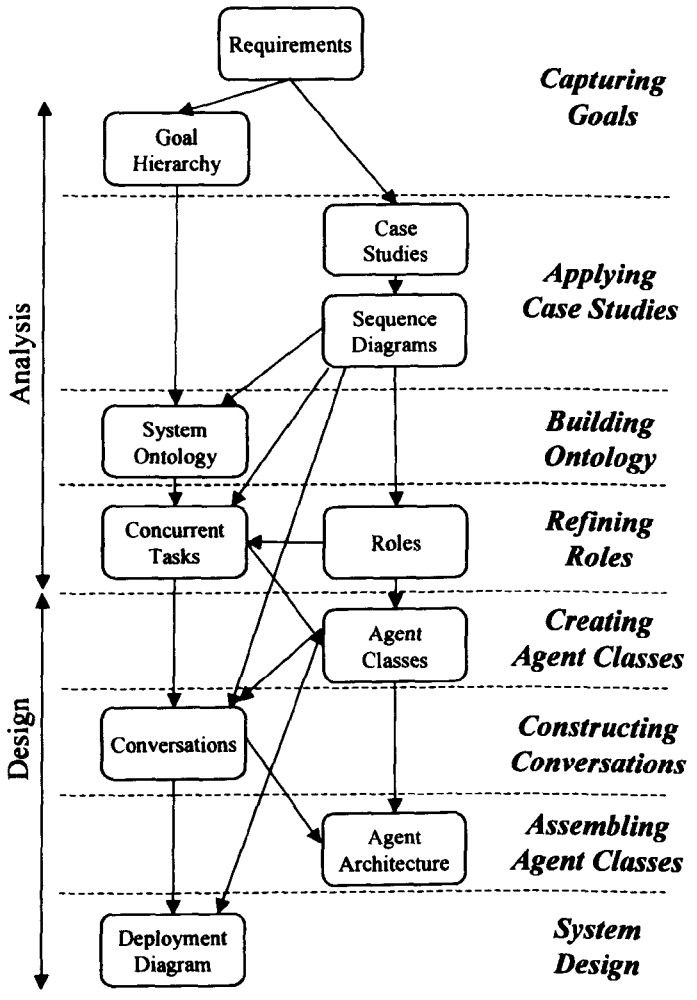


Figure 6-9 MaSE phases, steps and models

Terms from the goal hierarchy, use cases and sequence diagrams are then used during the *Building Ontology* step as possible concepts in the MAS ontology. The final step of analysis, *Refining Roles*, uses the outputs from the previous steps to create roles and assign the tasks to be performed by those roles. Tasks are associated with each role to describe the behaviour that the role must have to accomplish its assigned goals. Tasks often indicate parameter passing, so this step is placed after construction of the ontology to allow the designer to specify the type of parameters based on the classes in the ontology.

Once the system requirements have been modelled, MAS design can commence. The first step in the design phase is *Creating Agent Classes*, where the roles are assigned to specific agent classes. This step creates an Agent Class Diagram illustrating the classes, the roles played by the agent classes and the conversations between classes.

Details of the required conversations are defined in the *Constructing Conversations* step, where finite state automata are used to show the states in a conversation. Each conversation has two diagrams: one for the initiator and one for the responder of the conversation. The set of conversations that an agent class participates in is derived from the communications required by the roles that the agent plays.

The third step in design, *Assembling Agent Classes*, defines the components of the agent architecture, allowing for the logical decomposition of agents. The final step of system design creates a *Deployment Diagram* to show the amount and location of each type of agent in the system. The outputs from the design steps describe the actions and conversations used in the MAS.

6.3.5 Discussion

Each of the methodologies in the preceding sections provides a framework for the development of MAS. Although, all these methodologies assist with the specification of distributed systems, a common feature of decision support, none have been used to develop a MAS for decision support within the power industry.

Considering the main aspects of each methodology, and the features common to power engineering decision support systems, the existing methodologies fall short of being truly applicable to decision support in a number of areas:

6.3.5.1 Ontology

An ontology is an essential component of an integrated decision support architecture as it provides the vocabulary necessary for agents to request and provide heterogeneous data. New agents compliant with the ontology can also be easily introduced into the MAS. Together with the use of utility agents, an ontology can provide for an open architecture able to cope with frequent system turnover.

Only the MaSE methodology explicitly recognises the importance of an ontology and has a stage in the methodology devoted to ontology development. The other methodologies either ignore the issue or skirt around it.

6.3.5.2 Closed Architectures

The presented methodologies are only applicable to closed systems, where each agent within the MAS has been provided with knowledge of its acquaintances at design time. This does not provide for an open system, where agents can discover the location and abilities of other agents at runtime.

When implementing decision support architectures as MAS, an open architecture is essential for two reasons. Firstly, it allows for agents going incommunicado, due to temporary communications faults, and being rediscovered during run time when communications have been restored. Secondly, it enables new systems to be integrated into the architecture and obsolete systems removed whilst the system is running. Such a flexible and extensible architecture can only be achieved through the use of utility agents, a concept not entertained by the described methodologies.

6.3.5.3 Compliance with International Standards

None of the methodologies support an internationally accepted standard for agent communications, such as FIPA ACL. Compliance with an international communications standard would significantly enhance the future extensibility of a MAS. This is of importance to integrated decision support architectures, since new monitoring technologies are always coming onto the market, some requiring the development of new intelligent systems to interpret the generated data.

6.3.5.4 Legacy System Reuse

All of the methodologies focus on developing a MAS from scratch, with none considering the benefits of reusing legacy systems to provided some of the agent functionality. Consequently, none of the methodologies provide any assistance with extending legacy decision support software with the behaviour necessary for it to perform as an agent.

6.3.5.5 Specification of Data and Information Exchange Mechanisms

None of the methodologies discuss the selection of the data / information exchange mechanisms, or performatives, appropriate to interactions in an online environment. The selection of the correct mechanism at the specification stage is essential, as selection of the wrong technique may lead to unnecessary inter-agent communications, possibly resulting in reduced performance of the overall system. For example, if an agent requiring SCADA alarms was to 'subscribe' to an agent providing SCADA alarms, then a separate message for each alarm would be sent to the subscribing agent – in storm scenarios the volume of alarms and, therefore, messages could be in the tens of thousands.

6.3.5.6 Application within Online, Near Real time Environments

Finally, none of the existing methodologies have been used at the outset to specify a MAS for deployment in an online environment, where near real-time data interpretation is required. As a result, at no point in any of the methodologies can the MAS developer consider and specify the reasoning techniques to be used to perform the core functional agent tasks – predominately data interpretation in decision support.

6.4 New Methodology

The remainder of this chapter describes a new methodology taking many of the concepts present in existing methodologies and combining them with new ideas addressing the decision support issues identified in section 6.2. This methodology is for the specification of MAS for power engineering decision support.

Although the forthcoming discussion makes no specific reference to the methodology's applicability to online systems, it will be shown later in this thesis that it has been used successfully to develop the PEDA MAS for automation of post-fault disturbance analysis in an online environment.

6.4.1 Methodology Overview

Using the new methodology a developer can compile a specification of a MAS for decision support within the power industry documenting the required agents, legacy systems to be integrated, agent interactions, internal agent control, agent message handling functionality and system wide ontology. The final specification will provide both a textual description and, where appropriate, graphical illustrations of key components. To generate this specification the developer must follow an eleven-stage methodology as illustrated in Figure 6-10.

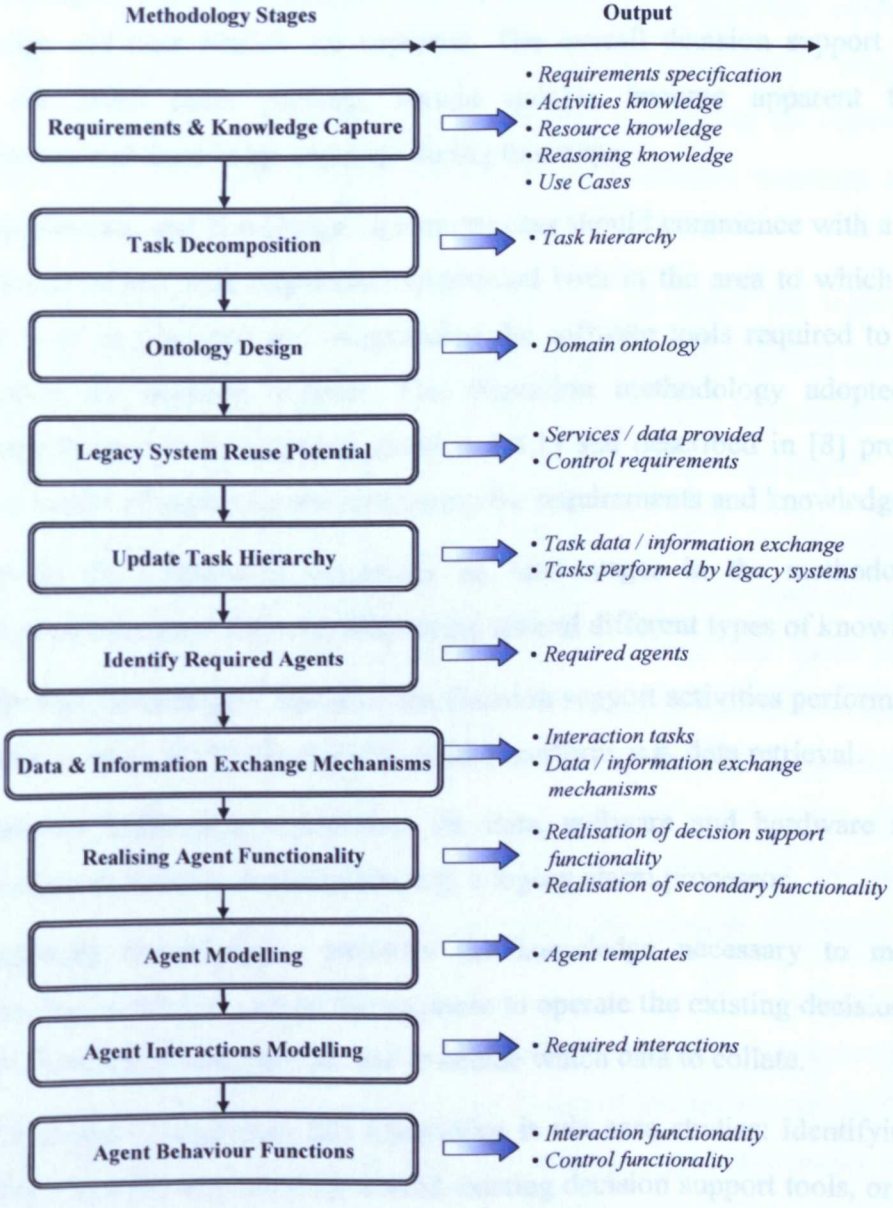


Figure 6-10 Methodology for specifying decision support MAS.

The methodology combines both a top-down approach where the main tasks required within the MAS are identified and assigned to agents and a bottom-up approach where the re-use of legacy systems constrains the top-down approach. A comprehensive description of each stage in the proposed methodology is presented in the subsequent sections.

6.4.2 Stage 1 - Requirements and Knowledge Capture

The first objective is to understand the decision support functionality required of the MAS through a high-level requirements specification, and to ensure any relevant knowledge and case studies are captured. The overall decision support function, which the MAS must perform, should quickly become apparent from the requirements and knowledge captured during this stage.

The requirements and knowledge capture process should commence with a series of elicitation meetings with engineers experienced both in the area to which decision support is to be provided and in operating the software tools required to generate information for decision support. The elicitation methodology adopted during Telemetry Processor development (section 3.4.1) and described in [8] provides an effective means of capturing and structuring the requirements and knowledge.

To provide the information necessary for later stages in the methodology the elicitation process must focus on identifying several different types of knowledge:

- **Activities knowledge** - describes the decision support activities performed by the engineer which the MAS may also need to perform, e.g. data retrieval.
- **Resource knowledge** - describes the data, software and hardware resources necessary to perform the activities, e.g. a legacy alarm processor.
- **Reasoning knowledge** - provides the knowledge necessary to mirror the reasoning processes used by the engineer to operate the existing decision support tools, interpret the output data and to decide which data to collate.

A useful means of capturing this knowledge is via case studies: identifying actual situations where the engineers have used existing decision support tools, or gathered and interpreted the data necessary for decision support. During the elicitation

meetings, each case study should be 'walked through' with each engineer and the activities performed by the engineer, the data and software resources used and the reasoning processes followed during each case study recorded. To provide a structured record of the knowledge captured during these elicitation meetings knowledge transcripts must be produced.

It is important to note that the knowledge elicited during the initial meetings may not consider all tasks required to facilitate integration and automation of legacy decision support systems within the MAS. Interaction tasks, where the engineer would previously have manually transferred data between the systems, can easily be overlooked. Before knowledge transcripts are finalised, it is therefore critical that the elicited knowledge is analysed and validated to determine whether a comprehensive coverage of the system-system and engineer-system interactions are realised. If the knowledge is lacking, further additional knowledge elicitation meetings should be conducted.

Later in the methodology it will be necessary to determine the legacy system functionality available to and used by engineers, to this end, the elicited activities and software resource knowledge should be modelled as UML Use Case diagrams. These graphical diagrams provide a useful addition to the knowledge transcripts.

The use case diagram presents a structured view of a system's functionality [9]. It does this by defining a number of *actors*, which model the roles users can play when interacting with the system, and describing the *Use Cases* that those actors can participate in. A Use Case describes one way in which a user can interact with a system. The Use Case diagram contains a set of Use Cases which should define the complete functionality of the system as seen from the user's perspective. Although a detailed description of Use Case diagrams, and Use Case modelling, can be found in [9], a simple example will be used to introduce the concept and assist with its application.

Consider the Energy Management System (EMS) used within power companies to provide a real-time picture of network status. Users of the system can fulfil a range of possible roles when interacting with the system, e.g. maintenance engineers preparing switching schedules for necessary planned outages and control engineers

approving or declining these outages and remotely operating circuit breakers via Tele-control. As illustrated in Figure 6-11, each of these roles can be depicted as an actor. The EMS functions, as viewed from the perspective of each actor, are the Use Cases and are represented as ellipses within the EMS system boundary.

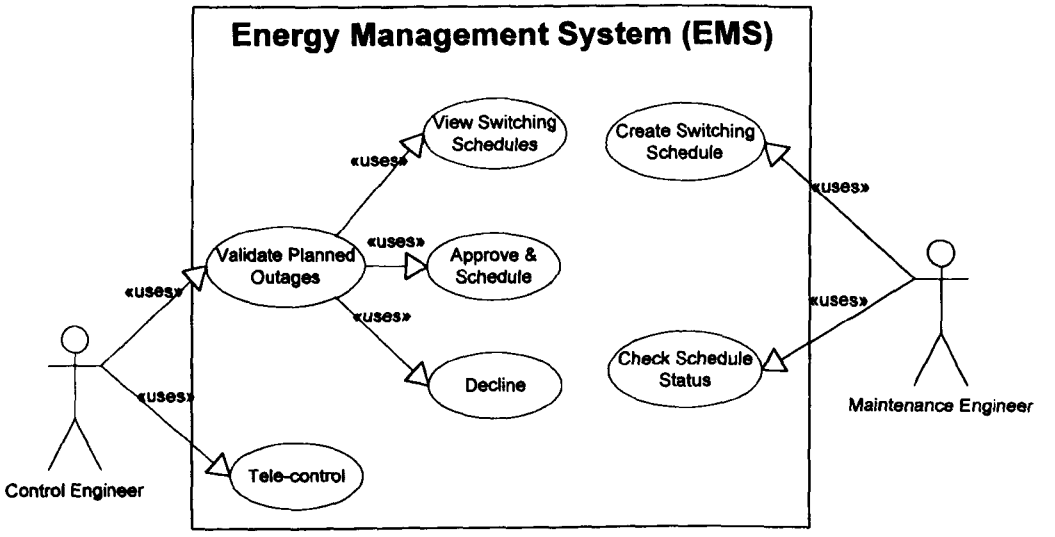


Figure 6-11 Example use case diagram for EMS system

Modelling of the elicited activities and resource knowledge as Use Case diagrams completes the first stage of the methodology. Together with the Use Case diagrams and knowledge transcripts, specification of the MAS can progress to identifying the tasks which must be performed: Stage 2 – Task Decomposition.

6.4.3 Stage 2 - Task Decomposition

In the decision support arena, a MAS will be designed to perform a particular decision support function, or high-level task. This task could be limited to the interpretation of one data type with the MAS consisting of agents, performing the same data interpretation function albeit distributed across the network, e.g. alarm interpretation at each substation. Alternatively, the task could be to provide generic decision support to a particular type of engineer with the MAS consisting of heterogeneous agents, each interpreting different data. Regardless of the identified application, in order for a MAS to perform its high-level task, the task must be

decomposed into sub-tasks which can, in turn, be assigned to agents for execution. Through a process of inter-agent collaboration, sub-task execution is achieved and the high-level task realised.

During this stage in the methodology the high-level task identified during requirements capture will be decomposed into its constituent sub-tasks so they can be assigned to agents later in the methodology. The principal output of the task decomposition process will be a task hierarchy based on the transcribed knowledge and case studies.

Generation of the task hierarchy commences with the identification of the high-level task assigned to the MAS. This high-level task is the root task of the task hierarchy, from which all sub-tasks will stem.

The knowledge transcripts and case studies are then analysed to identify the first layer of sub-tasks. In the case of a high-level decision support task the first layer will likely relate to the stages required to achieve the task. For example a condition monitoring high-level task would consist of three stages, each represented by a sub-task within the first sub-task layer as illustrated in Figure 6-12.

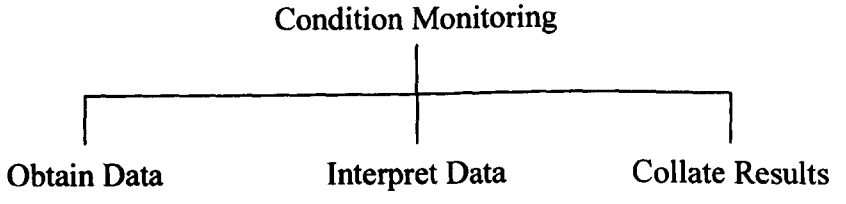


Figure 6-12 Example task hierarchy – first sub-task layer

Each of the sub-tasks are then taken in turn and the knowledge transcripts and case studies analysed to determine if the sub-task can be decomposed further into a second layer of sub-tasks. At this point, further decomposition can be justified if the task meets any one of the following criteria:

- The task requires operation, or access to, several separate systems.
- The task uses different data types each requiring a dedicated retrieval or analysis mechanism.

- The task uses several types of knowledge, e.g. rules, cases and models.

Some sub-tasks, may already be decomposed to such a level that they only require access to one particular system, e.g. a database, and only require a simple interaction with the system. To decompose such tasks further would neither be logical nor desirable.

The task hierarchy will contain tasks, which will eventually fulfil a core functional role within the agents they are later assigned to and others that will provide the social capabilities of the agents, facilitating interactions. To assist with later stages in the methodology it is useful to distinguish these interaction tasks from their functional neighbours in the task hierarchy – in this thesis a * symbol will be used.

Creation of the task hierarchy completes the second stage of the methodology. Specification of the MAS can now progress to creating a suitable ontology.

6.4.4 Stage 3 – Ontology Design

The ontology is the vocabulary used by the agents to exchange information and data resources. As such, it is the data dictionary that supports co-operation and social ability and is therefore critical to the operation of the MAS.

An ontology is formally defined by Uschold et al [15] as "an explicit formal specification of the terms in a domain and relations among them". Given this definition, the first stage in ontology design is to identify the terms used by the engineer to describe the domain concepts [16]. For example, in an asset management domain the following terms could be used to describe the domain concepts: plant life, plant type, commissioning date, purchase cost, etc.

To derive a list of terms the transcribed knowledge and task hierarchy should be analysed and every distinct term noted. At this stage it is important to get a comprehensive list of terms without worrying about overlap between the concepts they represent, relations among them or any properties that the concepts may have.

Having listed the terms the next step is to identify classes describing the domain concepts. This is achieved by looking at the list of terms for common concepts which describe the terms. For example, the terms transformer, generator and circuit breaker

can all be grouped under a class 'Plant' since they are all types of power system plant. Since transformers can either be ground mounted or pole mounted, a 'Transformer' class could be created with sub-classes 'Ground Mounted' and 'Pole Mounted'. These sub-classes represent more specific types of 'Transformer'.

The classes will enable the agents to provide and request particular types of data and information resources. To facilitate requests for specific instances of a resource, such as the commissioning date of a specific transformer, the attributes of each class need to be defined. Attributes are used to describe each class, such as 'substation', 'name' and 'commissioning date'. All sub-classes of a class inherit the attributes of that class, e.g. the 'Transformer' class would inherit the attributes of the 'Plant' class.

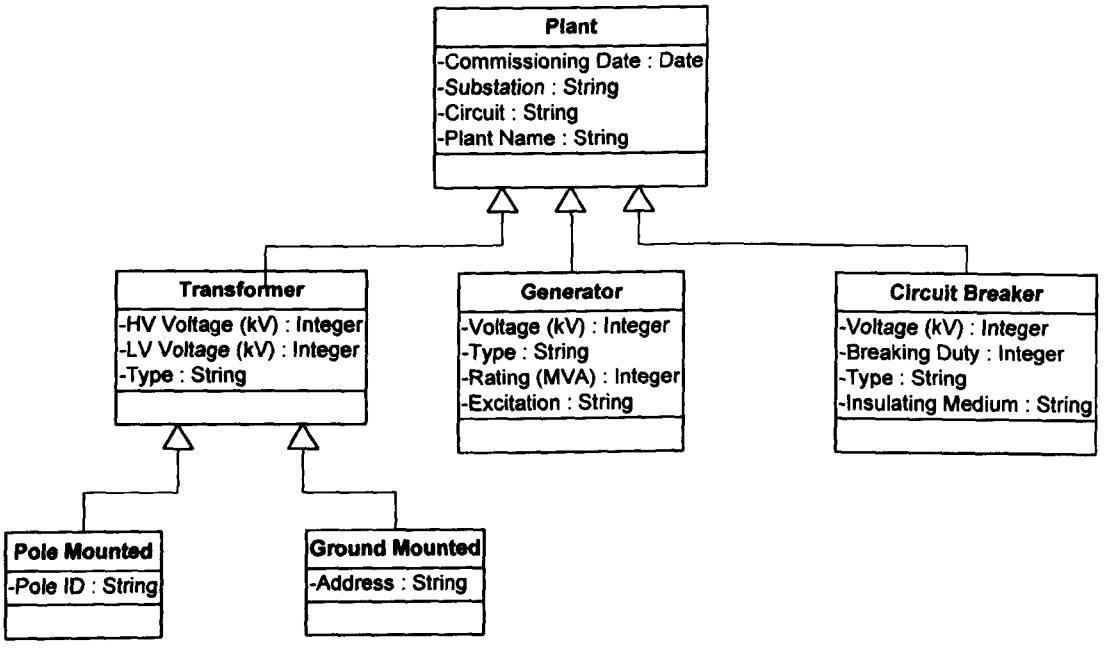


Figure 6-13 Plant class hierarchy

The class creation process should be repeated until all listed terms are assigned to classes and the classes organized into a class hierarchy. An example class hierarchy is presented in Figure 6-13.

A useful technique for attribute identification is to note the parameters used by the engineer to describe each term in the knowledge transcripts. In the case of classes representing data types, the attributes can also be determined by looking at the data

parameters, e.g. the fields in a SCADA alarm. The types of data which will represent each attribute should also be noted, e.g. String, Float, Integer, etc. The class attributes and data type are indicated beneath each class in the class hierarchy as indicated in Figure 6-13.

By the end of this stage in the methodology, an ontology should have been created defining the key concepts within the application domain. This ontology will be referenced later in the methodology when considering agent interactions. Specification of the MAS can now progress to assessing the capabilities of legacy systems.

6.4.5 Stage 4 - Legacy System Reuse Potential

The task hierarchy may already be suggesting possible agents based on visual groupings of sub-tasks. However, the final agent task allocations cannot be confirmed until the capabilities of legacy decision support tools are considered and their potential for reuse within the MAS determined, since their reuse may constrain the allocation of tasks to agents. It is therefore necessary to include a stage where the legacy systems, their functional capabilities, their data requirements and their potential for reuse can be determined.

The first step is to identify from the requirements specification, captured knowledge and Use Case diagrams the available legacy systems and the capabilities currently utilised. Each system should then be taken in turn and its functional and data provision capabilities identified.

The functional capabilities should map directly onto tasks, some of which may be indicated within the task hierarchy as being required within the MAS. Others may not be required but should none the less be noted, as it may be beneficial to integrate these currently non-essential tasks into an agent. This would make available all the capabilities of the existing resource to the other agents within the MAS and any future additional agents thereby realising a truly extensible MAS.

At this stage in the methodology it also important to capture the data and control requirements of the legacy software and whether the original source code and

Application Program Interface (API) are available. This information will be used later to help determine whether legacy system reuse is feasible and the means of integrating the software into the agent. If available, software manuals or specifications are a useful source of information for this task.

Software Resource Assessment:	*Name of legacy software*				
Resource Description:	*Brief description of software functionality*				
Source Code Available?	*Y/N*	**API?**	*Y/N*	**Language:**	*Software Language*
Control Requirements	*List of the interactions with the user and other systems necessary to start and control software execution.*				

Functional Capabilities:		
Function	**Description**	**Possible Task Mapping**
Name of a software function	*Description of the software function*	*If applicable, identification of the associated task in the task hierarchy, which the legacy system could possibly perform providing integration is feasible.*
Name of a software function	*Description of the software function*	*If applicable, identification of the associated task in the task hierarchy, which the legacy system could possibly perform providing integration is feasible.*

Data Input Requirements:		
Data	**Mechanism**	**Ontology Mapping**
Data name	*How data is input / uploaded*	*Mapping of data to an ontology class*

Data Provision Capabilities:		
Data	**Mechanism**	**Ontology Mapping**
Data name	*How data is output / archived*	*Mapping of data to an ontology class*

Figure 6-14 Template for recording legacy system functionality

To provide a record of the legacy software assessment exercise, the template in Figure 6-14 should be used. Textual descriptions of the legacy systems control requirements, functional capabilities, data input requirements and data provision capabilities can all be entered.

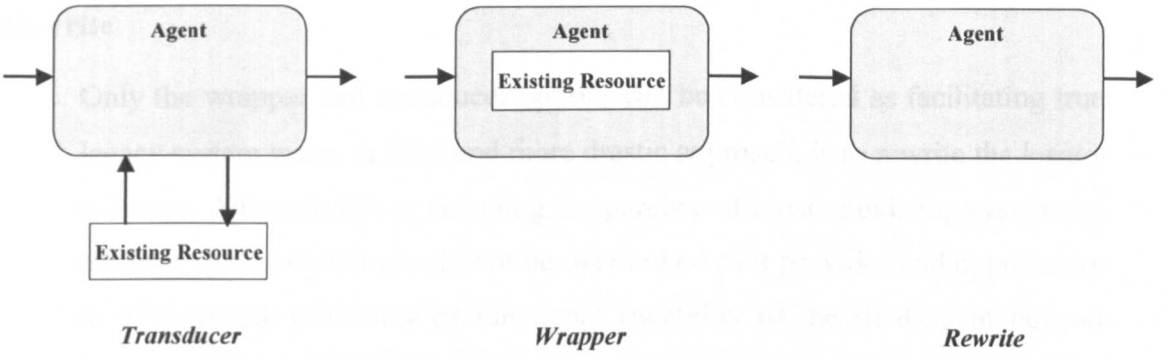


Figure 6-15 Legacy system integration alternatives

To assess the reuse potential of a legacy system it is first of all necessary to recognise the various integration alternatives available to the MAS developer. The alternatives are illustrated in Figure 6-15 [17] and are described below:

Transducer

A transducer can be implemented that mediates between a legacy system and other agents. The transducer accepts messages from other agents, translates them into the existing software's native communication protocol, and passes those messages to the software. It accepts the software's responses, translates into the Agent Communication Language (ACL) and ontology, and sends the resulting message on to other agents.

This approach has the advantage that it requires no knowledge of the software other than its communication behaviour and API. It is, therefore, especially useful for situations in which the software code is unavailable.

Wrapper

To implement a wrapper, code is added to the software program to allow it to communicate in ACL. The wrapper can directly examine the data structures of the program and can modify those data structures. Furthermore, it may be possible to include calls out of the program so that it can take advantage of externally available information and services.

This approach has the advantage of greater efficiency than the transducer approach, since there is less serial communication. However, it requires availability of the legacy system software code.

Rewrite

Only the wrapper and transducer options can be considered as facilitating true legacy system reuse. A third and more drastic approach, is to rewrite the legacy software. Although this is defeating the purpose of reusing existing systems to perform agent tasks, it should not be overlooked as it provides and opportunity to enhance the efficiency or functional capability of the final agent beyond what would be possible in either the transducer or wrapping approaches.

Given the available integration alternatives, the decision on whether or not legacy system reuse is possible is dictated by a number of factors, the most pertinent being:

- **Availability of Source Code and API:** If neither the source code nor API were known, then integration within an agent would not be feasible. However, if the software code, API or both were available then legacy system integration may be possible using either a transducer or wrapper and the legacy system should be considered for reuse.
- **Software Language:** If the language in which the legacy software was originally coded was known, is no longer obsolete and is still supported, then, providing the original software was available, it may be possible to modify the existing software and integrate the legacy system within the agent. In such cases the legacy system should be considered for reuse.
- **Control Requirements:** If a significant amount of user interaction is required to control the legacy system and upload data, then it may prove difficult to effectively automate the system within an agent. The general rule of thumb is that the fewer user interfaces that need to be automated then the easier it will be to integrate a legacy system within an agent.

Using the legacy software assessment templates and by considering the factors which dictate reusability, the feasibility of legacy system reuse should have been determined. Furthermore, the most appropriate integration option should also have been identified during this stage in the methodology.

6.4.6 Stage 5 – Update Task Hierarchy

Before proceeding with the remaining stages of the methodology, it is necessary to update the task hierarchy with the information obtained during ontology modelling and the preceding assessment of legacy system reuse feasibility.

The first step is to update the task hierarchy with the classes of information exchanged by the interaction tasks. Each interaction task is taken in turn and the type of information exchanged identified from the knowledge transcripts. The ontological mapping of the identified information type to an ontological class is then obtained. Finally, the information classes exchanged are placed beside each interaction task with the symbol '$\Rightarrow$' indicating that the resource is a required input to the task and the symbol '$\Leftarrow$' indicating that the resource is an output of the task.

The next step is to group the tasks which are capable of being realised by reuse of legacy systems.

To ensure an agent has full autonomy and control over the execution of tasks, those tasks which can be performed through legacy system reuse, need to be assigned to an individual agent. All the tasks performed by the legacy system therefore need to be grouped together under a single high-level task which the agent can perform. To indicate the grouping of tasks capable of being performed through legacy system reuse, a box is placed around the tasks along with the name of the software which performs the tasks.

6.4.7 Stage 6 – Identify Required Agents

At this stage in the specification process, the task hierarchy will indicate the high-level task which the MAS must perform and the sub-tasks that must be assigned and executed by agents within the MAS to realise the high-level task. The next stage is to identify the required agents and to decide which agents best perform which tasks.

Given the array of legacy decision support tools commonly available for decision support, and the fact that reuse would be desirable over development of new systems, it is highly likely that many of the functional tasks in the task hierarchy can be

performed by legacy systems. The tasks capable of being performed through reuse of legacy systems will already have been grouped in the task hierarchy. These task groupings provide the first indication of the required agents.

It is logical to assign individual agents to the control and execution of each legacy system so that complete autonomy can be realised. If it is determined during stage five of the methodology that a legacy system will fulfil some, if not all, of the functional sub-tasks under a single higher level task, then an individual agent should be assigned the higher level task. This same agent should also be furnished with the functional and interaction sub-tasks, composing the assigned high-level task, which cannot be performed by the legacy system, since it is highly likely that these will relate to the automation and control of legacy system.

Although legacy systems will perform many of the tasks within a MAS for decision support, there will no doubt remain tasks that cannot be realised through legacy system reuse. Some of these tasks may already be associated with a legacy system and should have been assigned to an agent; others will remain to be assigned.

When assigning these remaining tasks to agents the efficiencies of the final MAS must be a consideration. It would be inefficient to have several agents performing tasks which each require access to a common data source, since each agent would need to establish its own communications with the data source. A more elegant approach would be to assign all tasks with a common data need to one agent, in this case one communication with the data source would be sufficient, providing more available communications bandwidth for other agents.

The information to base these decisions on should be found in the requirements specification, knowledge transcripts and Use Case diagrams or by examination of the ontological classes exchanged by interaction tasks. Factors to consider are:

- Access to Data Sources: The number of simultaneous access requests to a data source may be limited, so the number of agents requiring access should also be restricted by grouping tasks requiring the same data within one agent.
- Security Concerns: The data being processed by a set of tasks may be of a sensitive nature, e.g. network performance statistics. For security reasons, it may be necessary to limit data access to one agent.

- Reliability and Redundancy: It may be necessary to distribute tasks across agents, at the expense of efficiency, to ensure an acceptable level of decision support provision is maintained even in the event of failure of an agent.

Each agent will have a particular role within the MAS. This role should be a simple textual description of the functionality provided by the agent. It may be as simple as 'interpret condition monitoring data and provide plant condition information'. The role should be determined by considering the high-level task and sub-tasks which the agent has been assigned and by referring back to the knowledge transcripts. Using the identified agent roles, simple agent names must be chosen to distinguish between the agents specified during the methodology.

To identify the agent task assignments it is useful to mark on the task hierarchy the high-level task performed by each agent. If this is done, as tasks are assigned to agents, it not only provides a useful indication of the sub-tasks assigned to each agent, but also identifies the tasks which remain to be assigned.

6.4.8 Stage 7 – Data and Information Exchange Mechanisms

So far the specification process has identified the interaction tasks each agent must perform and the ontological classes of data and information exchanged. To provide a more detailed specification, the mechanisms for achieving the provision and obtaining of data and information between agents must be identified.

Interaction tasks can only provide or obtain a data resource by sending and receiving a sequence of messages each structured using the ACL chosen for the MAS. This methodology supports the most common ACL, namely the FIPA ACL specification [18], which currently contains twenty-two different message types. Of these twenty-two messages, a number are directly related to information and data exchange, namely: 'subscribe', 'confirm', 'query-ref', 'inform', 'failure', 'refuse', 'agree' and 'request'.

The primary mechanism for obtaining data or information is either through subscribing for a regular update of new resources (sending a *subscribe* message), querying for a specific instance or set of resources (sending a *query-ref* message) or

requesting generation of a resource through task execution (sending a *request* message). The most appropriate mechanism for obtaining the data resource is determined by considering how often and for what purpose the resource is required.

An interaction task responsible for the provision of a data resource should also consider the permissible means of another agent obtaining the required resource. More often than not if an agent is capable of providing a resource via subscription, then provision via query-ref is not problematic. However, in many agents performing online functional tasks such as alarm processing, requesting the generation of a resource by execution of a task may cause undesirable delays in online processing so provision of a resource by request is not permissible.

Of the eight ACL messages directly related to information and data exchange the 'confirm', 'inform', 'failure', 'refuse' and 'agree' messages play an essential role in the message exchanges following the sending or receiving of a primary 'subscribe', 'query-ref' or 'request' message. Their role is not important at this stage in the methodology but will however be modelled during stage ten to facilitate the identification of the required message handlers during stage eleven of the methodology.

6.4.9 Stage 8 – Realising Agent Functionality

Having identified the required agents and their task assignments, the next stage is to determine the most appropriate means of realising the functional tasks assigned to each agent, which cannot be realised through legacy system reuse. The assigned functional tasks fall into two categories: primary decision support tasks and secondary decision support tasks.

Provision of decision support to power engineers, will often require systems capable of performing processor intensive functions, such as data interpretation, statistical analysis, signal processing, etc. Within a MAS for decision support, these demanding functions must be performed by the primary decision support functional tasks within each agent.

Many of the primary tasks, such as statistical analysis and signal processing, are common across a number of domains and have well documented software algorithms making them relatively simple to implement. However some may previously have been performed by an engineer and rely on the engineers' knowledge, experience and reasoning ability to perform the task. Furthermore, these tasks may be domain specific and a generic solution is neither available nor desirable. In this case intelligent reasoning techniques should be used since software algorithms may not be capable of implementing the reasoning ability.

There a number of techniques available with the choice of technique being largely dependent on the type of input data, the availability and format of reasoning knowledge and the type of output required. The most commonly used techniques are knowledge based systems, case based systems and model based systems. A detailed description of these techniques and their application suitability can be found in section 3.2.1 of chapter 3.

In addition to the primary decision support tasks, agents may have been assigned associated secondary tasks. Although not performing any demanding decision support function, such as data interpretation, these secondary tasks are still essential as they provide the functionality necessary to provide automated decision support. For example, parsing data ready for interpretation or scheduling and prioritising decision support based on received information.

When considering how to implement such secondary tasks it is important to consider how the task will be invoked and executed, as this will influence the decision on the most appropriate realisation method.

Using the knowledge transcripts and by consideration of when task execution would be required, it may become apparent that some tasks should be invoked when the agent is started and follow a repetitive and sequential task execution sequence. Examples of such tasks could be the regular polling of monitoring devices for new data or the scheduling of data for interpretation. The majority of these secondary tasks can be implemented as algorithms using traditional programming languages such as C++ and Java.

However, some tasks may be more reactive with the tasks' execution order being dictated by the timing and priority of requests and information received from other agents. It may be appropriate to implement such reactive tasks using rules.

On some occasions, it may even be necessary to implement a secondary decision support task using a combination of algorithms and rules. This would be appropriate when the agent needs to reprioritise its tasks in response to a change in conditions, e.g. reprioritisation of data retrieval based on the receipt of new fault information.

6.4.10 Stage 9 - Agent Modelling

Thus far, the specification has identified the required agents, their task execution responsibilities, and the mechanisms for data and information exchange. To collate all this information, and model the abilities of each agent, the methodology uses agent templates. Agent templates are a common theme across many of the methodologies and they provide an ideal means of documenting the agents' task execution responsibilities and how they will be realised. The agent template used in this methodology is presented in Figure 6-16.

AGENT NAME:	Agent A	
AGENT ROLE:	The agent's decision support role in the MAS	
Functional Tasks:	**Task Realisation Method**	
Task A	Rules	
Task B	Rules, Algorithmic Code	
Task C	Name of Legacy system to be used	
Interaction Tasks:	**Interaction Type**	**Exchanged Resource**
Obtain Resource B	subscribe	Resource B
Provide Resource A	query-ref, request	Resource A

Figure 6-16 Agent Modelling Template

The agent template begins with the name of the agent being modelled and the agent's decision support role in the MAS – these are obtained during stage six of the methodology.

The functional tasks assigned to the agent are then listed along with the identified task realisation method. The feasibility of realisation via legacy system reuse would have been determined during stage five of the methodology. If legacy system reuse was deemed infeasible, appropriate means for task realisation should have been identified when specifying the agent functionality during stage seven, e.g. rules, algorithms, etc..

The interaction tasks performed by the agent are also listed along with the permissible interaction types and the exchanged ontological classes. This information is obtained during stages three and seven of the methodology. Note that the interactions types are the FIPA ACL performatives permissible for the interaction.

Before moving onto the interaction modelling stage of the methodology it is important to consider the flexibility and extensibility of the MAS.

To ensure flexibility and openness it has already been highlighted that agents should not have hard coded knowledge of the other agents within the MAS and the resources they provide. Instead, agents requiring provision of a resource should be able to query utility agents for the name and network location of agents which can provide the required resource. Having identified a suitable agent, interactions and resource exchange can proceed.

Although utility agents will normally be provided by the tool-kit used to implement the final specification, it is important to note their requirement as part of the specification.

Assuming, for the moment, that this methodology is only being used to specify one self contained agent community, where all agents have full communications access to other agents, then only two utility agents would be required, a:

- **Nameserver** agent to provide the network addresses of the agents registered as being present in the MAS.

- **Facilitator** agent to act as a 'yellow-pages' providing a list of the abilities provided by the registered agents.

However, in many cases the agent community may not be self-contained, with agents distributed across a large network area. Furthermore, there may also be a requirement for interactions between interacting communities of agents across different platforms. In such cases, it may be necessary to duplicate these utility agent pairs across the network, with both a Nameserver and Facilitator being placed at key network nodes and at the interface between the communities. This would avoid bottlenecks being established by the utility agents, allowing distribution of the registered network addresses and abilities across a number of utility agents.

The initial decision on the most appropriate use and number of utility agents is based on assessment of the requirements specification and knowledge transcripts. However, the number of utility agents is largely dependent on the configuration of the MAS when the agents are deployed on a network, and should therefore be left to final system implementation. Given that this methodology is focussing on specification of the MAS, it is sufficient to merely highlight the need for utility agents.

6.4.11 Stage 10 - Agent Interactions Modelling

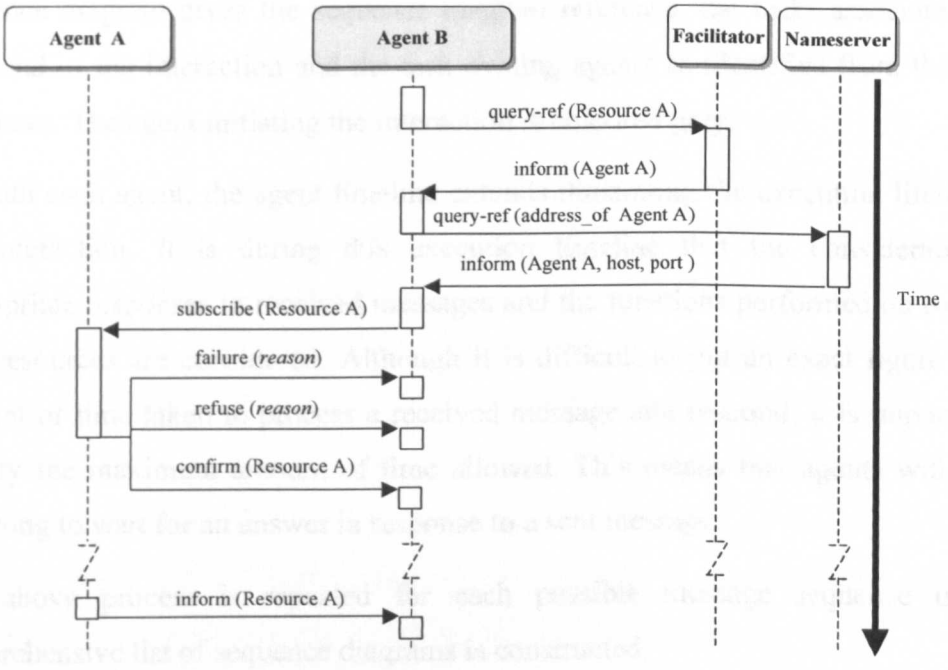
Regardless of the domain to which MAS are applied, an essential characteristic of MAS is the ability of the individual agents to interact. This is especially true within power engineering decision support, the domain to which this methodology is aimed.

Many of the tasks engineers perform to generate information useful for decision support can be classed as data and information collation. It is more than likely that the MAS being specified using this methodology will have to automate some of the data and information collation processes conducted by an engineer. The reasoning knowledge captured during 'walk through' of the case studies in stage one of the methodology will have identified the occasions when this is required, the software resources involved and the data and information resources exchanged.

To automate the collation process, the agents within the MAS will need to interact and exchange ontological classes representing data and information resources. Each

agent involved in the interaction will require an interaction task, or set of interaction tasks, to manage its end of the interaction and these will have already been identified during task decomposition and documented in the agent templates.

Automation of the collation process will only be achieved through each end of the interaction sending and receiving messages constructed using the ACL chosen for the MAS. This stage of the methodology models these message sequences so the required agent message handlers for each end of the interaction can be established during stage eleven of the methodology.



Sequence Diagram	SD_PEDA_xx: Subscribe for Resource A updates		
Task Owner(s)	Agent B	Initiating Task	Obtain Resource A
Task Owner(s)	Agent A	Responding Task	Provide Resource A
Other participants	NameServer	Responding Task	Provide Address
	Facilitator	Responding Task	Provide Abilities

Figure 6-17 Example sequence diagram

The first step is to determine the collation processes which the MAS is required to automate. These should have already been identified during task decomposition from the captured reasoning knowledge.

Each collation process is taken in turn and the sequence of exchanges between agents noted. The function an agent performs on the content of each received message is also noted, as this will play an important part in determining the time to allow for the receiving agent to execute the next stage in the interaction sequence.

Having listed all the required exchange sequences, the next step is to model these message exchanges using ***sequence diagrams***. An example of the sequence diagrams used in the methodology is presented in Figure 6-17.

The sequence diagram in Figure 6-17 depicts the sequence of messages required for Agent B to subscribe for automatic updates of Resource A. The table beneath the sequence diagram gives the sequence diagram reference, the tasks associated with each end of the interaction and the task owning agents as identified from the agent templates. The agent initiating the interaction is coloured grey.

Beneath each agent, the agent timeline extends illustrating the execution lifetime of the interaction. It is during this execution timeline that the consideration of appropriate responses to received messages and the functions performed on received data resources are considered. Although it is difficult to put an exact figure to the amount of time taken to process a received message and respond, it is important to specify the maximum amount of time allowed. This means that agents will know how long to wait for an answer in response to a sent message.

The above process is repeated for each possible message sequence until a comprehensive list of sequence diagrams is constructed.

6.4.12 Stage 11 - Agent Behaviour Functions

So far the methodology has specified the required agents, the functional and interaction tasks they must perform and how each agent will control its task execution and interactions with other agents. The Agent Functionality and Agent Modelling stages have also identified appropriate means, legacy system or otherwise, for realising the functional tasks assigned to each agent. The only remaining task is to specify the behavioural aspects of each agent. At this stage the properties necessary for legacy systems to behave as an intelligent agent are specified. The

properties software must possess are: autonomy, reactivity, pro-activeness and social ability: the weak notions of agency [19]. Before the agent can exhibit autonomy, reactivity and pro-activeness, it must be provided with a social ability. This is achieved through the use of the ACL, ontology and message handlers.

The required message handlers are identified from the sequence diagrams created during the Agent Interactions Modelling stage and will be implemented within the final agents as rules which monitor the agent's incoming mailbox for messages. When a new message is received, the agent should react by firing the rule appropriate to the received message and performing a function, thereby providing the agent with reactivity. Several message handlers may need to fire and functions be performed in a particular sequence before a particular interaction task is completed.

To provide agent autonomy an agent control function is necessary, which, at agent start-up, will create the message handlers essential for reactive and proactive behaviour and control the sequence and execution of the agent's interaction, primary and secondary decision support tasks. To design an agent control function the means by which primary and secondary decision support tasks are to be realised becomes paramount – these will have been determined during stage eight of the methodology.

Some tasks may have been identified as being purely reactive in nature, requiring rule-based implementation; others may require more stringent controls over their execution necessitating implementation as algorithms. To accommodate both these approaches, an agent control function requires both algorithmic and inference reasoning layers. To enable the eventual implementation of an agent control function where the tasks reside within these layers must be specified for each agent. An agent control diagram is a useful diagrammatical aid during this process.

As illustrated in Figure 6-18, an agent control diagram is split into an inference layer and an algorithmic layer. At the top of the diagram the set of tasks invoked upon the agent starting should be identified beginning with the inclusion of the 'Register location' and 'Provide abilities' tasks.

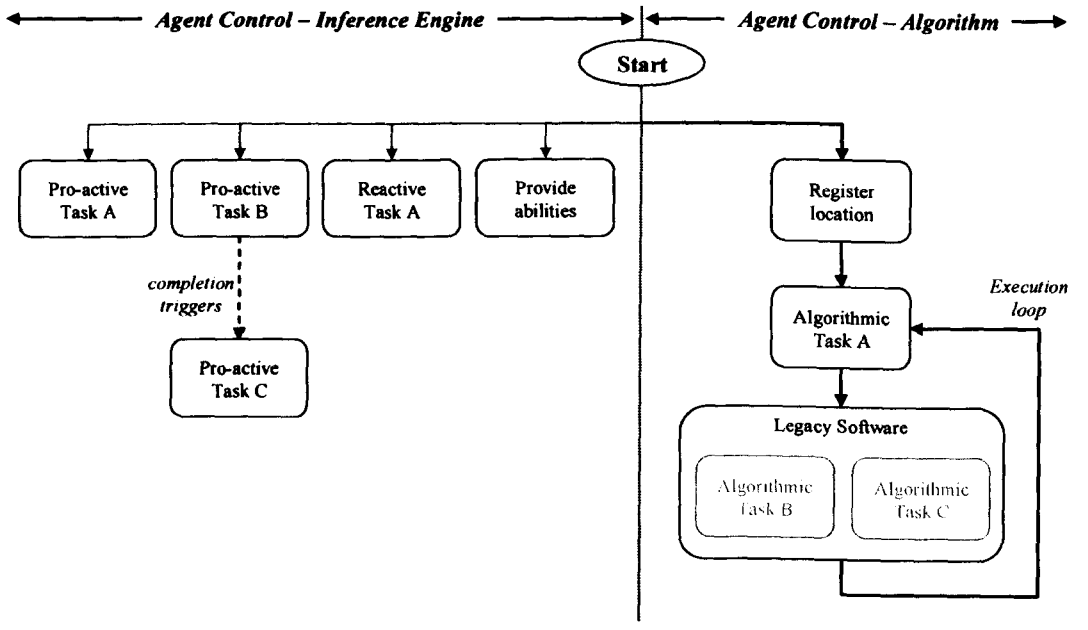


Figure 6-18 Agent control diagram

The 'Register location' task is essential as it is the only means by which the agent can register with the Nameserver agent and register its presence within the MAS. It must, therefore, be executed as soon as the agent starts. Once successfully registered, agents could start querying the agent for information on its abilities. The next task to be included in the agent control diagram should, therefore, be the reactive 'Provide abilities' task. Residing within the agent's inference layer this task will use message handlers to monitor for and respond to requests from other agents for information on the agents abilities. Design of the agent control function should now progress onto identifying the tasks which must run in the algorithmic layer following execution of the 'Register location' task.

The agent modelling templates created during stage nine should now be used to indicate the agent tasks which are to be performed within the agents' algorithmic reasoning layer, i.e. those functional tasks realised through reuse of legacy systems, intelligent reasoning techniques and algorithms. Some tasks may only be required to perform a configuration function and should, therefore, be executed immediately following the completion of the 'Register location' task. Others, performing an essential role in the provision of automated decision support, may require the agent

control function to loop their execution in order to provide continuous data interpretation and decision support provision.

The agent modelling templates will also have specified the interaction tasks required in order for an agent to exhibit proactive and reactive behaviour. Each interaction task identified in an agent modelling template will require the use of message handlers and should, therefore, be placed within the inference layer in the agent control diagram. Note, that some pro-active behaviour may only be necessary following successful completion of another pro-active task, e.g. information on where a fault has occurred must be at first obtained before fault related data can be identified and retrieved. In such cases, the tasks responsible for this pro-active behaviour should be placed in the inference layer below that which will trigger its execution.

The inclusion of the secondary decision support tasks required for pro-active and reactive behaviour concludes the design of an agent control function. The resulting agent control diagrams provide a specification of the tasks which each agent is required to invoke and the task execution sequence which must be managed.

6.5 Chapter Summary

This chapter has presented a methodology for the specification of MAS for power engineering decision support. Existing methodologies have been reviewed and their suitability for implementing decision support architectures as MAS assessed.

This assessment found that existing methodologies paid very little, if any, attention to specifying the components of a MAS necessary to produce flexible and extensible decision support architectures. Critically, none of the methodologies addressed legacy system reuse, nor did they consider specification of the agent behaviour functions necessary to automate legacy systems as agents and provide for collaboration between agents. Furthermore, the methodologies were lacking in their coverage of ontology specification and compliance with accepted FIPA communications standards.

Although similarities can be drawn between the new methodology and the methodologies reviewed in sections 6.3.1 to 6.3.4, the new methodology includes additional stages specifically aimed at addressing the shortcomings of existing methodologies in the field of decision support. Stages such as the 'Ontology Design', 'Legacy System Reuse Potential' and 'Agent Behaviour Functions' are just some of the features of the new methodology which distinguish it from others. By sequentially completing each stage in the methodology, a developer will create templates, textual descriptions and graphical representations of the main components, necessary to proceed with implementation of a MAS for decision support within the power industry.

The next chapter describes the application of this methodology during the development of the Protection Engineering Diagnostic Agents (PEDA) MAS for automating post-fault disturbance analysis.

6.6 References

[1]. K.P. Sycara, "Multiagent Systems", *AI Magazine*, v19, n2, 1998, pp 79-92.

[2]. C.A. Iglesias, M. Garijo, J.C. Gonzalez, "A Survey of Agent-Oriented Methodologies", *Proc. 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures and Languages (ATAL-98)*, 1998.

[3]. G. Bush, et al, "The Styx Agent Methodology", *The Information Science Discussion Paper Series*, n 2001/02, January 2001, ISSN: 1172-6024.

[4]. J.C. Collis, D.T. Ndumu, H.S. Nwana, L.C. Lee, "Zeus agent building tool-kit", BT Technology Journal, v16, n3, pp 60-68, 1998

[5]. D. Kinny, M. Georgeff, A.Rao, "A Methodology and Modelling Technique for Systems of BDI Agents", *Lecture Notes in Artificial Intelligence*, v1038, Springer-Verlag, 1996, pp 56-71.

[6]. G. Caire, et al., "Agent Oriented Analysis Using MESSAGE/UML", *Proc. Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, Montreal, Canada, May 2001.

[7]. C.A. Iglesias, M. Garijo, J.C. Gonzalez, J.R. Velasco, "Analysis and Design of Multiagent Systems using MAS-CommonKADS", *Intelligent Agents IV: Agent Theories, Architectures and Languages*, Springer-Verlag, 1998, pp 313-326.

[8]. Schreiber, G, Akkermans, H, Anjwierden, "Knowledge Engineering and Management: The CommonKADS Methodology", MIT Press, 1999.

[9]. Priestley, M, "Practical Object-Oriented Design with UML", McGraw-Hill Publishing, 2000.

[10]. M. Wooldridge, N.R. Jennings, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design", *Journal of Autonomous Agents and Multi-Agent Systems*, v3, n3, 2000, pp 285-312.

[11]. F.M.T. Brazier, B.M. Dunin-Keplicz, N.R. Jennings, J. Treur, "DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework", *International Journal of Cooperative Information Systems*, v6, n1, 1997, pp 69-94.

[12]. N.R. Jennings, T. Wittig, "ARCHON: Theory and Practice", *Distributed Artificial Intelligence: Theory and Praxis*, 1992, pp 179-195.

[13]. M.F. Wood, S.A. DeLoach, "An Overview of the Multiagent Systems Engineering Methodology", *in Proc. of the 1st International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, pp 207-221, 10th June 2000.

[14]. J. DiLeo, T. Jacobs, S. DeLoach, "Integrating Ontologies into Multiagent Systems Engineering", *in Proc. of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS)*, 15-16 July, 2002, Bologna, Italy.

[15]. M. Uschold, M. Gruninger, "Ontologies: Principles, Methods and Applications", *Knowledge Engineering Review*, v11, n2, June 1996.

[16]. N.F. Noy, et al, "Ontology Development 101: A Guide to Creating Your First Ontology", *Knowledge Systems Laboratory Technical Report* KSL-01-05, 2001.

[17]. M.R. Genesereth, S.P. Ketchpel, "Software Agents", *Communications of the ACM*, v37, n3, pp 48-53, 1994.

[18]. FIPA Communicative Act Library Specification, XC00037H, **http://www.fipa.org/repository/index.html**

[19]. M. Wooldridge, et al, "Intelligent Agents: Theory and Practice", *The Knowledge Engineering Review*, v10, n2, 1995, pp 115-152.

Chapter 7: Design of a MAS for Post-Fault Disturbance Analysis

7.1 Chapter Overview

The following chapter outlines how the methodology proposed in chapter six has been used to design a MAS for automating post-fault disturbance analysis. The resulting multi-agent architecture is entitled Protection Engineering Diagnostic Agents (PEDA) and consists of four core functional agents in addition to the Nameserver and Facilitator utility agents.

This chapter will describe in detail how each stage of the methodology was used to create the PEDA specification. By the end of the chapter the reader should have gained a clear picture of the PEDA multi-agent architecture, its constituent components and the design and engineering decisions taken during its development.

7.2 Introduction

Chapter two has already described in detail the manual post-fault disturbance analysis task conducted by protection engineers. This post-fault disturbance analysis process, illustrated again in Figure 7-1, is followed to validate the operation of a protection scheme in response to a disturbance on the transmission network.

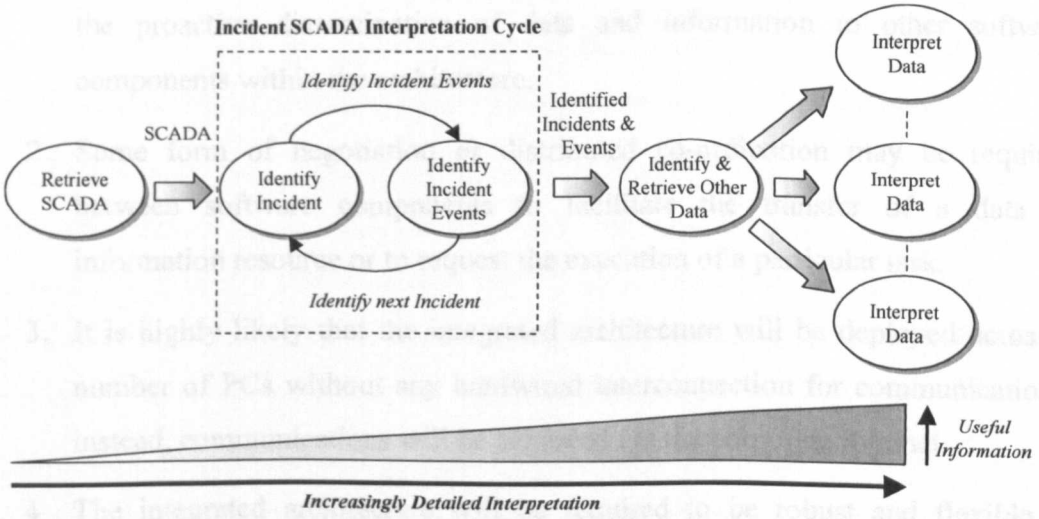


Figure 7-1 Manual post-fault disturbance analysis

The protection engineer uses a number of software tools, at different stages of disturbance analysis, to retrieve and interpret the disturbance data. The Telemetry Processor, described in chapter four, automates the retrieval and interpretation of SCADA data, identifying the incidents and pertinent events. Using the incident information, the protection engineer can then turn to proprietary fault record retrieval software to initiate the retrieval of incident fault records from DFRs. The retrieved fault records can then be interpreted using fault record interpretation software and be used in a protection validation toolkit to validate the protection scheme operation.

Each software tool operates either as a separate application on the same PC or as individual applications on dedicated PCs. The protection engineer must not only operate each software tool but also select the pertinent data and information produced or retrieved by the tool to be transferred to another tool for the next stage in disturbance analysis. The integration of these tools and the automation of the entire disturbance analysis process would lift this burden from the protection engineer.

Adopting an agent-based approach to achieve the automation of post-fault disturbance analysis through integration of the existing software tools is appropriate for a number of reasons:

1. Each software component within the integrated architecture will be required to exhibit a high degree of autonomy to manage its own tasks and to achieve the proactive dissemination of data and information to other software components within the architecture.

2. Some form of negotiation or distributed co-ordination may be required between software components to facilitate the transfer of a data or information resource or to request the execution of a particular task.

3. It is highly likely that the integrated architecture will be deployed across a number of PCs without any hardwired interconnection for communications, instead, communications will be achieved via the corporate Intranet.

4. The integrated architecture will be required to be robust and flexible to accommodate the temporary loss of a software component due to communications problems, the removal of an obsolete software component or the introduction of a new software component.

Given the appropriateness of an agent-based approach, the design of a multi-agent architecture for automating post-fault disturbance analysis commenced. The PEDA MAS was developed following the methodology presented in chapter six and the specification process is described in detail throughout the remainder of this chapter. Implementation of the specification will be described in chapter eight.

7.3 Requirements and Knowledge Capture

The first objective was to identify the desired PEDA functionality and to capture the knowledge required to realise this functionality within the agents. This was achieved by following the 'Requirements and Knowledge Capture' stage of the methodology – described in section 6.4.2.

7.3.1 Requirements Capture

Before commencing with PEDA development it was important to assign PEDA a global task which the PEDA agents would work together to perform. The assigned task was '**Automated Post-Fault Disturbance Analysis**' which was formally defined as:

> ***The automation of disturbance analysis retrieval and interpretation activities and the prioritisation of these activities to ensure the timely availability of decision support information to protection engineers.***

The global task was determined through informal discussions with protection engineers. However, to compile the more detailed requirements specification necessary to progress PEDA development a more structured and formal approach was required. The elicitation methodology adopted during Telemetry Processor development (section 4.4.1) and described in [1] was used.

A series of structured knowledge elicitation meetings were conducted with engineers experienced in performing post-fault disturbance analysis. The initial meetings focussed on scoping of the disturbance analysis task assigned to PEDA and on identifying suitable case studies for use during later knowledge elicitation meetings.

The later meetings focussed on identifying the knowledge and information necessary for the latter stages in the specification process.

No formal list of specific requirements was forthcoming during these scoping meetings, however five general requirements did emerge:

1. The existing software resources used by the engineer during manual disturbance analysis should be reused if at all possible. These are:
    - An online alarm processor – the Telemetry Processor.
    - Proprietary fault record retrieval software.
    - Offline fault record interpretation system.
    - A protection validation toolkit utilising Model Based Diagnosis.
2. Once configured, the autopolling facility provided by existing proprietary fault record retrieval software is static and does not allow for the prioritisation of fault record retrieval based on disturbances. The retrieval of fault records from DFRs on the circuit affected by a disturbance should be automated and receive highest priority.
3. The interpretation of fault records should be automated with interpretation priority being given to fault records related to the earliest disturbance.
4. As soon as possible after the disturbance the protection scheme operation should be validated and any identified discrepancies diagnosed.
5. The eventual PEDA architecture should facilitate the introduction of new software systems and removal of obsolete technologies without the requirement for extensive reengineering.

It is these five general requirements which form the basis of the PEDA requirements specification. The remainder of the specification was compiled from the activities, resource and reasoning knowledge elicited from the engineers at later meetings.

7.3.2 Knowledge Capture

The identified case studies covered a number of disturbance types ranging from simple circuit faults caused by a downed conductor to complicated double-circuit faults caused by lightning strikes. These case studies provided scenarios which were 'walked through' with the engineers thus enabling the disturbance analysis tasks they perform (activities knowledge), the software and data resources utilised (resource knowledge) and the knowledge used to operate the software resources, interpret and collate the data resources (reasoning knowledge) to be identified.

Capturing the activities knowledge and resource knowledge proved relatively straightforward. The elicitation of reasoning knowledge was however more time-consuming due to the need to identify three additional sub-sets of knowledge:

- **Software Control**: The engineering knowledge on how to operate the software resources used during disturbance analysis. This knowledge was especially important since existing software resources were required, where possible, to be integrated into PEDA as agents. These agents would need to have autonomous control over the execution of these legacy systems.
- **Interpretation**: The engineering knowledge used to interpret the data resources and information generated by legacy systems. This knowledge would, if required, be used to design the core functionality of data interpretation agents.
- **Data and Information Collation**: The engineering knowledge used to decide which data and information is pertinent to the particular disturbance analysis being conducted. In addition, in the case of multiple disturbances, the knowledge required to distinguish between and prioritise different disturbances. This knowledge would be used to implement agent behaviour.

The knowledge was initially captured and transcribed into separate knowledge transcripts. However, before the knowledge transcripts were finalised, the elicited knowledge was analysed again to determine whether a comprehensive coverage of the system-system and engineer-system interactions was present. It was found that, knowledge in this area was lacking, so additional knowledge elicitation meetings

were conducted where the specifics of interaction between systems and between a system and the engineer were captured. The importance of this interaction knowledge will be demonstrated later.

To assist both with validation of the elicited knowledge and with later stages in the design process the activities and software resource knowledge was modelled diagrammatically as Use Case diagrams. The Use Case diagrams illustrate software resource functionality and the functionality used by the engineer during disturbance analysis. Note that the Use Cases present a picture of all the functions used by the engineer during disturbance analysis and do not distinguish between activities. The elicited activities knowledge records the sequence of activities and the software functions used.

It should also be noted that a more detailed study of the functionality offered by each software resource is conducted later in the 'Legacy System Capabilities' stage of the applied methodology.

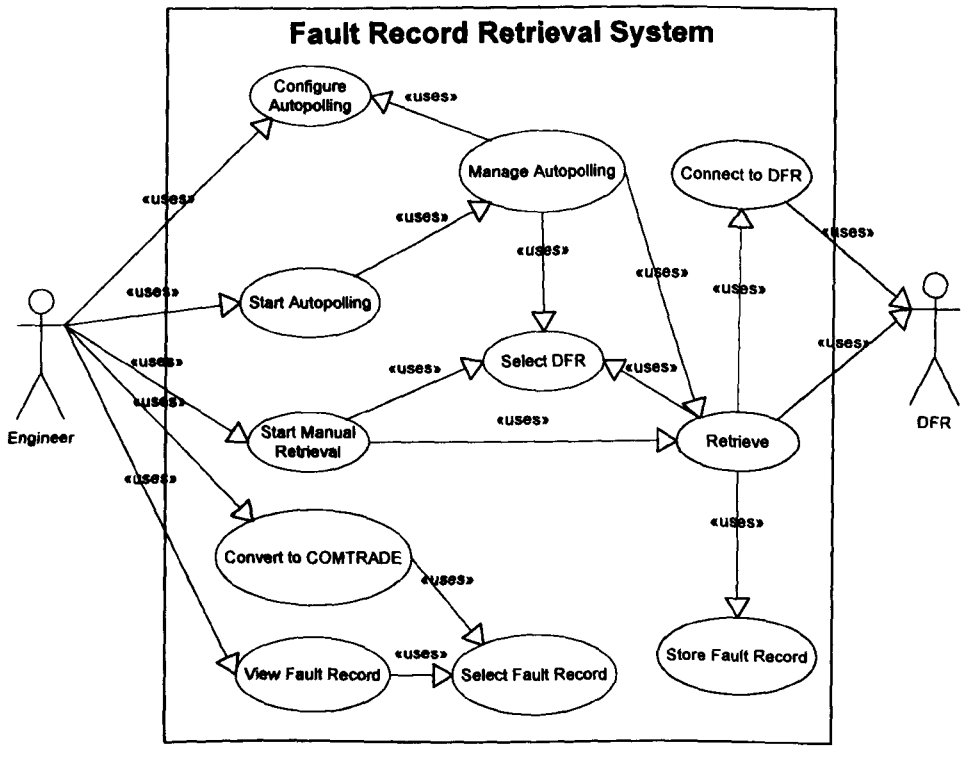


Figure 7-2 Fault Record Retrieval System Use Case diagram

A separate Use Case diagram was created for each of the identified software resources and is presented in Appendix B. Figure 7-2 presents a copy of the 'Fault Record Retrieval System' Use Case diagram as an example of the PEDA Use Cases.

An additional benefit provided by the Use Case diagrams is that they provide an ideal mechanism for validating that the captured activities and software resource knowledge was correct. The Use Case diagrams and knowledge transcripts were submitted to the engineers for validation and any errors or omissions corrected.

The knowledge captured during this process has not been included in the thesis, however the reasoning knowledge utilised will become apparent during explanation of later stages in the specification process.

7.4 Disturbance Analysis Task Decomposition

The intended primary role of PEDA was, and still is, to automate the overall post-fault disturbance analysis process currently conducted by engineers. The agents within PEDA would achieve this by coordinating the execution of the disturbance analysis activities, or tasks, and facilitating the exchange of data and information resources. Before the required agents and their task execution responsibilities could be identified the disturbance analysis task assigned to PEDA had to be decomposed into distinct sub-tasks. This was achieved by following stage two of the methodology 'Task Decomposition' – described in section 6.4.3.

The manual disturbance analysis process was already captured in the elicited activities knowledge and demonstrated using actual historical case studies. Using this elicited knowledge the decomposition of the disturbance diagnosis task was conducted using the decomposition criteria described in section 6.4.3, namely:

> *"further decomposition can be justified if the task meets any one of the following criteria:*
>
> - *The task required operation, or access to, several separate systems.*
> - *The task uses different data types each requiring a dedicated retrieval or analysis mechanism.*

- *The task uses several types of knowledge, e.g. rules, cases and models."*

Using the task decomposition criteria, the first layer of sub-tasks were identified as 'Identify Incidents and Events', 'Retrieve Fault Records', 'Interpret Fault Records', and 'Validate Protection Performance'. It was clear from the elicited knowledge that each of these sub-tasks not only required access to different software resources but also used differing types of knowledge, e.g. 'Identify Incidents and Events' used rules to interpret SCADA whereas 'Validate Protection Performance' used models.

Each of these sub-tasks was taken in turn and the elicited knowledge further analysed to determine if they could decompose further. In each case, further decomposition was possible and a second layer of sub-tasks was identified. This decomposition process continued until no further information could be gleaned from the knowledge.

As mentioned earlier in section 7.3, the initial knowledge capture process had not captured effectively the interactions between systems and between the engineer and systems. To illustrate this problem, consider the requirement for prioritised retrieval of disturbance related fault records.

The initial transcribed knowledge indicated that incident information must be obtained to locate the faulted circuit and identify the DFRs to prioritise retrieval from. This activity is represented as the 'Obtain identified incidents' task within the task hierarchy. However the provision of incident information was not explicit in the initial knowledge transcripts, since the engineer obtained this information from the Telemetry Processor. Analysis of the initial knowledge transcripts highlighted this omission, and more detailed knowledge elicitation meetings were conducted. As a result, the additional knowledge indicated the requirement for an additional sub-task to the 'Identify Incidents and Events' task to 'Provide Incidents'.

The resulting task hierarchy for PEDA is presented in Figure 7-3 with all interaction tasks indicated by a * symbol.

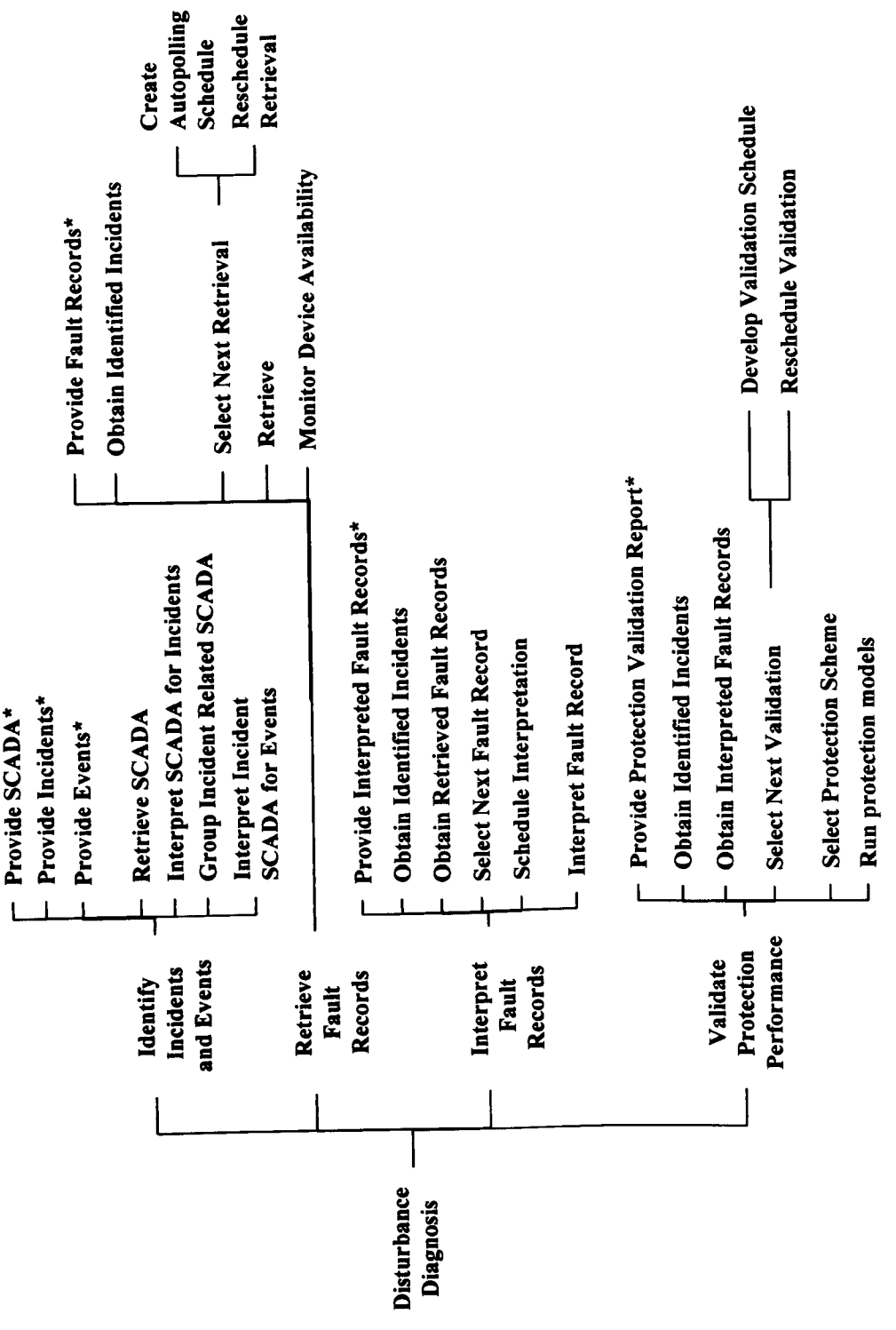


Figure 7-3 PEDA task hierarchy

7.5 An Ontology for Post-Fault Disturbance Analysis

So far the elicited data and information resource knowledge had only been described textually in the knowledge transcripts. This stage in the design process focussed on the modelling of the knowledge so an ontology for post-fault disturbance analysis could be created. This ontology would provide the vocabulary for information and data exchange between PEDA agents.

Before proceeding with ontology design a literature search was conducted to ascertain whether a suitable ontology already existed which could be reused or adapted for use in PEDA. An ontology for fault diagnosis was found [2], however it viewed fault diagnosis from the perspective of the control engineer. Furthermore it did not encompass all the terms used by the protection engineers to describe disturbance analysis. It was therefore deemed inappropriate and design of the post-fault disturbance analysis ontology for PEDA commenced.

The knowledge transcripts and task hierarchy were analysed and every term used during the description of a disturbance analysis activity, resource or referred to during walk through of a case study were noted. Terms such as 'Incident', 'SCADA', 'Circuit' and 'Transformer' were identified.

Having listed the terms, the next stage was to identify classes describing the terms. For example, the terms 'Incident', 'Event', 'Interpreted Fault Record' and 'Protection Validation Report' can all be grouped under a class 'Information' since they are all generated by existing intelligent systems. This process was repeated until all terms were assigned to classes and the classes organized into a class hierarchy as illustrated in Figure 7-4.

The classes enable the agents to provide and request particular types of resource. To facilitate requests for specific instances of a resource, such as a fault record from a particular DFR, the attributes of each class had to be defined. These were identified from a more detailed review of the data and information resource knowledge captured in the knowledge transcripts and by examining the actual historical data used in the case studies during capturing of the reasoning knowledge.

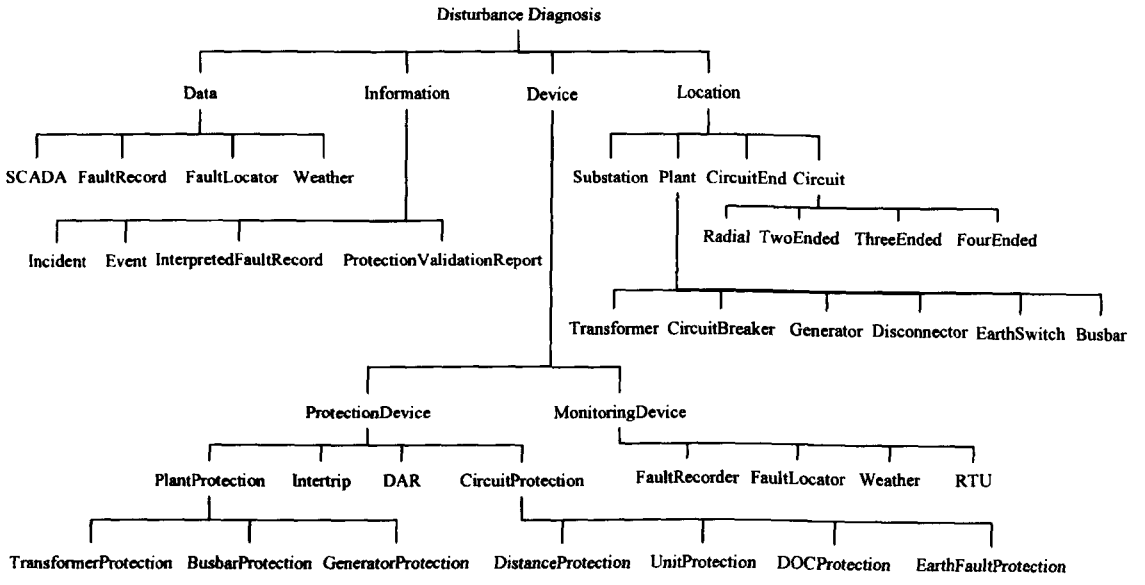


Figure 7-4 Post-fault disturbance analysis ontology class hierarchy – without attributes

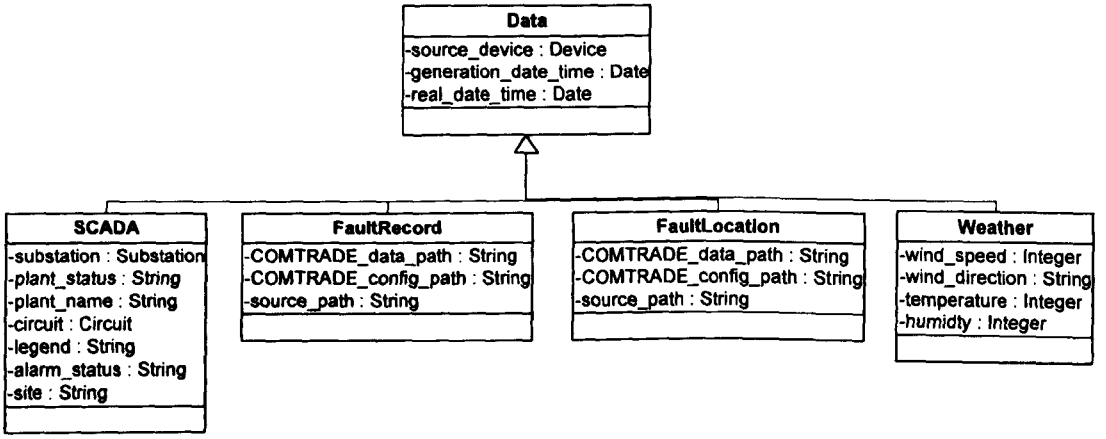


Figure 7-5 Extract from the post-fault disturbance analysis class hierarchy

Using the fault record data type as an example, it was clear from the knowledge transcripts that fault records were actually proprietary data files stored in a directory structure. Therefore one important attribute was the path to the original source data file: the attribute source_path in the FaultRecord class. It was also evident that, for the purposes of fault record interpretation and protection validation, each fault record needed to be stored using the COMTRADE format [3]. COMTRADE format requires both a .dat file to store each sample recorded in the record and a .cfg configuration file to enable the identification of channels in the .dat file. It was

therefore necessary to include additional COMTRADE_data_path and COMTRADE_config_path attributes in the FaultRecord class.

The complete disturbance diagnosis ontology including attributes can be found in Appendix C. An extract from the completed ontology is presented in Figure 7-5.

7.6 Possibilities of Legacy System Reuse

The desire for reuse of legacy systems within PEDA was stated in the requirements specification. If possible, the agents within PEDA should reuse the data retrieval and interpretation functions provided by these software resources to perform the tasks identified in the task hierarchy.

To determine the task execution capabilities of legacy systems, the requirements specification, captured knowledge and Use Case diagrams were analysed and the available functionality noted. Mapping of every function to a disturbance analysis task was not possible since the legacy systems had functions which, although used during manual disturbance analysis, did not have any role in the automation of post-fault disturbance analysis – *the global task assigned to PEDA*. However, these functions were noted as they may become useful in later versions of PEDA.

Having identified the tasks performed by existing software, the availability of the software for modification, its API and its control and data requirements had to be ascertained. This information was obtained through familiarisation with the software and access to available software documentation.

To capture the required information in a structured manner, the software assessment template introduced in section 6.4.5 was used for each legacy system. The templates generated for each of the software resources requiring reuse in PEDA can be found in Appendix D.

Having captured all legacy system capabilities in the software assessment templates, the PEDA specification process moved onto determining whether legacy system reuse was possible and, if so, identifying which of the three integration alternatives would be best suited for realising reuse of each legacy system.

As outlined in section 6.4.5 the decision on whether legacy system reuse is feasible is based on: the availability of source code and API, the software language used and the control and data requirements. Using the legacy software assessment templates (see Appendix D) the following decisions were reached with regards the reuse feasibility and selection of appropriate integration technique for each of the available legacy systems:

7.6.1 Telemetry Processor Decision Support System

The availability of the source code, API and the use of Java made this system an ideal candidate for reuse. Furthermore, all of the functional tasks, except 'Search for Incidents', were required by the 'Identify Incidents and Events' task. There was no need to rewrite the Telemetry Processor software or develop a transducer since the availability of the source code allowed for use of a wrapper: the most efficient approach for agent integration of software. It was evident from the software assessment template in Appendix D.1 that the wrapper may need to perform a database query to obtain the Incident, Event and SCADA information that the IEI agent must provide.

7.6.2 Fault Record Retrieval Software

Although a number of functional tasks did map onto the 'Retrieve Fault Records' sub-tasks, the software was proprietary with neither the source code nor API being available. A wrapper was not suitable due the unavailability of source code, so the integration options under consideration were limited to either a transducer or a rewrite. The options were further restricted to a rewrite by the fact that there was no information on the API for the existing software thereby eliminating the transducer option.

7.6.3 Fault Record Interpretation Decision Support System

Although the software did require user intervention, the availability of the API and source code suggested that software reuse would be possible. Of the available

functional tasks, the tasks requiring user intervention could be ignored, as they did not map onto any tasks in the task hierarchy. This left the 'Interpret fault record' function, which did have a direct task mapping. Legacy system reuse was, therefore, feasible and would be limited to realising the 'Interpret Fault Record' task.

7.6.4 Protection Validation Toolkit (PV Toolkit)

Yet again, the availability of the API and source code facilitated software reuse. Similar to the fault record interpretation software, the tasks requiring user intervention did not have any mapping to the task hierarchy and could be ignored. However, both the 'Select protection scheme' and 'Validate protection performance' software functions could be reused to perform the 'Select Protection Scheme' and 'Run Protection Models' sub-tasks of 'Validate Protection Performance' in the task hierarchy. The availability of the source code made reuse of the Protection Validation Toolkit to perform these sub-tasks assigned to PVD possible. Yet again, the availability of the source code allowed for integration using a wrapper.

7.7 Task Hierarchy Update

The PEDA specification process had now reached a stage in the methodology, where the task hierarchy required updating with the information obtained during ontology modelling and assessment of legacy systems.

The task hierarchy was first updated with the classes of data and information exchanged by the interaction tasks (indicated by an * symbol on the task hierarchy). Each interaction task was taken in turn and the type of information exchanged identified from the knowledge transcripts. The ontological mapping of the identified data or information types to an ontological class was then obtained. Finally, each exchanged ontological class was placed beside the appropriate interaction task with the symbol '$\Rightarrow$' indicating that the resource is a required input to the task and the symbol '$\Leftarrow$' indicating that the resource is an output of the task.

The final amendments to the task hierarchy were to group the tasks capable of being realised through reuse of the legacy systems identified in the Use Case diagrams.

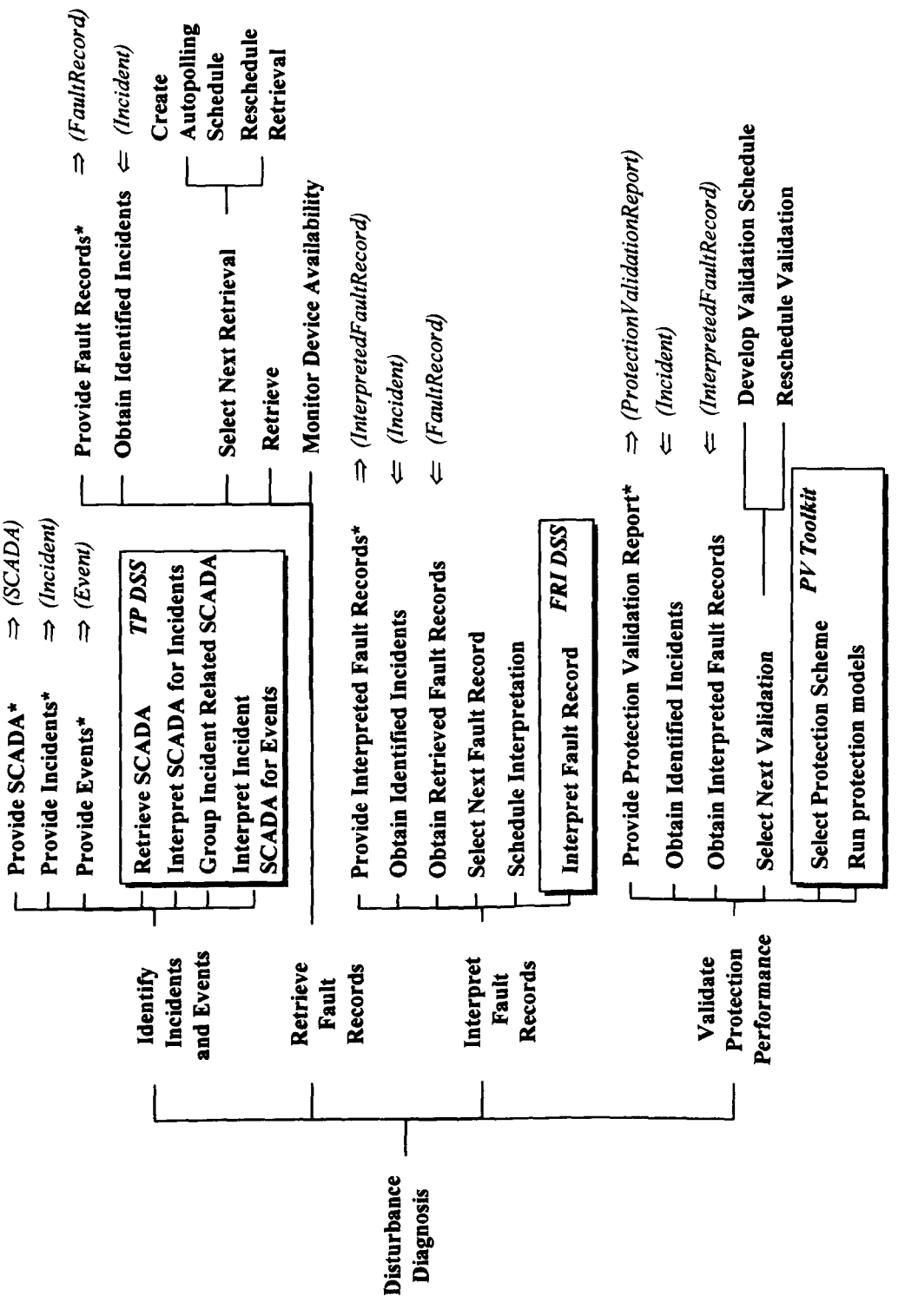


Figure 7-6 Task hierarchy updated with tasks performed by legacy systems and exchanged ontological classes

Given that all except the proprietary fault record retrieval software could be used to realise some, if not all, of the functional tasks in the task hierarchy, the functional tasks which could be realised by each legacy system were grouped under an abbreviated name for the legacy system, i.e. TP DSS (Telemetry Processor DSS), FRI DSS (Fault Record Interpretation DSS) and PV Toolkit (Protection Validation Toolkit). The remaining functional tasks would have their realisation methods identified during stage eight of the specification process.

The revised task hierarchy, generated during this stage in the PEDA specification process, showing the exchanged ontological classes and tasks that will be realised through legacy system reuse is presented in Figure 7-6.

7.8 Required Disturbance Diagnosis Agents

The objective of the next stage in the PEDA specification process was to identify the required agents and their task execution responsibilities. This was conducted by following stage six of the methodology described in section 6.4.7.

It was clear from the task hierarchy in Figure 7-6 that the disturbance analysis task assigned to PEDA could be broken down into four distinct root tasks, each with their own set of sub-tasks. In all but the 'Retrieve Fault Record' root task, it had already been determined that some sub-tasks would be achieved through reuse of legacy systems with others remaining to have a task realisation method identified.

For those functional tasks to be realised through legacy system reuse, it is logical to assign individual agents to the control and execution of the legacy systems so that complete agent autonomy can be realised. On this basis, the higher-level tasks encompassing each grouping of sub-tasks to be realised by a legacy system were assigned to individual agents. This identified the following agents:

- **Incident and Event Identification (IEI)** – responsible for control and execution of the Telemetry Processor legacy system.
- **Fault Record Interpretation (FRI)** – responsible for control and execution of the fault record interpretation software, specifically the fault record interpretation functionality.

- **Protection Validation and Diagnosis (PVD)** – responsible for control and execution of the protection validation toolkit, specifically the functionality to select protection schemes and validate protection performance.

The high-level tasks assigned to these agents each had a combination of interaction and functional sub-tasks, which were each to be realised through means other than legacy system reuse. In each case, these remaining sub-tasks were directly related to the automation and control of the legacy systems. It was, therefore, logical to assign them to the same agents that controlled the legacy systems.

Having assigned agents to control of the legacy systems and their associated sub-tasks, the PEDA design process turned to the 'Retrieve Fault Records' task and its sub-tasks, all of which remained to have their agent task assignments determined. Given that there were no constraints imposed by legacy systems, the efficiencies of the final MAS needed to be considered when determining the most appropriate assignment of these fault record retrieval tasks to agents.

Based on the requirements specification and knowledge transcripts, it was clear that the 'Retrieve Fault Records' task, and its sub-tasks, would need to utilise the existing communications infrastructure to access the remote DFR's and retrieve any generated fault records. The most efficient solution was, therefore, to assign the 'Retrieve Fault Records' task, and all its sub-tasks to one **Fault Record Retrieval (FRR)** agent. This way, one agent would handle fault record retrieval and communications management.

It is important to note that the creation of multiple fault record retrieval agents, each handling communications with individual DFRs was considered. However, the reliance on modems to dialup the DFRs and retrieve fault records did not lend itself to agent communications. If and when, ScottishPower PowerSystems connect all their substations to a wide area network, the number of agents assigned to fault record retrieval should be reconsidered.

This completed identification of the agents and their task assignments. The identified agents and their root tasks are illustrated in Figure 7-7.

Having identified the required PEDA agents and their task assignments, it was necessary to specify the role each agent will play in achieving the global task assigned to PEDA, namely: **Automated Post-Fault Disturbance Analysis.**

Each agent role was determined by considering, through analysis of its assigned tasks and by making reference to the requirements specification, what other PEDA agents and the engineers will ask of the agent and expect it to perform.

For example, the requirements specification stated that "*the interpretation of fault records should be automated with interpretation priority being given to the earliest disturbance*". The 'Interpret Fault Record' task has been assigned to the FRI agent, so it is the role of the FRI agent to meet this requirement. Furthermore, it is indicated in the task hierarchy that the PVD agent must 'Obtain Interpreted Fault Records', therefore FRI's role in PEDA must be extended to include provision of interpreted fault records to other agents.

This role identification process was conducted for each of the required agents and a textual description of the roles documented in Table 7-1.

Agent Name	Role within PEDA
Incident &Event Identification (IEI)	Automated interpretation of transmission SCADA alarms and the provision of SCADA data, and incident and event information to agents.
Fault Record Retrieval (FRR)	Automated and prioritised retrieval of fault records and the provision of fault records to agents.
Fault Record Interpretation (FRI)	Automated and prioritised interpretation of fault records and the provision of interpreted fault records to agents.
Protection Validation and Diagnosis (PVD)	Validation of protection performance and diagnosis of protection failures and the provision of protection validation reports to agents.

Table 7-1 PEDA Agent Roles

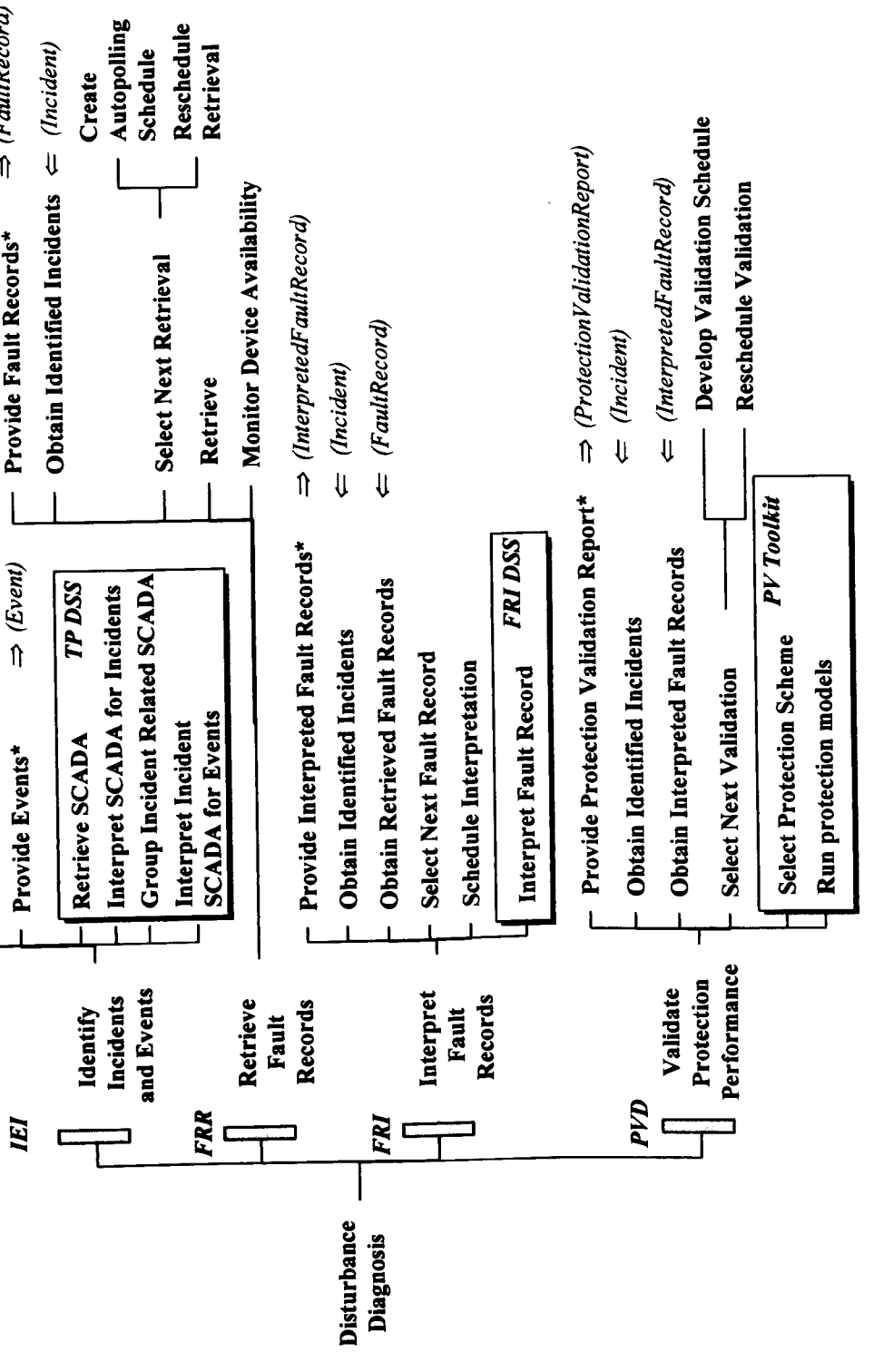


Figure 7-7 PEDA task hierarchy updated with agent task assignments

7.9 PEDA Data and Information Exchange

The next stage in the PEDA specification process was to detail the required data and information exchanges between agents and to select appropriate mechanisms for facilitating these interactions. This was conducted following stage seven of the methodology, as outlined in section 6.4.8.

7.9.1 IEI Data and Information Exchange

In essence, IEI is only providing the Telemetry Processor with the functionality necessary to behave as an agent within PEDA. This agent behaviour is required to enable the provision to other PEDA agents of the SCADA data retrieved by the Telemetry Processor and any generated incident and event information.

IEI can only provide this data and information to other agents as instances of the SCADA, Incident and Event ontological classes. IEI must, therefore, generate unique instances of these ontological classes for each received alarm, incident and event generated. This requirement would need to be realised during integration of the Telemetry Processor within the agent.

Given that IEI should be able to provide instances of the Incident, Event and SCADA ontological classes, the following FIPA ACL performatives were considered appropriate for providing this information:

- 'subscribe': The ability for agents to subscribe for automatic updates of Incident, Event and SCADA information was considered very important. IEI was the only 'real-time' window on what is happening on the network and other agents would need to be automatically informed of disturbances in order to prioritise their tasks.
- 'query-ref': Given that some agents may not require regular updates of Incident, Event and SCADA information, the facility to query for particular instances of the available information was deemed appropriate.

Note that, due to the online nature of the Telemetry Processor, IEI cannot generate Incident, Event or SCADA information at another agent's request. Therefore, the provision of a 'request' facility was not deemed appropriate.

7.9.2 FRR Data and Information Exchange

As identified in its role description, the primary function of FRR is to automate the retrieval of fault records from DFRs. In addition, other PEDA agents must be able to obtain these retrieved fault records from FRR when required. The obtaining of fault records will be via messages constructed using the FIPA ACL and with the message content specifying an instance of the FaultRecord ontological class.

To enable FRR to determine which of the retrieved fault records are required by the other agents, FRR must create unique instances of the FaultRecord ontological class for each retrieved fault record, populating the attributes of FaultRecord with the details of the retrieved fault record.

To provide timely retrieval of disturbance related fault records, it was also noted that FRR must prioritise fault record retrieval based on knowledge of what is happening on the power system. Furthermore, to identify the DFRs most likely to contain fault records directly related to the disturbance, FRR would require knowledge of the circuit affected by the disturbance and the disturbance time window. Such information is only available from IEI as ontological classes of the form Incident.

Given that FRR is required to prioritise fault record retrieval based on this information, the easiest way to obtain the Incident information was deemed to be via subscription to IEI, using the 'subscribe' performative available in the FIPA ACL. FRR would, therefore, be automatically updated with incident information as it becomes available, allowing FRR to concentrate on automated fault record retrieval until prioritised retrieval is required.

The final step was to determine the mechanisms by which other agents could obtain fault records. The following FIPA ACL performatives were deemed appropriate:

- 'subscribe': The provision of a FaultRecord subscription capability was desirable so agents could obtain automatic updates of retrieved fault records.

- 'query-ref': Using this performative, agents would be able to specify the fault records required by defining values for the attributes of the FaultRecord class, e.g. by specifying a substation FRR would only respond with fault records which have been generated by DFRs at the specified substation.
- 'request': Using this performative, agents would be able to specify the DFR from which fault record retrieval is required by defining values for the attributes of the FaultRecorder class, e.g. by specifying the DFR name, FRR could initiate fault record retrieval and inform the requesting agent of any retrieved fault records.

7.9.3 FRI Data and Information Exchange

As identified in its role description, FRI is responsible for fault record interpretation and will use the legacy fault record interpretation software to generate text files containing the interpretation results, e.g. fault type, faulted phases, fault clearance time, etc. To allow other agents to obtain these interpreted fault records, it was recognised that FRI must convert each textual interpreted fault record, generated by the legacy software, to instances of the InterpretedFaultRecord ontological class.

Within PEDA FRI will only be able to obtain fault records from the FRR agent. Section 7.9.2 has already described the mechanisms by which FRR will be able to provide fault records. The mechanisms that FRI will use to obtain fault records now had to be decided upon.

As identified in its role description, FRI is required to automate fault record interpretation. In essence, this means that it must interpret every fault record retrieved by FRR. The most efficient way of achieving this was deemed to be FRI subscribing to FRR for automated FaultRecord updates, using the 'subscribe' mechanism provided by FRR.

To provide timely interpretation of disturbance related fault records, it was also noted that, like FRR, FRI would require Incident information from IEI. Yet again, the easiest way to obtain the Incident information was deemed to be via subscription to IEI, using the 'subscribe' performative available in the FIPA ACL.

Although at this stage in the specification process, it is clear that FRR will prioritise fault record retrieval based on the same Incident information received by FRI, there is no guarantee that FRR will have received Incident information – there may have been a communications breakdown between IEI and FRR. Therefore, having been automatically informed of an incident by IEI, FRI must assume that FRR hasn't received the incident information.

To ensure prioritised fault record interpretation, FRI must be capable of sending 'query-ref' and 'request' messages to FRR. If, having sent a 'query-ref' message to FRR asking for any fault records relating to the disturbance, FRR returns no fault records, FRI must then 'request' fault record retrieval from the DFRs at each circuit end affected by the disturbance.

The final step was to determine the mechanisms by which other agents could obtain interpreted fault records. The following FIPA ACL performatives were deemed appropriate:

- 'subscribe': The provision of a InterpretedFaultRecord subscription capability was desirable so agents could obtain automatic updates of any interpreted fault records.
- 'query-ref': Using this performative, agents would be able to specify the interpreted fault records required by defining values for the attributes of the InterpretedFaultRecord class, e.g. by specifying a FaultRecorder and time frame FRI would respond with all the interpreted fault records generated from interpretation of fault records retrieved from the DFR within the time frame specified.
- 'request': Using this performative, agents would be able to specify the DFR from which fault record retrieval is required by defining values for the attributes of the FaultRecorder class, e.g. by specifying the DFR name, FRR could initiate fault record retrieval and inform the requesting agent of any retrieved fault records.

7.9.4 PVD Data and Information Exchange

As identified in its role description, PVD was required to validate the operation of protection schemes following a disturbance and diagnose any protection failures. The core reasoning would be achieved through reuse of the Protection Validation toolkit, which will generate protection validation reports. To allow other agents to obtain these reports, it was recognised that PVD must convert each report, generated by the toolkit, to instances of the ProtectionValidationReport ontological class.

To provide timely validation of a protection schemes operation in response to a disturbance, it was also noted that, like FRR and FRI, PVD would require Incident information from IEI. Yet again, the easiest way to obtain the Incident information was deemed to be via subscription to IEI, using the 'subscribe' performative available in the FIPA ACL.

Only having been informed of an incident, PVD should attempt to obtain the interpreted fault records associated with the incident and necessary for protection validation. Given this approach, subscription to FRI for automated fault record updates is not logical. A better approach was to adopt the same query and request process used by FRI to obtain incident related fault records from FRR, but this time use 'query-ref' and 'request' messages to obtain interpreted fault records from FRI.

Given that PVD must provide instances of the ProtectionValidationReport ontological class, the mechanisms by which other agents could obtain this information had to be identified. The following FIPA ACL performatives were deemed appropriate:

- 'subscribe': The provision of a ProtectionValidationReport subscription capability was desirable so agents could obtain automatic updates of any protection validation reports generated by PVD.
- 'query-ref': Using this performative, agents would be able to specify the protection validation reports required by defining values for the attributes of the ProtectionValidationReport class, e.g. by specifying a time window, PVD would respond with all reports generated within the time window.

It was considered that provision of a 'request' handling capability was not deemed appropriate in the initial version of PVD. This was due to the 'Incident' driven nature of PVD, i.e. having subscribed to IEI for incident information, PVD will automatically validate the protection performance for each identified incident. Furthermore, it is not possible for PVD to perform a protection validation at the request of another agent if no incident has occurred, since no fault records will be available.

7.10 Disturbance Analysis Functionality

At this stage in the PEDA specification process, the functional tasks each agent must perform to realise their disturbance analysis role in PEDA have been identified. In addition, those tasks that can be realised through the integration and automation of legacy systems within the PEDA agents have also been ascertained. The next challenge was to specify the means by which tasks that cannot be realised through legacy system reuse are to be implemented. This was conducted by following stage eight of the methodology described in section 6.4.9.

The process started with the primary decision support tasks assigned to each agent, i.e. those tasks performing the core reasoning necessary for disturbance analysis.

Given that primary decision support tasks are commonly processor intensive functions, and often rely on mirroring an engineers' knowledge, experience and reasoning ability through use of AI techniques, only three PEDA agents could be considered as having primary decision support tasks. These agents and their primary decision support tasks, all of which are to be realised using legacy systems, are listed in Table 7-2.

Although FRR had a number of functional tasks, none would require the encapsulation of an engineer's reasoning knowledge, use of an inference engine or processor intensive reasoning, e.g. the 'Retrieve' task was only required to dialup DFRs and retrieve fault records. The lack of primary decision support functions within FRR combined with the fact that all other agents were to realise their primary

decision support functions via legacy systems meant that the specification process could move onto the secondary decision support functions.

PEDA Agent	Legacy System	Primary Decision Support Tasks Realised Using Legacy Systems
IEI	Telemetry Processor	▪ Interpret SCADA for Incidents ▪ Group Incident Related SCADA ▪ Interpret Incident SCADA for Events
FRI	Fault Record Interpretation Engine	▪ Interpret Fault Record
PVD	Protection Validation Toolkit	▪ Run Protection Models

Table 7-2 Primary decision support tasks within PEDA agents

Before proceeding, it should be noted that, if legacy systems were not available for integration into IEI, FRI and PVD, the data being interpreted and available reasoning knowledge would need to have been assessed to identify the most appropriate reasoning techniques. Having identified the most appropriate reasoning techniques, the interpretation knowledge elicited during stage one of the specification process could have been used to design an appropriate reasoning engine.

The secondary decision support tasks assigned to each agent would allow realisation of the autonomy, pro-active, reactive and social behaviour essential for PEDA to achieve post-fault disturbance analysis. To determine the most appropriate means of implementing these secondary tasks, it was necessary to consider how each task will be invoked.

Following identification of the primary decision support tasks, it was clear from the task hierarchy that ten secondary decision support tasks remained to have their task realisation method identified. The ten tasks, their agent assignments and the chosen realisation method are presented in Table 7-3.

To describe how the software mechanism appropriate to realising a secondary decision support task was identified, the 'Select Next Fault Record' and 'Schedule Interpretation' tasks assigned to FRI will be used.

PEDA Agent	Secondary Decision Support Task	Chosen Realisation Method
FRR	Select Next Retrieval	Algorithmic Code
FRR	Create Autopolling Schedule	Algorithmic Code
FRR	Reschedule Retrieval	Rules, Algorithmic Code
FRR	Retrieve	Algorithmic Code
FRR	Monitor Device Availability	Algorithmic Code
FRI	Select Next Fault Record	Algorithmic Code
FRI	Schedule Interpretation	Rules, Algorithmic Code
PVD	Select Next Validation	Algorithmic Code
PVD	Develop Validation Schedule	Rules, Algorithmic Code
PVD	Reschedule Validation	Rules, Algorithmic Code

Table 7-3 Chosen secondary decision support tasks realisation methods

So far in the PEDA specification process, it was clear from the role descriptions that FRI must automate and prioritise fault record interpretation. Furthermore, consideration of the required data and information exchanges identified that prioritisation would be based on received incident information.

Fundamental to achieving its assigned role, FRI would be required to manage an interpretation schedule containing all the fault records requiring interpretation. The routing management of this schedule would require the 'Schedule Interpretation' task to add received fault records to the schedule as they are received – this could easily be achieved by algorithmic code. The 'Schedule Interpretation' task will, however, require a reactive element which will execute every time a fault record is received, adding the fault record to the schedule. This can be achieved using rules and an inference engine.

Given that the 'Schedule Interpretation' task would have prioritised the fault record interpretation, the 'Select Next Fault Record' task would only be required to select the fault record with highest interpretation priority from the schedule. This functionality can easily be achieved with algorithmic code.

7.11 Modelling of PEDA Agents

Agent modelling templates were created for each of the PEDA agents identified in Figure 7-7 and are available in Appendix E. Each template was compiled by collating the information identified during the previous stages of PEDA specification, namely: the agent role description, the functional tasks assigned to the agent, the decision on whether or not each functional task can be realised by a legacy system, the interaction tasks assigned to the agent, the permissible interaction types and the ontological classes exchanged by each interaction.

The requirements specification had indicated that the "*PEDA architecture should facilitate the introduction of new software systems and removal of obsolete technologies without the requirement for extensive reengineering*". To achieve this additional Nameserver and Facilitator utility agents were deemed essential.

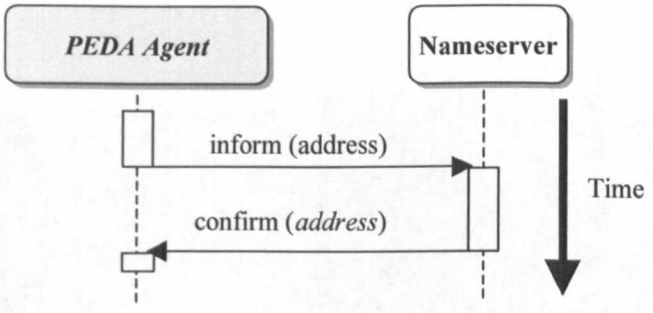
PEDA only needs four agents to perform the required disturbance diagnosis functionality. This combined with the likelihood that all these agents would be in close network proximity to each other (probably in a head office computer room and possibly even sharing the same PC's) makes for a self-contained agent community where only one instance of each utility agent is required. It wasn't necessary to model these utility agents, as they would be provided by the toolkit chosen to implement PEDA – described in the next chapter.

7.12 Specification of PEDA Agent Interactions

The next stage in the PEDA specification process was to model the agent interactions necessary to achieve automated disturbance analysis. The agent interactions modelling process was conducted following stage ten of the methodology as described in section 6.4.11 with the resultant diagrams illustrated in Appendix F.

The process commenced with the creation of sequence diagrams modelling the agent interactions with the PEDA utility agents. These interactions are the most basic of all the PEDA interactions and are essential for realising a flexible and extensible architecture where agents can discover the abilities and location of other agents.

It should be noted that these interactions had not been explicitly specified thus far. This is because they were not an essential part of the disturbance diagnosis task and are instead a requirement of the MAS architecture itself. Many MAS simply do without utility agents and hardcode the knowledge of other agents' abilities and locations into the agents within the MAS. Adopting this approach would still enable the disturbance analysis task to be achieved but would result in an inflexible and non-scalable architecture.

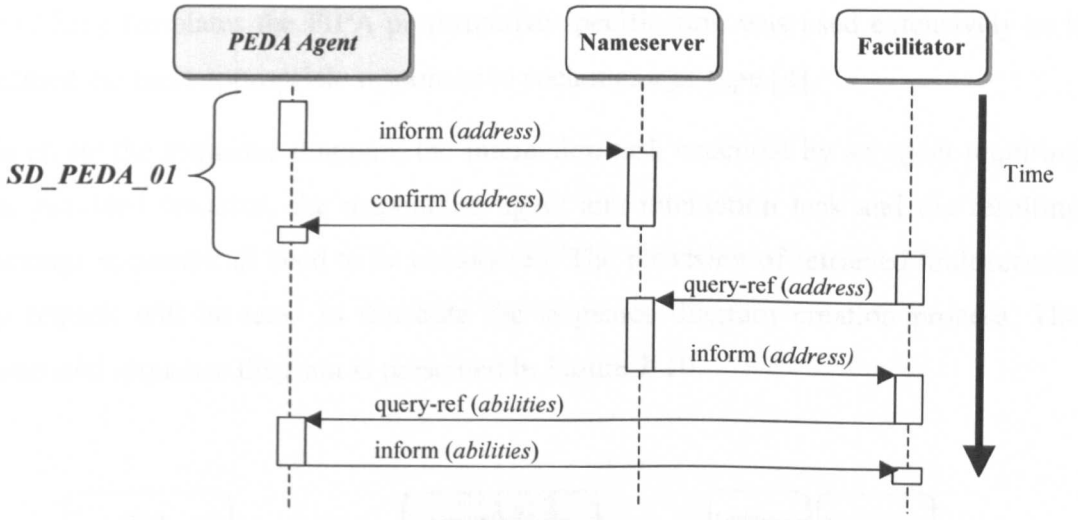


Sequence Diagram	SD_PEDA_01: Nameserver registration		
Task Owner(s)	Any PEDA agent	Initiating Task	*Register location*
Task Owner(s)	Nameserver	Responding Task	Acknowledge Registration
Other participants	None	Responding Task	N/A

Figure 7-8 PEDA Sequence Diagram: Nameserver registration

The most basic interaction with a utility agent is at start-up with the registering of the agents' location. Sequence diagram SD_PEDA_01 was created to model this message exchange and is presented in Figure 7-8.

Another interaction sequence fundamental to the running of flexible and extensible MAS is that initiated by the Facilitator to maintain an up-to-date record of the abilities each agent can offer. On a regular basis (time interval configured by the developer), the Facilitator requests an update from the Nameserver of the addresses of all the agents within the MAS. Using this information, the Facilitator then queries each agent about what abilities the agent can offer. This process is modelled in SD_PEDA_02 and is presented in Figure 7-9.



Sequence Diagram	SD_PEDA_02: Provide Abilities		
Task Owner(s)	Facilitator	**Initiating Task**	Request Abilities
Task Owner(s)	Any PEDA agent	**Responding Task**	*Provide Abilities*
Other participants	None	**Responding Task**	N/A

Figure 7-9 PEDA Sequence Diagram: Providing Abilities to Facilitator

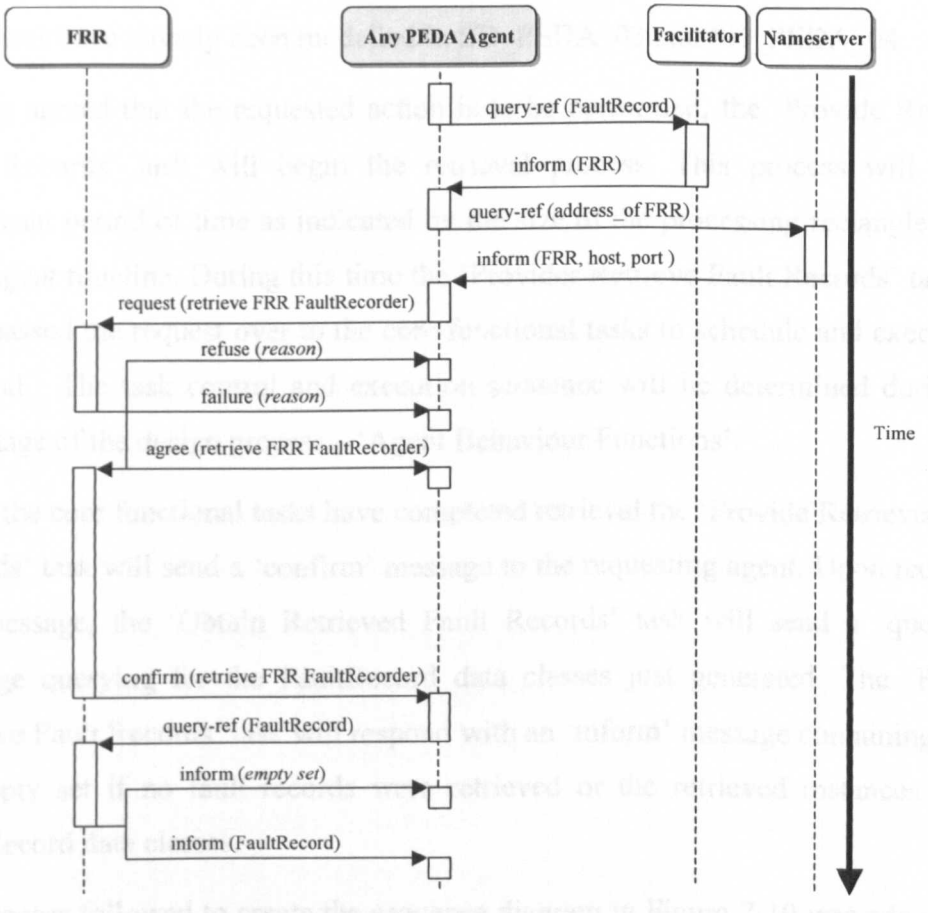
PEDA agents must also be able to query the utility agents to identify the name and location of agents capable of providing a desired resource. The sequence diagrams modelling the 'query for abilities' and 'query for address' interactions are presented in SD_PEDA_03 and SD_PEDA_04 respectively in Appendix F.

The agent modelling templates have described the mechanisms each individual agent will use to provide its data and information resources and specified the associated interaction tasks. To enable the later identification of the message handlers and control functionality necessary to manage these interactions, the sequence of messages required to obtain each desired resource were modelled.

The modelling process started by taking each agent in turn and identifying the interaction tasks that facilitate the provision of data and information resources to other agents, e.g. the 'Provide Incidents' task in IEI. The different interaction types for each task were then identified from the agent modelling templates and a separate sequence diagram was created for each interaction type. In addition to the agent

modelling templates the FIPA performative specification was used extensively as it defined the most appropriate responses to each message type [4].

To create the sequence diagram, the interaction task executed by an agent requiring the provided resource, the responding agent and interaction task and the resulting message sequence all need to be considered. The provision of retrieved fault records by request will be used to illustrate the sequence diagram creation process. The generated sequence diagram is presented in Figure 7-10.



Sequence Diagram	SD_PEDA_13: Request Retrieval of Fault Record(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Retrieved Fault Records
Task Owner(s)	FRR	**Responding Task**	Provide Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

Figure 7-10 PEDA Sequence Diagram: Request Retrieval of Fault Record(s)

The first step was to consider how a PEDA agent, such as FRI, would request the retrieval of fault records. It was clear from the FRI agent template that the 'Obtain Retrieved Fault Records' task would be used. This task would need to start by identifying a provider of the FaultRecord data class specified in the ontology. A 'query-ref' message with content FaultRecord would be sent to the Facilitator and the task would then wait for an inform message from the Facilitator in response to the 'query-ref'. The response would identify the FRR agent as a provider of the FaultRecord data class. A similar procedure would need to be conducted to query the Nameserver for the network location of the FRR agent. Note that these message sequences have already been modelled in SD_PEDA_03 and SD_PEDA_04.

Having agreed that the requested action is to be performed, the 'Provide Retrieved Fault Records' task will begin the retrieval process. This process will take a significant period of time as indicated by the size of the processing rectangle on the FRR agent timeline. During this time the 'Provider Retrieve Fault Records' task will have passed the request over to the core functional tasks to schedule and execute the retrieval. The task control and execution sequence will be determined during the next stage of the design process – 'Agent Behaviour Functions'.

When the core functional tasks have completed retrieval the 'Provide Retrieved Fault Records' task will send a 'confirm' message to the requesting agent. Upon receipt of this message, the 'Obtain Retrieved Fault Records' task will send a 'query-ref' message querying for the FaultRecord data classes just generated. The 'Provide Retrieve Fault Records' task will respond with an 'inform' message containing either an empty set if no fault records were retrieved or the retrieved instances of the FaultRecord data classes.

The process followed to create the sequence diagram in Figure 7-10 was adopted for each of the interaction tasks responsible for provision of a resource. Note that not all of the sequence diagrams in Appendix F relate to interactions which occur within the current version of PEDA, e.g. subscribe for Events. However these need to be modelled to ensure the agents can provide all resources in case future agents are introduced.

7.13 Required PEDA Agent Behaviour

The PEDA specification process had now reached perhaps the most significant, at least in agent terms, point where the behaviour of each agent had to be specified. Agent behaviour specification was conducted following stage eleven of the methodology as described in section 6.4.12.

There are two aspects to the specification of agent behaviour which will be described in detail in the sections to follow, namely:

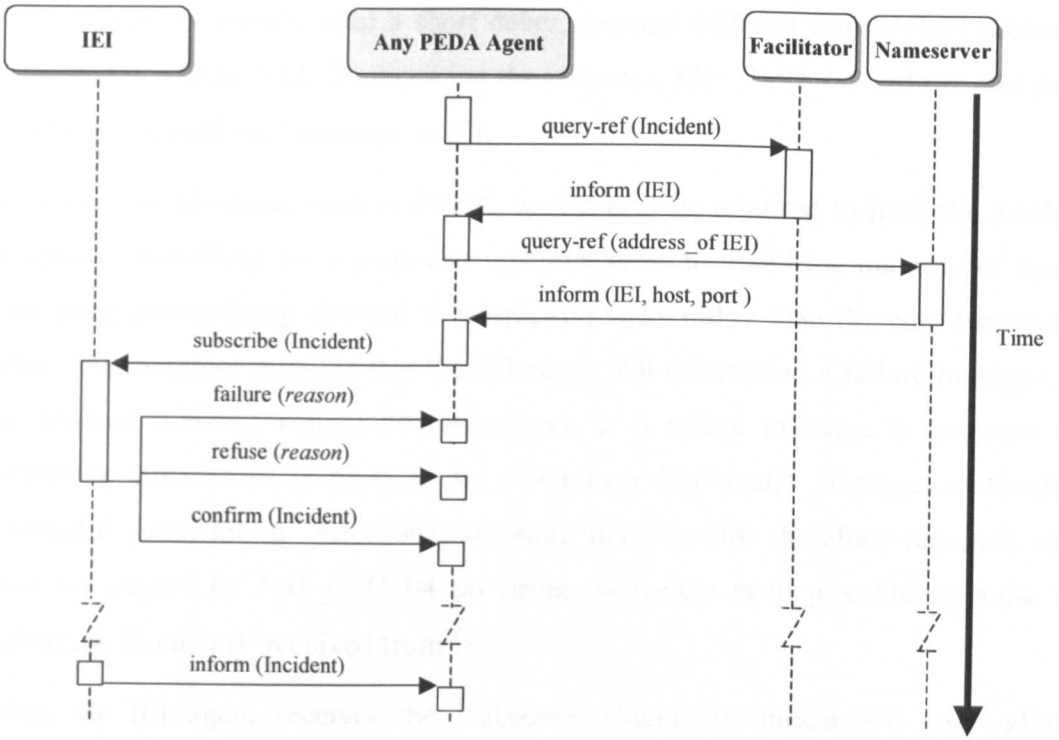
- Identification of the message handlers essential for pro-activeness, reactivity and social interactions.
- Specification of the agent control functions required for agent autonomy.

7.13.1 Message Handlers

When an agent's control function decides that it must interact with another agent it will exhibit proactive behaviour by constructing and sending a message to one or more agents. Message handlers must be present within the initiating agent to handle and react to the messages received in response to the initial message. These message handlers, essential for an agent's pro-active behaviour, must be identified.

The agents that will respond to a message sent pro-actively by another agent must also have message handlers to react appropriately to the received message. These message handlers, essential for an agent's reactive behaviour, must also be identified.

To illustrate how the message handlers required for both pro-active and reactive behaviour were identified consider an agent's subscription to IEI for Incident information, as illustrated in the sequence diagram presented in Figure 7-11. For the purposes of this illustration the 'Any PEDA Agent' will be PVD. The sequence of agent interactions will be walked through, and the message handlers required for each interaction identified, starting with the pro-active sending of the 'query-ref (Incident)' message by the PVD agent for Incident subscription. Note that the message handlers identified during this stage of the PEDA specification process and referred to in the subsequent text are documented in Appendix G.



Sequence Diagram	SD_PEDA_05: Subscribe for Incident updates		
Task Owner(s)	Any PEDA Agent	Initiating Task	Obtain Identified Incidents
Task Owner(s)	IEI	Responding Task	Provide Incidents
Other participants	NameServer	Responding Task	Provide Address
	Facilitator	Responding Task	Provide Abilities

Figure 7-11 Sequence diagram for Incident subscription

Before sending the 'query-ref (Incident)' message to the Facilitator, a message handler must be created within PVD to handle the Facilitators' response. The creation of the message handler will be managed by the PVD agent control function, the design of which will be described in the next section. Figure 7-12 presents the message handler required to handle the Facilitators' response, MH_PVD_03.

Having received, in the PVD agent's incoming mailbox, an inform message from the Facilitator containing IEI as an identified Incident provider, message handler MH_PVD_03 will fire and respond by constructing and sending the 'query-ref (address of IEI)' message to the Nameserver. Another message handler, MH_PVD_04 will also be required to handle the response from the Nameserver.

The Nameserver should, after a short delay, respond with the inform (IEI) message illustrated in Figure 7-12. On receiving the response, MH_PVD_04 will fire and send a 'subscribe (Incident)' message to IEI.

In an open architecture, such as PEDA, agents may be required to limit the number of agents subscribing for a particular resource so as to avoid the majority of agent processing power being devoted to interaction tasks rather than the core functional tasks. It is therefore possible that the IEI agent will either send a failure message, if the original message cannot be understood, or a refuse message in response to 'subscribe (Incident)' as opposed to a 'confirm (Incident)' message confirming successful subscription. Additional message handlers are therefore required, and must be created by MH_PVD_04 on firing, to handle each possible response to 'subscribe (Incident)' received from IEI.

When the IEI agent receives the 'subscribe (Incident)' message it will exhibit reactive behaviour by firing message handler MH_IEI_01 described in Figure 7-13. This message handler will be created by the IEI agent control function when the agent is started to provide the IEI agent with reactive behaviour. If subscription is successful, the MH_IEI_01 message handler will respond with a 'confirm (Incident)' message and create a subscription rule to monitor for new Incidents and automatically inform subscribed agents.

If Incident subscription has been successful, the PVD agent requires a message handler to receive future 'inform (Incident)' messages from IEI and add them to its memory. The required message handler, MH_PVD_08, is created when MH_PVD_07 fires and remains in the PVD agent's memory as long as the agent is running.

The process followed to create the message handler templates in Figures 7-12 and 7-13 was adopted for each of the PEDA agents. Each sequence diagram in Appendix F was walked through and the required reactive and pro-active message handlers documented in Appendix G. The message handler templates reference the sequence diagram and initiating or responding task in the task hierarchy that they relate to.

Agent	PVD	Behaviour	Proactive
Sequence Diagram	SD_PEDA_05 – Subscribe for Incident updates		
Initiating Task	Obtain Identified Incidents		

Message Handler ID	Incoming Message			Response
	Type	Content	In Reply To	
MH_PVD_03	inform	*agent name*	query-ref (Incident)	• Send query-ref (address_of *agent name*) to Nameserver • Create MH_PVD_04 to handle response
MH_ PVD_04	inform	*agent address*	query-ref (address_of *agent name*)	• Send subscribe (Incident) to *agent name* • Create MH_PVD_05, MH_PVD_06 & MH_PVD_07 to handle response.
MH_ PVD_05	failure	*reason*	subscribe (Incident)	• Log reason for failure and reattempt later
MH_ PVD_06	refuse	*reason*	subscribe (Incident)	• Log reason for refusal and reattempt later
MH_ PVD_07	confirm	Incident	subscribe (Incident)	• Log successful subscription • Create MH_PVD_08 to handle inform (Incident) messages • Terminate 'Obtain Identified Incidents' task
MH_ PVD_08	Inform	Incident	subscribe (Incident)	• Add Incident to memory

Figure 7-12 Proactive message handlers required by PVD 'Obtain Identified Incidents' task

Agent	IEI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_05 – Subscribe for Incident updates		
Responding Task	Provide Incidents		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_IEI_01	subscribe	Incident	• If subscribe message cannot be understood, then o Send failure (*reason*) to sending agent • If subscribing agent does not have correct access privileges, then o Send refuse (*reason*) to sending agent • If subscribing agent message can be understood and subscribing agent does have correct access privileges, then o Send confirm (Incident) to subscribing agent o Create a 'subscription rule' to monitor for new Incident data and automatically inform subscribed agent

Figure 7-13 Reactive message handler required by IEI 'Provide Incidents' task

7.13.2 Agent Control

The agents within PEDA are provided with autonomy by their agent control function, which, at agent startup, creates the message handlers essential for reactive and proactive behaviour and controls the sequence and execution of the agent's interaction, primary and secondary decision support tasks for the execution lifetime of the agent.

To design the agent control functions, it was necessary to specify how each task will be invoked and the task execution sequences. This was achieved by building upon the details of the specification thus far, namely: the identified interaction and functional tasks and how the primary and secondary decision support tasks were to be realised.

It was clear from the results of the agent functionality stage (see Table 7-3), that secondary decision support tasks would need to be implemented as algorithms, rules or a combination of both. Therefore, within each agent, the agent control functionality would need to manage both rule-based tasks, for interaction and functional tasks handling prioritisation, and algorithmic functional tasks. Each agent control function therefore required two reasoning mechanisms: inference and algorithmic. Agent control diagrams were created for each agent to illustrate both the inference and algorithmic agent control functionality – available in Appendix H.

To illustrate how an agent control diagram was created, the PVD agent control diagram illustrated in Figure 7-14, and also available in Appendix H.4, will be used. How each of PVD's interaction and functional tasks are invoked and how the task execution sequence was identified, will be described.

As with all PEDA agents, the PVD agent must manage the execution of its core tasks while monitoring for, and reacting to, messages from other agents. The agent control function is therefore required to create and manage a number of concurrent reasoning streams which are illustrated in Figure 7-14.

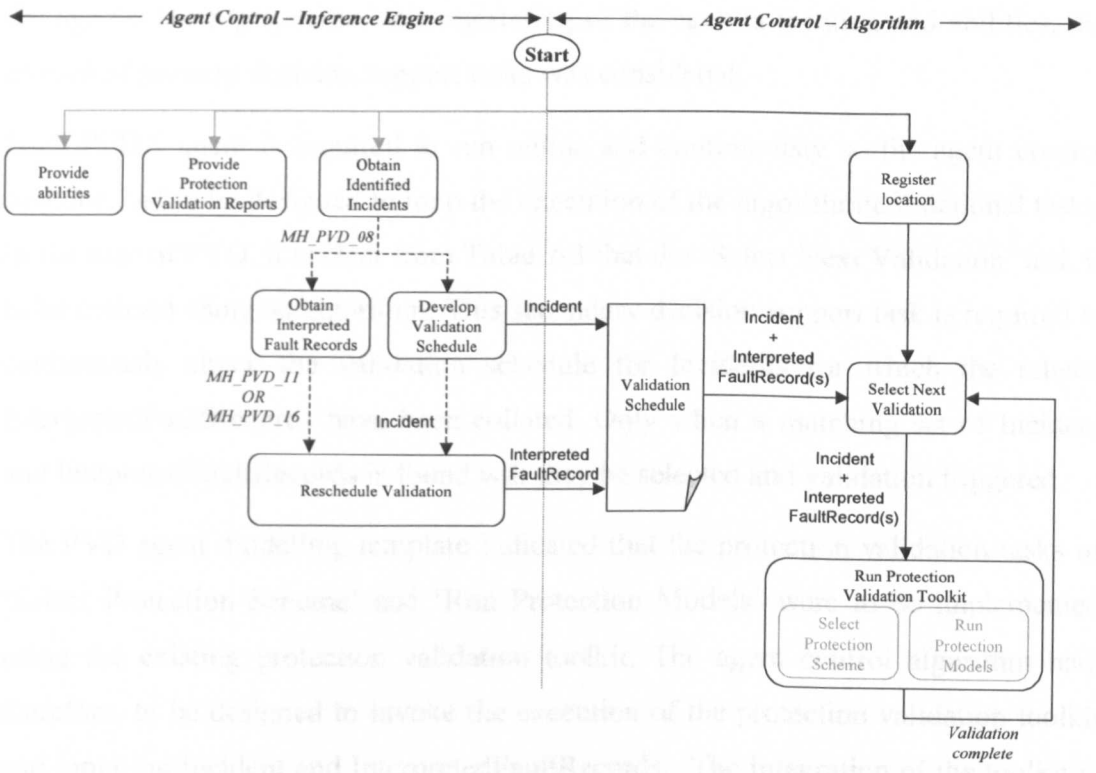


Figure 7-14 PVD Agent control diagram

The primary reasoning stream runs in the algorithmic layer and controls the execution of tasks which are ***not*** triggered in reaction to the receipt of a new message or additions to agent memory and are to be implemented as algorithms or via integration of existing systems.

In this reasoning stream, each agent control function had to be designed to, at agent startup, execute a 'Register location' interaction task registering the agent's physical network location with the Nameserver. To handle queries from the Facilitator as to the agent's data and information provision abilities, an additional reasoning stream had to be designed to, at agent startup, create, within the inference layer, the message handlers necessary to process and respond to queries from the Facilitator. In the case of PVD, the 'Provide abilities' task would respond to queries from the Facilitator, indicating that PVD can provide instances of the ProtectionValidationReports ontological class – this was identified from the agent modelling template.

Although neither the 'Register location' nor 'Provide abilities' tasks are explicitly represented in the agent modelling templates, they had to be included in the agent control diagrams as they facilitate information discovery and interactions between

the agents. Having specified the registering of the agent's location and abilities, the control of primary decision support tasks was considered.

Each PEDA agent is required to run online and continuously so the agent control function had to be designed to loop the execution of the algorithmic functional tasks. In the case of PVD, it is clear from Table 7-3 that the 'Select Next Validation' task is to be realised using an algorithm. This secondary decision support task is required to continuously check the validation schedule for Incidents for which the related InterpretedFaultRecords have been collated. Only when a matching set of Incident and IntepretedFaultRecords is found will they be selected and validation triggered.

The PVD agent modelling template indicated that the protection validation tasks of 'Select Protection Scheme' and 'Run Protection Models' were to be implemented using the existing protection validation toolkit. The agent control algorithm had, therefore, to be designed to invoke the execution of the protection validation toolkit and input the Incident and InterpretedFaultRecords. The integration of the toolkit is discussed in section 7.13.3. For the moment, it was sufficient to design the agent control algorithm to trigger the validation on completion of the 'Select Next Validation' task and to reinitiate the 'Select Next Validation' task on completion of validation.

The FRR, FRI and PVD agents all exhibit proactive behaviour to establish the mechanisms for obtaining the Incident information necessary for them to prioritise their retrieval, interpretation and validation tasks. To handle this proactive behaviour, an additional reasoning stream had to be included in the design of the agent control function for each of the three agents. Design of the proactive reasoning stream within PVD is described.

The proactive reasoning stream executes the 'Obtain Identified Incidents' task at startup to commence the message sequence depicted in Figure 7-9 (SD_PEDA_05) to identify and subscribe to a provider of Incidents. Having created and sent the first 'query-ref' message to the Facilitator, the 'Obtain Identified Incidents' task will create message handlers MH_PVD_03 to MH_PVD_08 to handle the responses as indicated in Appendix G.4.2. Whenever an 'inform (Incident)' message is received, MH_PVD_08 will fire and add the received Incident to memory.

When MH_PVD_08 fires, the agent control function must begin the retrieval and collation of the InterpetedFaultRecords related to the Incident and create a validation schedule, if one has not already been created. This was achieved by designing the agent control function to create two additional concurrent reasoning streams when MH_PVD_08 fires, one for each of the 'Obtain Interpreted Fault Records' and 'Develop Validation Schedule' tasks.

Across both these reasoning streams is the 'Reschedule Validation' task which the agent control function must invoke either when an Incident is provided by the 'Develop Validation Schedule' task or message handlers MH_PVD_11 or MH_PVD_16 fire in response to an InterpretedFaultRecord being received.

All PEDA agents are required to react to messages from other agents for provision of generated resources, either by a 'subscribe', 'query-ref' or 'request' message. To monitor the incoming mailbox for, and react to, these messages, an additional reasoning stream is required to handle the agent's reactive behaviour. Each agent control function has been designed to initiate an additional reasoning stream in the inference layer for just this purpose.

Within PVD, this additional reasoning stream is responsible for execution of the 'Provide Protection Validation Reports' task. This task requires no additional functionality other than that included in message handlers MH_PVD_01 and MH_PVD_02 designed earlier and presented in Appendix G.4.1. When either a 'subscribe' or 'query-ref' message is received the appropriate message handler will fire and take the necessary action. This reasoning stream remains active for the execution lifetime of the agent.

7.14 Chapter Summary

This chapter has demonstrated the successful application of the methodology, described in chapter six, for specification of decision support MAS to the automation of post-fault disturbance analysis.

Following the justification of a MAS solution to post-fault disturbance analysis automation, the design issues covered within this chapter are:

- *Requirements and knowledge capture*: Capturing of the protection engineering requirements for automated disturbance analysis and the elicitation of the activities, resource and reasoning knowledge for use within the MAS.
- *Disturbance analysis task decomposition*: Specification of the disturbance analysis task assigned to the MAS and its decomposition into sub-tasks.
- *Disturbance analysis ontology modelling*: Identification and creation of an ontology for representation of the core concepts within the field of power system post-fault disturbance analysis.
- *Reuse of Legacy Systems*: Assessment of whether legacy software can be reused within the MAS to perform disturbance analysis sub-tasks.
- *Agent identification and modelling*: Identification of the required disturbance analysis agents and assignment of their functional and interaction subtasks.
- *Agent functionality*: Identification of the most appropriate reasoning technique for realising an agent's disturbance analysis behaviour.
- *Agent interactions modelling*: Modelling of the message exchanges necessary for agent collaboration and essential for automation of disturbance analysis.
- *Design of agent behaviour*: Identification of the message handlers essential for agent's social, proactive and reactive behaviour and design of the agent control functions essential for agent autonomy.

The specification process described in this chapter resulted in a specification for the PEDA MAS and its agents. The next chapter describes the implementation and deployment of the PEDA MAS and evaluates its performance using a case study. The models, templates and output of the specification process that form the bases of the specification have been documented in Appendices B to H.

7.15 References

[1]. Schreiber, G., Akkermans, H., Anjwierden, "Knowledge Engineering and Management: The CommonKADS Methodology", MIT Press, 1999.

[2]. A. Bernaras, et al, "An Ontology for Fault Diagnosis in Electrical Networks", ISAP 96, January 28 – February 2, pp 199-203, 1996.

[3]. "Standard Common Format for Transient Data Exchange (COMTRADE) for Power Systems", ANSI/IEEE C37.111.1991

[4]. "FIPA Communicative Act Library Specification", XC00037H, [Online], Available: http://www.fipa.org/repository/index.html

Chapter 8: PEDA Implementation and Performance Evaluation

8.1 Chapter Overview

The previous chapter illustrated how the methodology presented in chapter six was used to create a specification for the PEDA MAS. This chapter describes the implementation of the specification.

The realisation of the agents through integration of legacy systems and implementation of the message handlers and agent control functions essential for agent autonomy, reactivity, proactiveness and social interaction is described. The anticipated architecture for deployment of the agents within ScottishPower PowerSystems is also presented.

The performance of the PEDA MAS is also evaluated using the case study described in chapter four previously used to evaluate the Telemetry Processor's performance. In this chapter the performance evaluation focuses on evaluating PEDA's ability to manage the gathering and dissemination of disturbance diagnosis data and information. It should be noted that although in this thesis the evaluation of PEDA's capabilities is limited to one case study in practice the performance of PEDA was evaluated using a diverse range of case studies.

8.2 Selection of Agent Building Toolkit

A number of toolkits exist that allows developers to implement software systems as as MAS. A report commissioned by AgentLink [1] provides a comprehensive review of over thirty of the available toolkits. Three of the mainstream toolkits are briefly reviewed below:

- Zeus Agent Building Toolkit [2] was developed by BT exact Technologies' Intelligent Systems group and provides a library of software components and tools that facilitate the design, development and deployment of MAS.
- JACK [3] was developed by Agent Oriented Software Pty. and is an environment for building, running and integrating commercial Java-based multi-agent software using a component based approach. JACK agents can be

organized into teams for modelling team behaviour or performing joint tasks, however they are restricted to running within a JACK environment.

- FIPA-OS [4] was developed by Emphoria as a component based toolkit for enabling the rapid development of FIPA compliant agents. The Foundation for Intelligent Physical Agents (FIPA) has issued many accepted and experimental specifications for the specification of agent communication, agent management and agent message transport. These specifications are becoming the standard within the agent research community and the majority are supported within FIPA-OS. This toolkit is the most widely supported and is likely to remain so for the foreseeable future.

At the time of commencing the research within this thesis the Zeus toolkit was one of the most well-supported and advanced toolkits available and was adopted for implementation of the PEDA specification. However, as the research progressed, more toolkits were introduced and support for the Zeus toolkit faltered. Nevertheless, the Zeus toolkit continued to provide a suitable mechanism for implementing PEDA.

It should be noted that neither the methodology presented in chapter six, nor the resulting PEDA specification, is tied in any way to a particular agent building toolkit. The PEDA specification could have been implemented using any of the four toolkits described above and by the majority of the toolkits summarised in the report commissioned by AgentLink [1].

8.3 Specification Implementation

Implementation of the PEDA specification commenced with creation of the disturbance analysis ontology. The ontology is a fundamental requirement of PEDA and is used by all agents. It cannot be obtained through agent interaction at runtime so therefore had to be created prior to agent implementation.

Ontology creation was relatively simple since Zeus [2] provided an ontology creation facility. This allowed the classes and their attributes to be input into the software which then created a textual representation of the disturbance diagnosis ontology in an ontology file, a copy of which would be deployed with each agent.

Having created the disturbance analysis ontology, implementation of the PEDA agents could proceed. Agent implementation was assisted by a facility within Zeus to automatically generate agent shells written in Java. These agent shells segregated the agent's execution process into three execution threads that would run concurrently. Two threads were devoted to the agent's incoming and outgoing mailbox and provided a common inference engine to manage the agent's message handlers. The remaining thread was the main algorithmic thread to manage execution of the agent's core functional tasks.

The agent shells also provide a FIPA compliant communications protocol for agent interactions and execute the 'Register location' and 'Register abilities' tasks within the main algorithmic thread. The following sections describe how the specification was used to populate the agent shells to realise the PEDA agents.

8.3.1 Utility Agents Implementation

The Nameserver and Facilitator agents were the first to be implemented since the other PEDA agents would be required to register their network locations and abilities with them at startup. Implementation was straightforward since Zeus provided generic utility agents which could be used within any MAS. No additional configuration or software development was therefore required.

The Nameserver agent maintains a database of the name and network location of agents that have registered with it. The Nameserver only exhibits reactive behaviour since it is only required to receive and acknowledge an agents' registration request.

The Facilitator agent maintains a database of the name and information and data provision abilities of each agent. To gather the abilities information, the Facilitator agent exhibits both pro-active and reactive behaviour. Pro-active behaviour is exhibited by the regular querying of the Nameserver for addresses of agents, and then the querying of each agent for its information and data provision abilities. Reactive behaviour is exhibited by responding to an agents query for providers of a data or information resource.

8.3.2 IEI Agent Implementation

IEI was the first, and perhaps simplest, of the PEDA agents to be implemented. The implementation simplicity can be attributed to two features of the IEI specification:

- Functional tasks are realised through integration of the Telemetry Processor.
- IEI is not required to exhibit proactive behaviour.

The IEI agent control diagram in Appendix H.1 indicated that the only algorithmic functionality requiring addition to the IEI agent shell was the Telemetry Processor wrapper. Upon execution, the wrapper would run the Telemetry Processor software within the agent's algorithmic execution thread and output identified incidents and events to the agent's memory. These incidents and events would be added to memory as facts constructed using the disturbance analysis ontology, i.e. Incident and Event, which could, in turn, be disseminated to other agents by message handlers.

To achieve the required functionality, the PEDA specification identified that additional software would be required to manage Telemetry Processor initiation and configuration and monitor the output database for incidents and events. This additional software would be introduced into the Telemetry Processor source code via a wrapper. Analysis of the original source code identified two Java classes dedicated to performing these functions.

To avoid source code modification, two new Java classes were written, specifically for the wrapper, extending the original classes with functionality providing the control of Telemetry Processor initiation and configuration and enabling output to the agent memory of incidents and events as ontological facts. The wrapper runs the Telemetry Processor by executing these classes overriding the original classes, thereby providing the required functionality.

The final components of IEI to be implemented were the reactive interaction tasks, namely: 'Provide SCADA', 'Provide Incidents' and 'Provide Events'. The six message handlers required to implement these tasks had already been specified and were listed in Appendix G.1.1.

To implement the message handlers, the agent shell required the addition of the message patterns that the agent must monitor its inbox for and the function which

should be executed when a matching message is received. The agent shell handled the addition of each message pattern to the inference engine and the execution of the function on firing of the message handler. The only remaining implementation task was to write the function to be executed. This was a relatively simple task and only required writing of code to create and send messages in response to the received message or to generate additional message handlers.

The architecture of the implemented IEI agent is presented in Figure 8-1 illustrating the components initially present in the agent shell, the additional groups of message handlers to realise each interaction task and the wrapper required to realise the post-fault disturbance analysis tasks assigned to IEI.

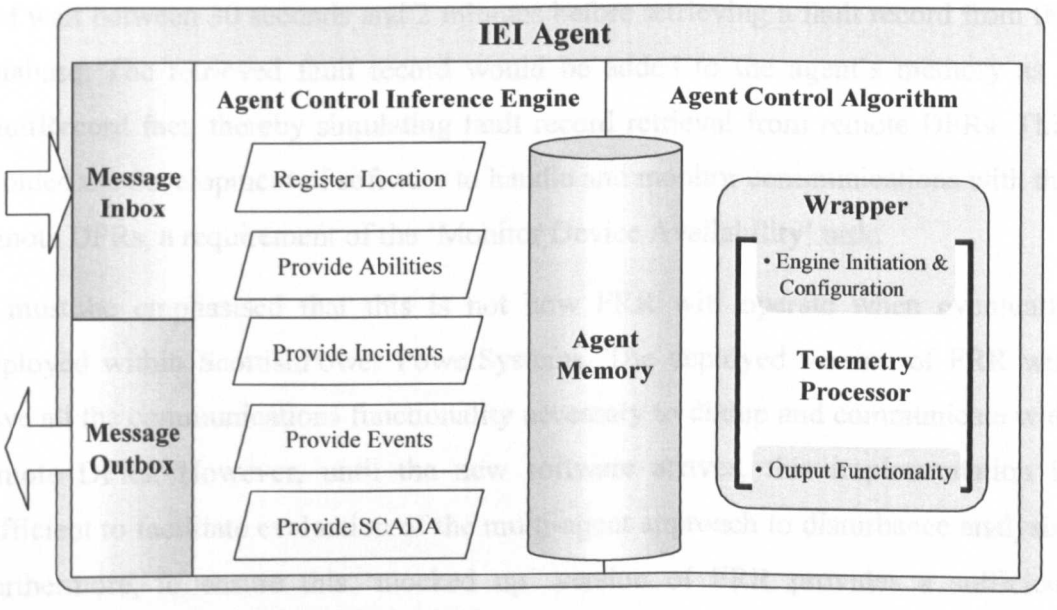


Figure 8-1 IEI agent architecture

8.3.3 FRR Agent Implementation

Unlike IEI, none of FRR's functional tasks were to be implemented using existing software. Although software was available which effectively managed autopolling and fault record retrieval, it did not offer any external software interfaces and the source code was not available for modification. As a result, FRR could neither

interface with the software using a transducer nor extend the existing code using a wrapper. A software rewrite was therefore required and had been specified.

At this stage in the research project, the focus was on implementing a PEDA prototype to not only assess the quality of the specification produced by following the proposed methodology but also whether a MAS brought the anticipated systems integration and automated data collation benefits to post-fault disturbance analysis. It was therefore neither desirable nor practical to expend time and effort on rewriting the fault record retrieval software. An alternative was chosen which still allowed evaluation of the multi-agent approach but avoided a software rewrite.

The alternative was to implement the 'Retrieve' task as an algorithm which, when called by the 'Select Next Retrieval' task, would check a database of fault records and wait between 30 seconds and 2 minutes before retrieving a fault record from the database. The retrieved fault record would be added to the agent's memory as a FaultRecord fact, thereby simulating fault record retrieval from remote DFRs. This avoided the development of software to handle and monitor communications with the remote DFRs, a requirement of the 'Monitor Device Availability' task.

It must be emphasised that this is not how FRR will operate when eventually deployed within ScottishPower PowerSystems. The deployed version of FRR will have all the communications functionality necessary to dialup and communicate with remote DFRs. However, until the new software arrives, this implementation is sufficient to facilitate evaluation of the multi-agent approach to disturbance analysis. Furthermore, to ensure this 'mocked up' version of FRR provides a sufficient platform to assess PEDA's disturbance analysis capabilities, the fault records used were not random but reflected the real data to be used in the case study disturbances.

Within the 'mocked up' version of FRR the functional tasks were implemented in Java as algorithms which could, all except 'Reschedule Retrieval', be executed within the agent shell's algorithmic execution thread by the agent control function illustrated in Appendix H.2. 'Reschedule Retrieval' was part of the agent's proactive behaviour and would instead be executed in the shell's inference engine.

The only proactive behaviour FRR is required to exhibit is performed by the 'Obtain Identified Incidents' interaction task to subscribe to an incident provider. The

message handlers required by this interaction task had already been specified in Appendix G.2.2 and were implemented within the agent shell in exactly the same manner as in IEI. Message handler MH_FRR_09 is of particular significance since it is responsible for triggering execution of the 'Reschedule Retrieval' algorithm when an inform message is received containing an Incident fact.

The final component was the 'Provide Fault Records' interaction task required for reactive behaviour. The message handlers required to provide the 'subscribe', 'query-ref' and 'request' message handling functionality were implemented in the agent shell. Only message handler MH_FRR_03 for handling requests for fault record retrieval is of particular relevance since it executed the 'Reschedule Retrieval' task on firing.

The architecture of the implemented FRR agent is presented in Figure 8-2.

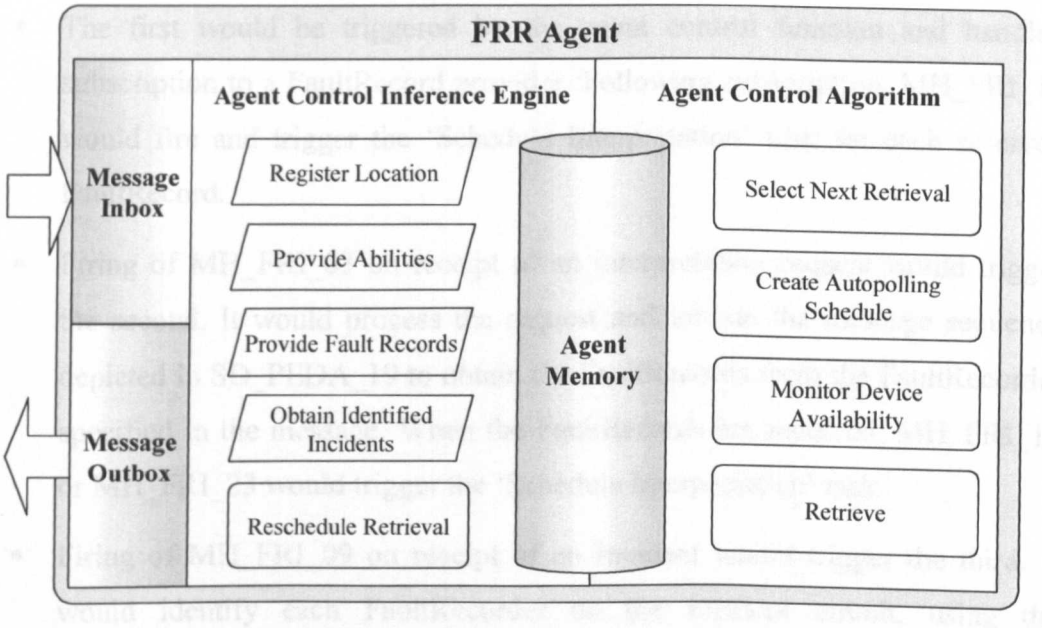


Figure 8-2 FRR agent architecture

8.3.4 FRI Agent Implementation

As described in FRI's agent modelling template in Appendix E.3 and illustrated in the agent control diagram in Appendix H.3, the FRI agent control function must execute two interaction tasks to prioritise the interpretation of fault records for

proactive behaviour: 'Obtain Identified Incidents' and 'Obtain Retrieved Fault Records'.

The 'Obtain Identified Incidents' task is implemented using message handlers in exactly the same way as in IEI and FRR, however message handler MH_FRI_09, which adds a received Incident fact to memory, also triggers the execution of 'Obtain Retrieved Fault Records' to initiate the retrieval of the incident related FaultRecord facts and ensure that their interpretation is prioritised.

The message handlers implemented for the 'Provide Interpreted Fault Records' reactive task also trigger the 'Obtain Retrieved Fault Records' task to handle a request for interpretation of fault records from a particular FaultRecorder.

Each possible trigger for 'Obtain Retrieved Fault Records' requires different functionality. The task was therefore implemented as three sub-algorithms which could be triggered independently:

- The first would be triggered by the agent control function and handled subscription to a FaultRecord provider. Following subscription, MH_FRI_15 would fire and trigger the 'Schedule Interpretation' task for each received FaultRecord.
- Firing of MH_FRI_03 on receipt of an interpretation request would trigger the second. It would process the request and initiate the message sequence depicted in SD_PEDA_19 to obtain the FaultRecords from the FaultRecorder specified in the message. When the FaultRecords are received, MH_FRI_18 or MH_FRI_23 would trigger the 'Schedule Interpretation' task.
- Firing of MH_FRI_09 on receipt of an Incident would trigger the third. It would identify each FaultRecorder on the incident circuit, using the substation and circuit identifiers, in the Incident fact and trigger the 'Schedule Interpretation' task for each identified FaultRecorder.

The 'Schedule Interpretation' task was implemented as an algorithm which, depending on what triggered its execution, would decide where to place the FaultRecord fact in the interpretation schedule. FaultRecord facts related to Incidents would received highest priority and would be placed at the top of the interpretation

schedule, followed by those related to request messages then FaultRecord facts provided by subscription.

The 'Select Next Fault Record' task was also implemented as an algorithm but executed within the agent shell's algorithmic execution thread. The agent control function was written to execute the task as regular intervals until a FaultRecord was available in the interpretation schedule. If more than one FaultRecord was available, the FaultRecord at the top of the schedule was selected for interpretation.

As illustrated in the agent control diagram in Appendix H.3 and described in section 7.10 of the thesis, FRI requires integration of existing fault record interpretation software to perform the agent's 'Interpret Fault Record' task using a wrapper.

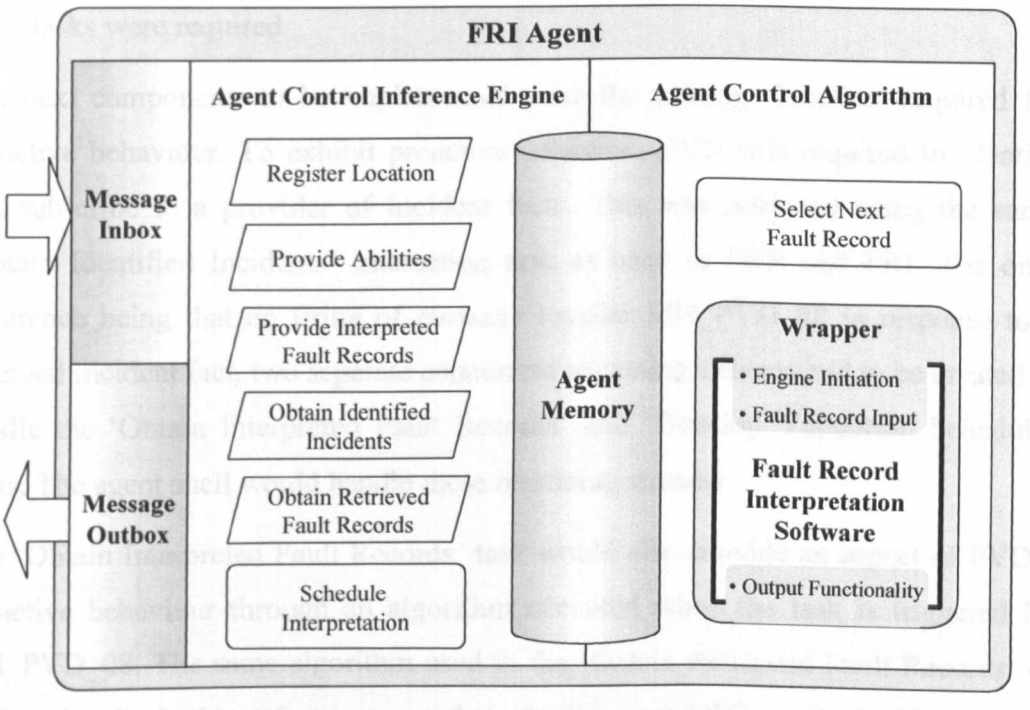


Figure 8-3 FRI agent architecture

The wrapper was implemented using exactly the same approach as adopted for IEI using Java classes to override existing classes and initiate the interpretation engine, input the selected FaultRecord fact and create InterpretedFaultRecord facts containing the destination directory of the interpretation results text file. The only difference was that the integrated fault record interpretation software does not run

continuously and the agent control function must execute the wrapper each time a new FaultRecord fact is provided by 'Select Next Fault Record'.

The architecture of the implemented FRI agent is presented in Figure 8-3.

8.3.5 PVD Agent Implementation

The PVD realisation process commenced with implementation of the message handlers required for the reactive interaction task 'Provide Protection Validation Reports' – the message handlers are specified in Appendix G.4.1. The PVD agent modelling template in Appendix E.4 indicated that the PVD agent was only required to provide ProtectionValidationReports through 'subscribe' and 'query-ref' FIPA performatives. This simplified the implementation process since no interactions with other tasks were required.

The next components to be implemented were the message handlers required for proactive behaviour. To exhibit proactive behaviour PVD was required to identify and subscribe to a provider of Incident facts. This was achieved using the same 'Obtain Identified Incidents' interaction task as used in FRR and FRI. The only difference being that on firing of message handler MH_PVD_08 in response to a received Incident fact, two separate concurrent reasoning streams had to be created to handle the 'Obtain Interpreted Fault Records' and 'Develop Validation Schedule' tasks. The agent shell would handle these reasoning streams.

The 'Obtain Interpreted Fault Records' task would also provide an aspect of PVD's proactive behaviour through an algorithm executed when the task is triggered by MH_PVD_08. The same algorithm used in the 'Obtain Retrieved Fault Records' of FRR to handle Incident facts was used to identify each DFR on the incident circuit and generate a FaultRecorder fact. The algorithm then initiates the message sequence depicted in SD_PEDA_20 and creates message handlers MH_PVD_09 to MH_PVD_16 to handle collation of Incident related InterpretedFaultRecords. When an InterpretedFaultRecord is received, the 'Reschedule Validation' task is triggered.

'Develop Validation Schedule' was implemented as an algorithm triggered when an Incident fact is received. The algorithm was required to create a validation schedule,

if one did not exist, and update the schedule with the latest incident. The validation schedule was implemented as a two dimensional array, as illustrated in Figure 8-4, so that received InterpretedFaultRecord facts could be added to the validation schedule and grouped with their related Incident facts. This would be performed by the 'Reschedule Validation' task.

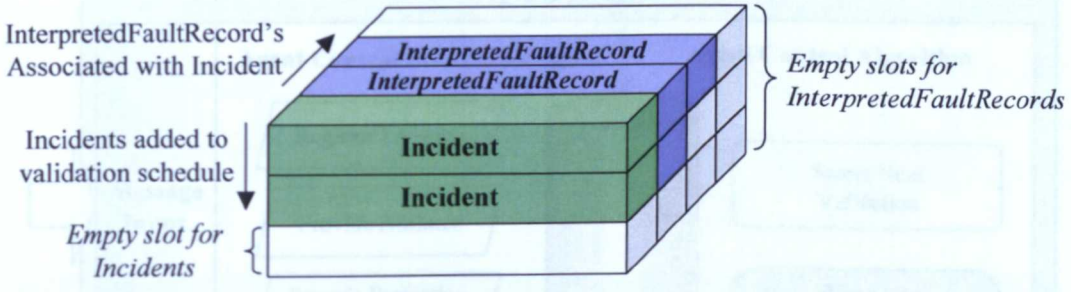


Figure 8-4 Illustration of the array used to hold the validation schedule

An algorithm was written for the 'Reschedule Validation' task which would reschedule the validation schedule based on the number of InterpretedFaultRecords expected for an Incident and the number actually received. When all InterpretedFaultRecords expected for an Incident were available it would be moved to the top of the schedule and a flag set indicating that it is ready for validation. If all expected InterpretedFaultRecords are not received within a timeout period, it may be possible to perform a partial validation so the Incident is scheduled for validation but with lower priority.

The 'Select Next Validation' task was also implemented as an algorithm but executed within the agent shell's algorithmic execution thread. The agent control function was written to execute the task at regular intervals until an Incident is scheduled for validation.

As illustrated in the agent control diagram in Appendix H.4, PVD requires integration of an existing protection validation toolkit to perform the agent's 'Select Protection Scheme' and 'Run Protection Models' tasks using a wrapper.

The wrapper was implemented using exactly the same approach as adopted for IEI and FRI using Java classes to override existing classes and initiate the interpretation

engine, select the protection scheme based on the incident circuit, input the selected InterpretedFaultRecord facts and create ProtectionValidationReport facts containing the destination directory of the protection validation report text file.

The architecture of the implemented PVD agent is presented in Figure 8-5.

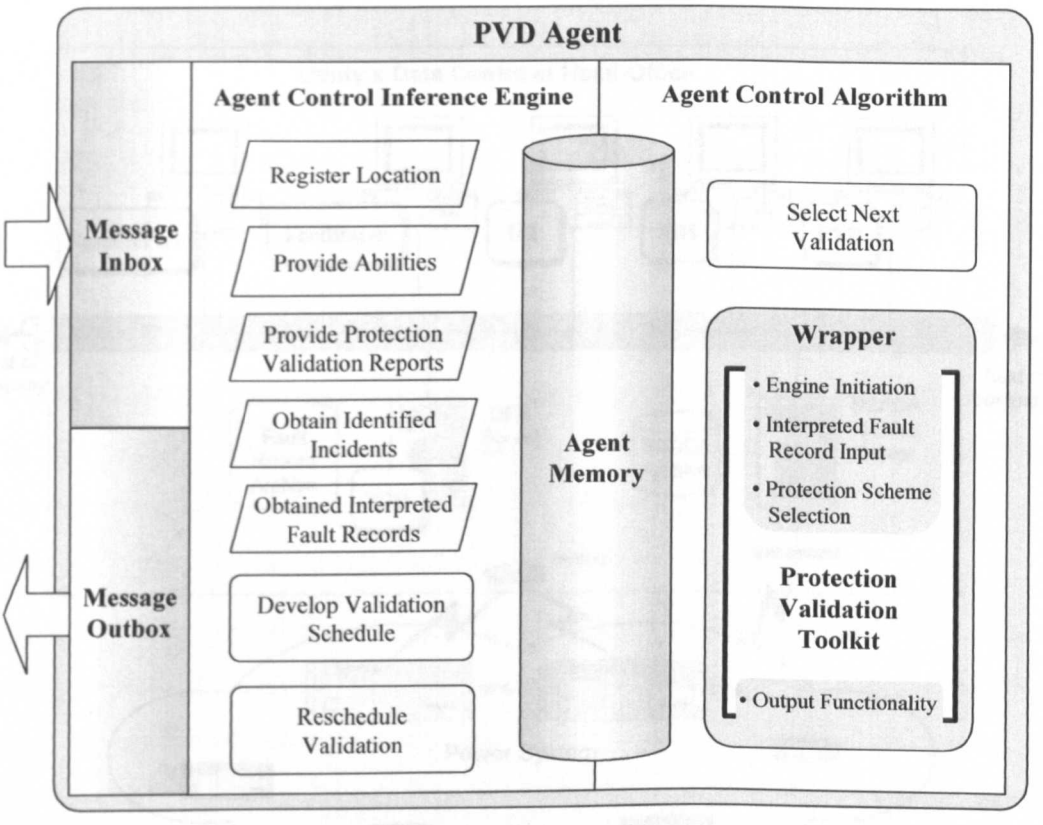


Figure 8-5 PVD agent architecture

8.4 PEDA Deployment

Although there is no physical restriction as to where the PEDA agents are deployed in a network, it is logical to place the agents in close proximity to their primary data sources and to those other agents involved in interactions so that communications delays are minimised. It is therefore anticipated that the six PEDA agents will be deployed in a utility's data centre.

Data centres are often housed within a head-office with secure and reliable Ethernet communications to other offices and WAN or dialup modem connections to the

SCADA RTU's and DFRs installed in substations. Within the data centres, high-specification PC's and servers are used to archive retrieved SCADA and fault records and serve the data to other parts of the company. This concentration of data sources at the hub of the utility's communications infrastructure makes the data centre the ideal deployment location for the PEDA agents.

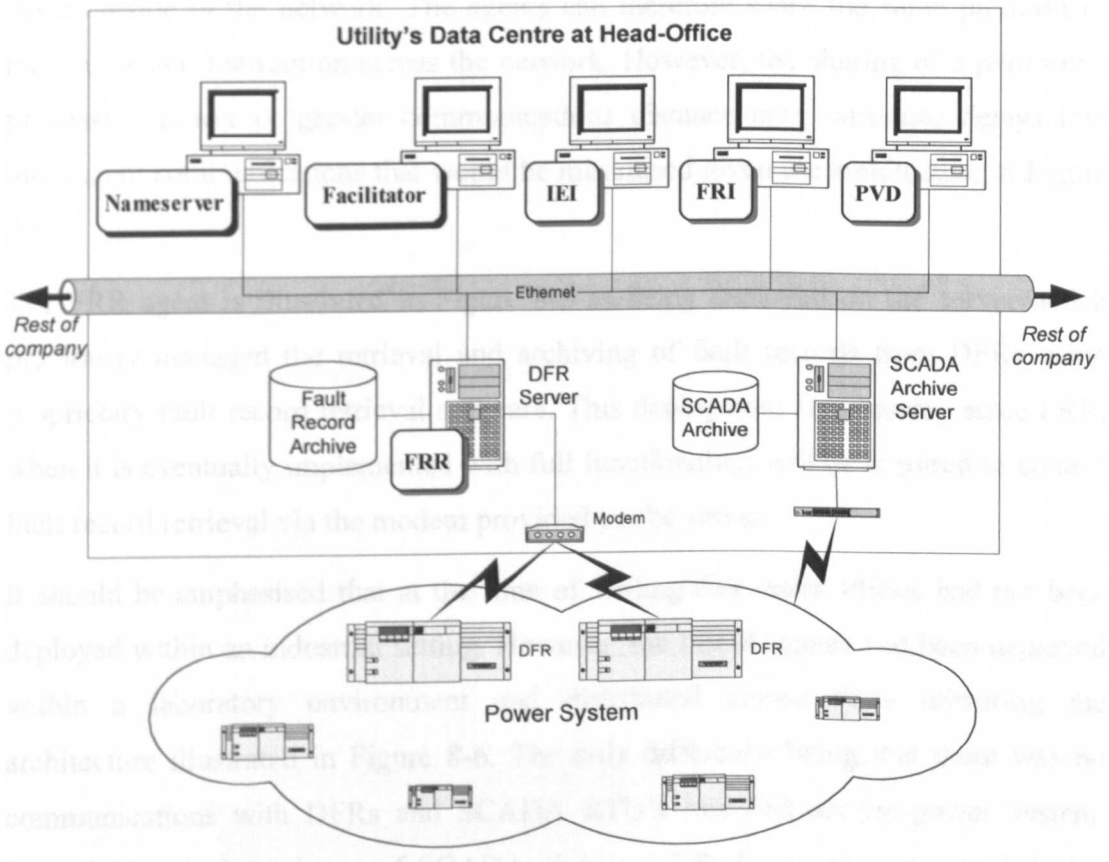


Figure 8-6 Deployment of PEDA agents

The diagram in Figure 8-6 illustrates the deployment of the PEDA agents within a utility's data centre. There are two aspects of deployment which are not explicit in Figure 8-6 but which are nonetheless essential to successful deployment:

- All agents must be deployed with knowledge of the Nameserver location. This is realised by a text file containing the IP address deployed with the agents. This ensures all agents can register their location at startup and then ask for the location of the Facilitator to enable further information discovery.

- All agents must be deployed with a copy of the disturbance diagnosis ontology. This ensures that all agents can understand the messages constructed and sent by other agents.

Figure 8-6 should not be interpreted as the only way of deploying PEDA. One of the benefits of MAS is that, provided all agents share the same network and have knowledge of the Nameserver location, there is no restriction imposed on where the agents reside in the network. The agents can therefore share the same platform or have a greater distribution across the network. However, the sharing of a platform's processing power or greater communications distance may introduce delays into inter-agent communications that would be minimised given the architecture in Figure 8-6.

The FRR agent is illustrated in Figure 8-6 as being deployed on the server which previously managed the retrieval and archiving of fault records from DFRs using proprietary fault record retrieval software. This deployment is necessary since FRR, when it is eventually implemented with full functionality, will be required to control fault record retrieval via the modem provided on the server.

It should be emphasised that at the time of writing this thesis PEDA had not been deployed within an industrial setting. However, the PEDA agents had been deployed within a laboratory environment and distributed across PC's mirroring the architecture illustrated in Figure 8-6. The only difference being that there was no communications with DFRs and SCADA RTU's installed on the power system. Instead historical databases of SCADA alarms and fault records, generated during actual disturbances, were obtained from ScottishPower PowerSystems and used to evaluate the disturbance diagnosis performance of PEDA.

8.5 PEDA User Interface

Thus far, the research presented in this thesis has focused on the specification and implementation of the PEDA agents required to automate post-fault disturbance analysis. It is now appropriate, at this stage in the thesis, to consider the means by

which the protection engineer will gain access to the disturbance data and information generated by PEDA, i.e. the user interface.

The most basic form of user interface utilised within PEDA are those that are associated with each agent. These interfaces are opened automatically when each agent is started and run on the same hardware platform as the agent. Each agent has two distinct interfaces displaying the following:

- *Agent Functionality:* Provides a window on the interactions the agent has been involved in and on its knowledge of ongoing disturbances, obtained through the information received from other agents. The contents of both the incoming and outgoing mailbox are displayed and the viewer can select each message to view message detail. The information is received as instances of ontological facts, which can also be viewed.
- *Disturbance Analysis Functionality:* This interface provides a window on the disturbance analysis tasks the agent is performing and on the retrieved disturbance data and generated information. In all but FRR, the original legacy system user interfaces are reused, e.g. in IEI the Telemetry Processor user interface illustrated in Figure 4-11 is used. In the case of FRR, a new interface has been developed as a temporary measure until the new FRR software arrives.

These individual agent interfaces are ideal for providing a local indication of the interactions the agent has participated in and the results of the disturbance analysis tasks the agent is performing. However, given that the PEDA agents may be distributed across a large network and deployed on different platforms, the requirement for the protection engineer to move between different local user interfaces is not ideal. Not only may such transitions between PC's and locals be unfeasible but this is placing the onus on the protection engineer to collate all the disturbance data and information PEDA has made available.

A more amenable and efficient solution is to use an agent to collate all the disturbance data and information generated by PEDA for display to the user. This User Interface agent would subscribe to all agents within PEDA, asking for updates of any new disturbance data or information. When new data or information arrives,

the User Interface agent can make it available to the protection engineer via a single user interface. Such an agent could operate anywhere within an organisation, even running on a protection engineers desktop PC.

To demonstrate PEDA's potential for collating disturbance related information and for the purposes of evaluating PEDA, a PEDA User Interface agent has been created.

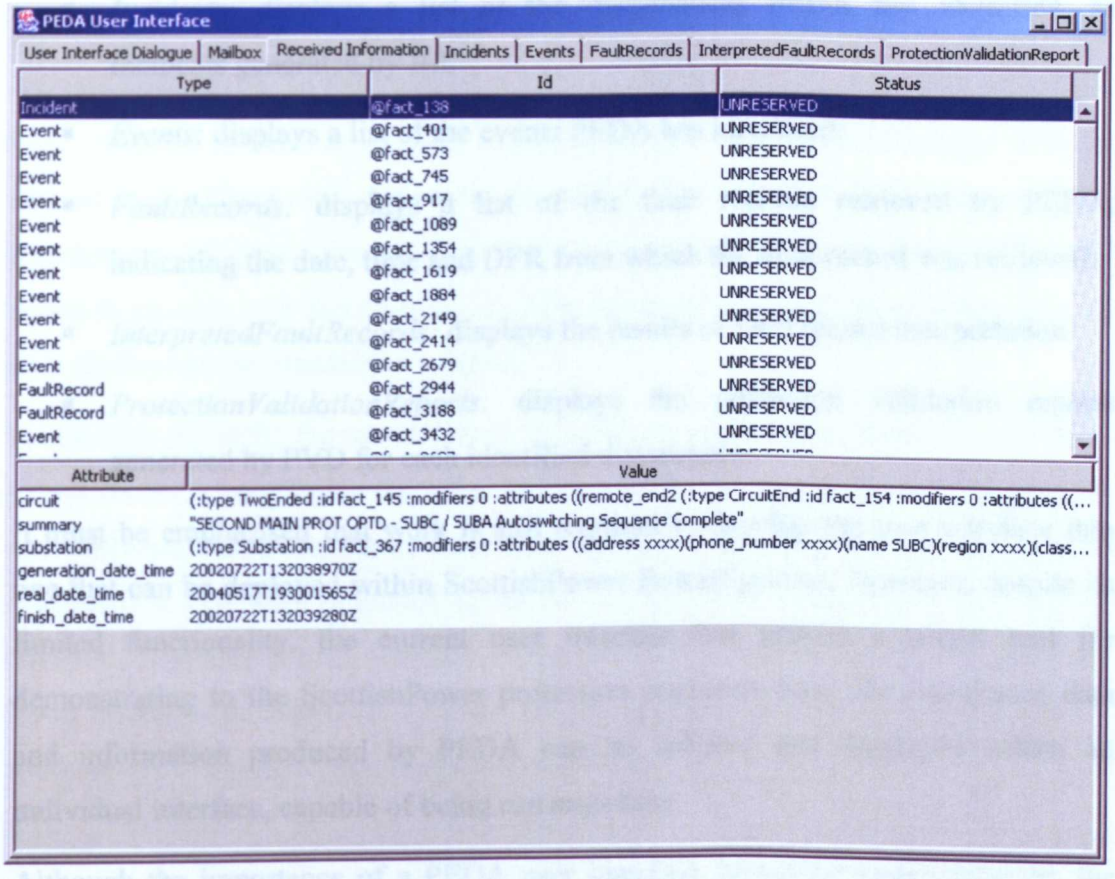


Figure 8-7 User interface developed to evaluate PEDA

The snapshot of the PEDA user interface in Figure 8-7 shows how the user interface is constructed from a number of tabbed panes. Each tabbed pane presents a different insight into the activities of the User Interface agent and the disturbance data and information provided by the PEDA agents. By selecting the appropriate tabbed pane, the user can view the following:

- *User Interface Dialogue:* displays information on the current agent state and a history of its activities, e.g. "Subscribing to IEI for Incident information".

- *Mailbox:* displays the messages received and sent by the User Interface agent.
- *Received Information:* displays the information received by the User Interface agent as instances of ontological facts. As illustrated in Figure 8-7, by selecting a fact, the user can view the values attributed to each fact attribute – note that the attribute values illustrated have been constructed using the FIPA-SL and ACL.
- *Incidents:* displays a list of the disturbances PEDA has identified, as Incidents generated by IEI.
- *Events:* displays a list of the events PEDA has identified.
- *FaultRecords:* displays a list of the fault records retrieved by PEDA, indicating the date, time and DFR from which the fault record was retrieved.
- *InterpretedFaultRecords:* displays the results of fault record interpretation.
- *ProtectionValidationReports:* displays the protection validation reports generated by PVD for each identified disturbance.

It must be emphasised that work is still required to develop the user interface into one that can be deployed within ScottishPower PowerSystems. However, despite its limited functionality, the current user interface has proved a useful tool for demonstrating to the ScottishPower protection engineers how the disturbance data and information produced by PEDA can be collated and displayed within an individual interface, capable of being run anywhere.

Although the importance of a PEDA user interface cannot be underestimated, the well-timed provision of accurate disturbance information to an engineer is strongly influenced by the underlying PEDA architecture. It should also not be forgotten that the principle goal of PEDA (as outlined in section 7.3.1), and the focus of the research described in this thesis, has been:

> *"The automation of disturbance analysis retrieval and interpretation activities and prioritisation of these activities to ensure the timely availability of decision support information to protection engineers".*

It is, therefore, necessary to focus on evaluating the disturbance analysis capabilities of PEDA and to assess the performance of the architecture in an online real-time environment. The user interface issues will be revisited later in this chapter, when the disturbance analysis capabilities of PEDA are discussed.

8.6 Evaluation of Disturbance Analysis Capability

An extensive testing program was conducted to evaluate PEDA's disturbance analysis capabilities. In the cases of IEI, FRI and PVD, the primary decision support tasks assigned to each agent were realised through integration of legacy decision support tools. There was, therefore, no need to test the core disturbance analysis functionality of these agents, since the legacy systems, now integrated within the agents to provide core functionality, had already undergone extensive testing prior to their deployment as standalone systems.

In the case of FRR, the agents' core disturbance analysis functionality was fault record retrieval and, unlike the other PEDA agents, this was to be realised through new software. However, the new software was not available and, in its absence, its functionality had been emulated using simple algorithms, simulating fault record retrieval. A simple testing program was followed to prove that fault records were being retrieved as expected. A more formal approach to testing will need to be followed when the final version of FRR is realised.

Testing of PEDA's overall post-fault disturbance analysis capabilities was a more challenging prospect due to a distinct lack of case studies with complete data sets. Nearly all case studies were lacking in disturbance fault records due to the original records being overwritten in the DFR buffers by new records before they could be retrieved – a problem PEDA will overcome.

The following sections will present a case study used during testing which will illustrate how PEDA manages the entire disturbance diagnosis process. This case study has been derived from disturbances occurring on the sponsoring utility's transmission network. The actual information generated by the PEDA agents and archived for timely presentation to the protection engineer will also be illustrated.

8.6.1 Case Study

The case study is the same as that used in section 4.6.1 of chapter four to illustrate the Telemetry Processor reasoning methodology and to evaluate its performance. This case study is ideal for evaluating PEDA's performance since it tests PEDA's ability to retrieve, interpret and collate the SCADA alarms, fault records and information generated by the PEDA agents for two distinct disturbances which are close in both time and network location. It should be noted that this case study merely represents one of a selection of case studies used during the actual evaluation process.

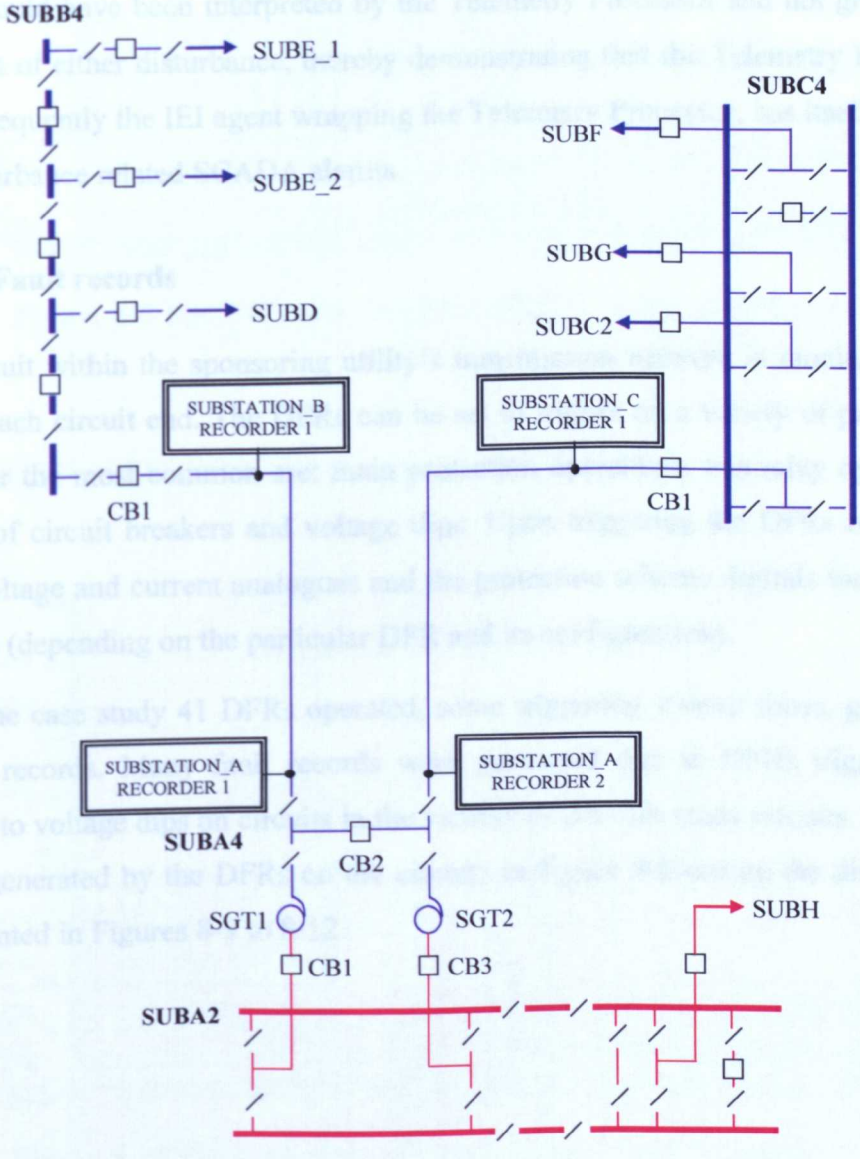


Figure 8-8 PEDA Case Study: Network Diagram

8.6.1.1 Power system network

The portion of the transmission network in which the disturbances took place and the location of DFRs are shown in Figure 8-8. The protection schemes on both feeders are illustrated in Figures 4-13 and 4-14 in chapter four.

8.6.1.2 SCADA alarms

As has already been noted in section 4.6.1.2, over 110 alarms were generated during the disturbance and only the 38 directly related to the disturbances have been presented in Table 4-2 of chapter four. This is significant, since the remaining 72 alarms would have been interpreted by the Telemetry Processor and not grouped as being part of either disturbance, thereby demonstrating that the Telemetry Processor and, consequently the IEI agent wrapping the Telemetry Processor, has itself collated only disturbance related SCADA alarms.

8.6.1.3 Fault records

Each circuit within the sponsoring utility's transmission network is monitored by a DFR at each circuit end. The DFRs can be set to trigger on a variety of parameters but by far the most common are: main protection operations, trip relay operations, tripping of circuit breakers and voltage dips. Upon triggering the DFRs record the circuit voltage and current analogues and the protection scheme digitals for a period of 600ms (depending on the particular DFR and its configuration).

During the case study 41 DFRs operated, some triggering several times, generating 58 fault records. Many fault records were generated due to DFRs triggering in response to voltage dips on circuits in the vicinity of the case study circuits. The fault records generated by the DFRs on the circuits in Figure 8-8 during the disturbance are presented in Figures 8-9 to 8-12.

Fault Records on SUBA4 / SUBB circuit

SUBSTATION_A RECORDER 1 triggered at 13:20:39.080

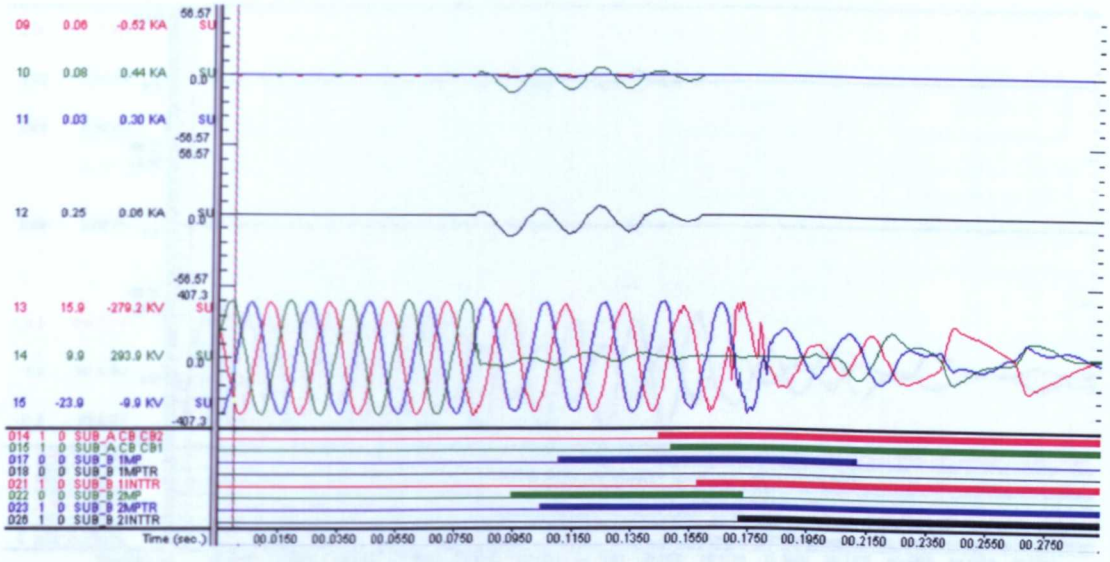


Figure 8-9 Fault Record – SUBSTATION_A RECORDER 1

SUBSTATION_B RECORDER 1 triggered at 13:20:39.080

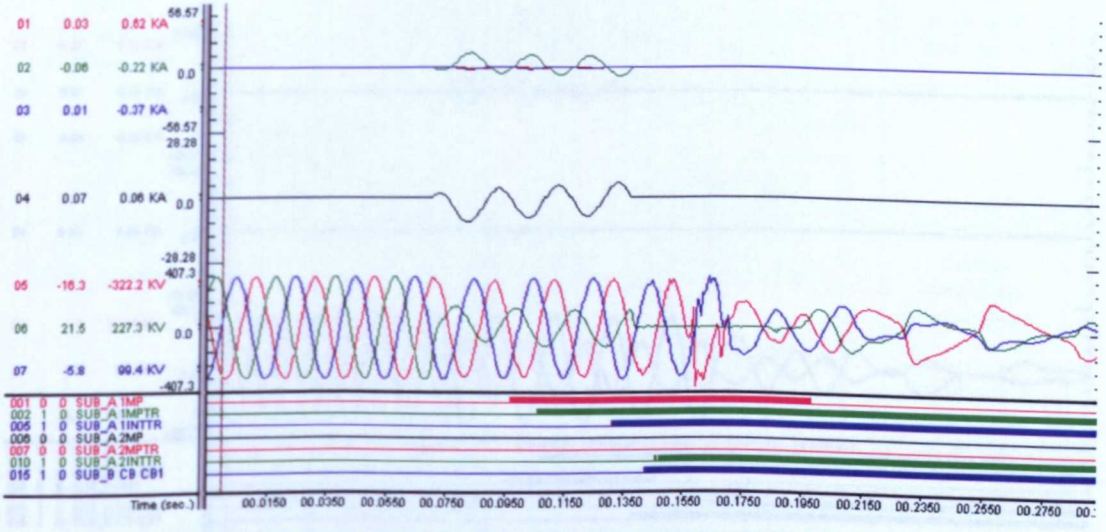


Figure 8-10 Fault Record – SUBSTATION_B RECORDER 1

Fault Records on SUBA4 / SUBC circuit

SUBSTATION_A RECORDER 2 triggered at 13:20:38.980

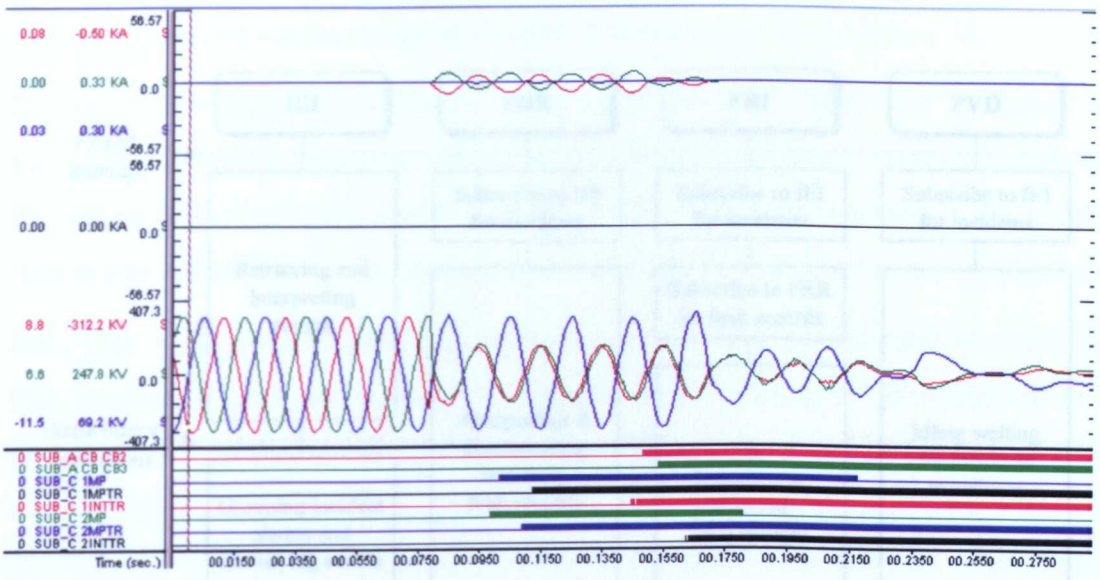


Figure 8-11 Fault Record – SUBSTATION_A RECORDER 2

SUBSTATION_C RECORDER 1 triggered at 13:20:38.980

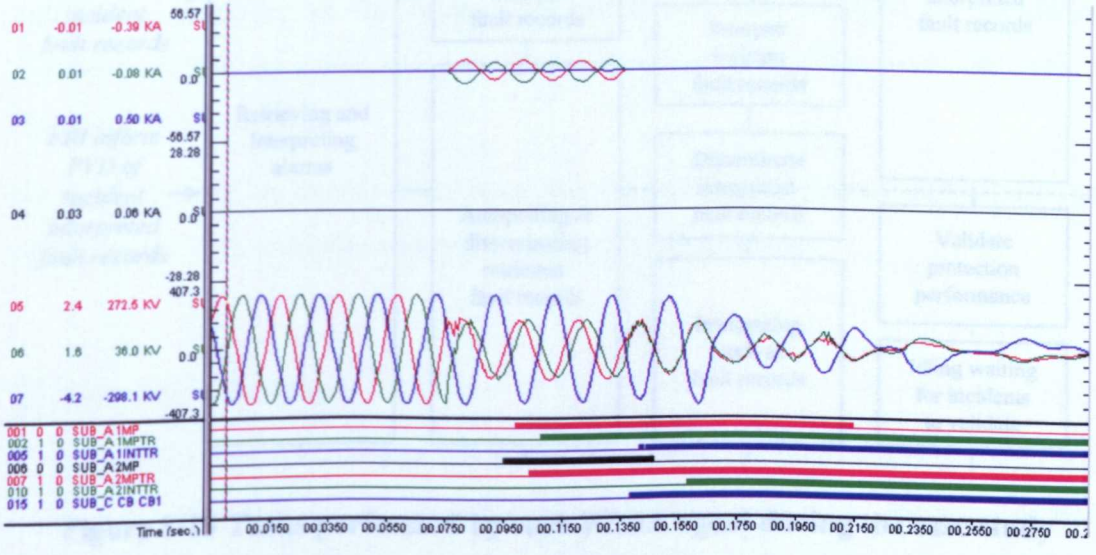


Figure 8-12 Fault Record – SUBSTATION_C RECORDER 1

8.6.2 Agent Interactions and Reasoning

To illustrate how PEDA retrieves, interprets and gathers the SCADA and DFR data the tasks performed by each agent following startup and during the disturbances are illustrated in Figure 8-13 and will be described.

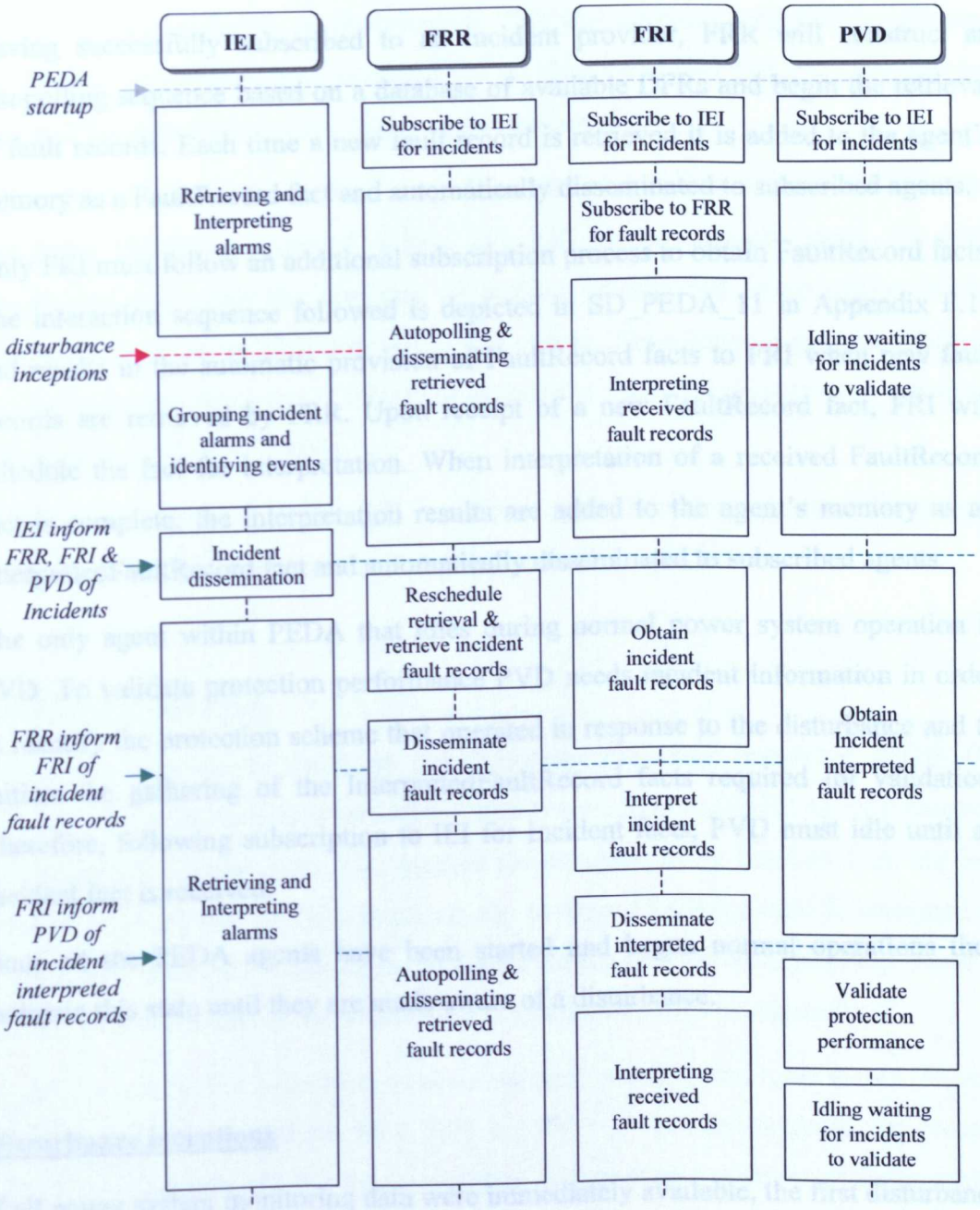


Figure 8-13 Tasks performed by each PEDA agent during the case study

Post-startup

Once all PEDA agents have been started and are registered with the Nameserver and Facilitator, FRR, FRI and PVD identify IEI as a provider of incident information and begin the incident subscription process depicted in SD_PEDA_05 in Appendix F.5. At this point in time, IEI will already have begun online retrieval and interpretation of SCADA alarms via the wrapped Telemetry Processor.

Having successfully subscribed to an incident provider, FRR will construct an autopolling sequence based on a database of available DFRs and begin the retrieval of fault records. Each time a new fault record is retrieved it is added to the agent's memory as a FaultRecord fact and automatically disseminated to subscribed agents.

Only FRI must follow an additional subscription process to obtain FaultRecord facts. The interaction sequence followed is depicted in SD_PEDA_11 in Appendix F.11 and results in the automatic provision of FaultRecord facts to FRI when new fault records are retrieved by FRR. Upon receipt of a new FaultRecord fact, FRI will schedule the fact for interpretation. When interpretation of a received FaultRecord fact is complete, the interpretation results are added to the agent's memory as an InterpretedFaultRecord fact and automatically disseminated to subscribed agents.

The only agent within PEDA that idles during normal power system operation is PVD. To validate protection performance PVD needs incident information in order to identify the protection scheme that operated in response to the disturbance and to initiate the gathering of the InterpretedFaultRecord facts required for validation. Therefore, following subscription to IEI for Incident facts, PVD must idle until an Incident fact is received.

Once all the PEDA agents have been started and begun normal operations they maintain this state until they are made aware of a disturbance.

Disturbance inceptions

If all power system monitoring data were immediately available, the first disturbance indicators would be the SCADA alarms indicating protection operation at each circuit end and the fault records generated by triggering of the circuit DFRs by

protection operation. However, all monitoring data is not immediately available with fault records remaining in the DFR buffers until retrieved by FRR

IEI will be the first agent to realise that disturbances have occurred since the SCADA alarms are the first type of monitoring data to be retrieved from the field, archived and made available to the PEDA agents in the data centre. Although fault records will exist, FRR will, more than likely, be in the process of retrieving fault records from DFRs not related to the disturbances. This will continue until FRR is informed by IEI that disturbances have occurred via Incident facts. FRI and PVD are also not aware of the disturbances until they receive Incident facts.

Recognising alarm patterns indicating disturbance inception on both circuits, the Telemetry Processor wrapped within IEI will create incident starts for each disturbance and begin the grouping of incident alarms and interpretation of the grouped alarms for incident conclusion and events. The reasoning process followed by the Telemetry Processor at this point in the case study has already been described in section 4.6.1.5 of this thesis.

At this point in the disturbance, IEI is performing the first three stages of the manual disturbance diagnosis process conducted by protection engineers.

Incident dissemination

When the disturbances conclude the Telemetry Processor integrated within IEI generates incident and event summaries, which are added to the IEI agent's memory as Incident and Event facts. To illustrate the structure of an Incident fact, the fact generated by IEI for the disturbance on the SUBA4 / SUBB circuit is presented in Figure 8-14. Note how many of the attributes are constructed from facts representing instances of the classes in the disturbance diagnosis ontology in Appendix C.

On addition of the Incident facts to agent memory, subscription rules within IEI will fire and automatically inform FRR, FRI and PVD of the new incidents. To comply with FIPA standards [5] the *inform* message will contain a number of fields which indicate the sender of the message, the intended receiver, the id of the message it is being sent in reply to and the content. An example of the Incident *inform* message

sent from IEI to FRR in response to an earlier subscribe message with id FRR47 is presented in Figure 8-15. The content field of this message has been constructed using the FIPA SL content language [6] from the contents of the original subscription message and the Incident fact presented in Figure 8-14.

```
(Incident :generation_date_time      20020722T132039070Z
          :finish_date_time          20020722T132039280Z
          :real_date_time            20030804T091836607Z
          :substation   (Substation :substation_name SUBA4)
          :circuit      (TwoEnded :source_end (CircuitEnd :substation (Substation
                                                  :substation_name SUBA4))
                                  :remote_end2 (CircuitEnd :substation (Substation
                                                  :substation_name SUBB)))
          :summary      "SECOND MAIN PROT OPTD - SUBA4 / SUBB Autoswitching
                        Sequence Complete")
```

Figure 8-14 Incident fact for disturbance on SUBA4 / SUBB circuit

inform
:sender IEI
:receiver FRR
:in-reply-to FRR47
:content "((= (all ?x (= ?x (Incident))) (Incident :generation_date_time 20020722T132039070Z :finish_date_time 20020722T132039280Z :real_date_time 20030804T091836607Z :substation (Substation :substation_name SUBA4) :circuit (TwoEnded :source_end (CircuitEnd :substation (Substation :substation_name SUBA4)) :remote_end2 (CircuitEnd :substation (Substation :substation_name SUBB))) :summary "SECOND MAIN PROT OPTD - SUBA4 / SUBB Autoswitching Sequence Complete")))"

Figure 8-15 FIPA inform message sent by IEI informing FRR of the Incident on SUBA4 / SUBB circuit

Upon receipt of each Incident *inform* message, FRR reschedules its autopolling sequence to prioritise the retrieval of fault records from DFRs on the incident circuits. Retrieval from the incident DFRs does not commence until all fault records have been retrieved from the current DFR.

When FRI receives an Incident *inform* message, it recognises that a disturbance has occurred and begins the interaction sequence depicted in SD_PEDA_19 in Appendix

F.19 to obtain incident fault records. This interaction sequence is initiated for each DFR on the incident circuit. Incident related fault records are not yet available, so FRI sends *request* messages to FRR requesting retrieval of fault records from each DFR on the incident circuit. These *request* messages may seem unnecessary since we know that FRR is already aware of the incident and is in the process of retrieving any available fault records from DFRs on the incident circuits. However, in reality FRI has no knowledge of whether FRR is aware of the incident and indeed what FRR is intending on doing about it. To ensure FRI obtains the fault records required to fulfil its role in PEDA it must, therefore, request fault record retrieval.

PVD also recognises that a disturbance has occurred when an Incident *inform* message is received and begins the interaction sequence depicted in SD_PEDA_20 in Appendix F.20 to obtain incident interpreted fault records. Again, this interaction sequence is initiated for each DFR on the incident circuit. Incident related interpreted fault records are not yet available, so PVD sends *request* messages requesting interpretation of fault records from each DFR on the incident circuit between the incident start and finish times. Similarly to FRI having no knowledge of whether FRR is aware of the incident, PVD has no knowledge of whether FRI is aware of the incident and must, therefore, request fault record interpretation.

The automatic dissemination of the Incident facts by IEI has resulted in the other PEDA agents being made aware of the disturbances and begun prioritisation of disturbance data retrieval and interpretation: the final stages of the manual disturbance diagnosis process conducted by protection engineers. Furthermore, the sending of *request* messages by PVD and FRI will ensure prioritised fault record retrieval, fault record interpretation and protection validation even if FRR and FRI failed to receive the Incident *inform* messages from IEI.

At this point in the case study, IEI has completed its role in disturbance diagnosis and resumed interpretation of SCADA alarms, FRR is beginning the retrieval of disturbance related fault records, FRI is waiting to be informed of FaultRecord facts by FRR and PVD is awaiting disturbance related InterpretedFaultRecord facts.

Dissemination of disturbance fault records

As fault records are retrieved from DFRs on the incident circuits they are added to FRRs' agent memory as FaultRecord facts. The disturbance related fault records retrieved by FRR are illustrated in Figures 8-8 to 8-11.

FRR will have received a *request* message from FRI for each of the four DFRs related to the incidents. As retrieval from each DFR is completed, a *confirm* message is sent to FRI using the message id from the original request indicating that retrieval is complete and all available fault records have been retrieved from the DFR.

Upon receipt of each *confirm* message, FRI continues with the interaction sequence depicted in SD_PEDA_19 and constructs a *query-ref* message for the FaultRecord facts between the incident start and finish time appropriate to the DFR from which retrieval has been completed. These *query-ref* messages are sent to FRR which responds with *inform* messages containing the FaultRecord facts matching the queries. This ensures only FaultRecord facts generated during the incidents are obtained and prioritised for interpretation. All other fault records retrieved from the DFR are of lower priority and are automatically forwarded to FRI by FRR in response to the earlier subscription for FaultRecord facts.

When a FaultRecord *inform* message with the in-reply-to field matching the id of a sent *query-ref* message is received FRI knows that the FaultRecord facts contained within the content field of the FIPA *inform* message are related to an incident. The contained facts are added to the interpretation schedule with the earliest FaultRecord fact being added to the top of the interpretation schedule.

The interactions between FRI and FRR following incident notification have resulted in the prioritised retrieval and collation of disturbance related fault records. This collaboration facilitates the prioritised interpretation of disturbance fault records and the later timely validation of protection performance by PVD.

At this point in the case study, IEI is continuing to interpret SCADA alarms, FRR has resumed normal autopolling, FRI is about to begin interpreting disturbance related fault records and PVD is awaiting the provision of disturbance related InterpretedFaultRecord facts by FRI.

Dissemination of disturbance interpreted fault records

When the fault record interpretation software embedded within FRI has finished its current interpretation, FRI selects the next FaultRecord fact from the top of the interpretation schedule. The fact is input into the wrapper controlling the embedded fault record interpretation software and the interpretation results output to agent memory as an InterpretedFaultRecord fact. Only when FRI has interpreted all the FaultRecord facts related to an earlier interpretation request from PVD will a *confirm* message be sent indicating completion of the request.

Upon receipt of each *confirm* message, PVD continues with the interaction sequence depicted in SD_PEDA_20 and constructs a *query-ref* message for the InterpretedFaultRecord facts between the incident start and finish time appropriate to the DFR from which fault record interpretation has been completed. This *query-ref* message is sent to FRI which responds with an *inform* message containing the InterpretedFaultRecord facts matching the query. The received facts are added to the validation schedule and grouped with their related Incident fact.

Only when a *confirm* message in response to each of the original *request* messages and an *inform* message in response to each *query-ref* message sent to FRI have been received will the incident be scheduled for protection validation. Note that there is no guarantee that all of the inform messages received by FRI will contain FaultRecord facts. Nevertheless, providing one or more FaultRecord fact is received, validation should be attempted since it may be possible to gain valuable information from a partial validation using only interpreted fault records from one circuit end.

At this point in the case study, IEI is continuing to interpret SCADA alarms, FRR is continuing normal autopolling, FRI has finished interpreting disturbance related fault records and resumed interpretation of fault records retrieved through autopolling and PVD has received InterpretedFaultRecord facts from each of the DFRs on the incident circuits and is ready to proceed with protection validation.

PVD selects the first of the incidents ready for validation from the validation schedule. Using the Incident fact, the wrapper around the embedded protection validation toolkit selects the circuit protection scheme and uploads the component protection models. The wrapper then inputs the InterpretedFaultRecord facts into the

embedded software and runs the toolkit. The generated protection validation report is output to the agent's memory as a ProtectionValidationReport fact. This process is repeated for the remaining incident in the validation schedule.

The collaborations between PVD and FRI following incident notification have resulted in the collation of the interpreted fault records necessary for protection validation. The agents within PEDA have now completed their role in disturbance diagnosis and resumed their post-startup tasks.

8.6.3 Disturbance Data and Information Generated

The preceding section has described how the functional and interaction tasks within each agent have enabled PEDA to retrieve, interpret and gather the case study SCADA and DFR data presented in section 8.6.1. To assess the disturbance analysis capabilities of PEDA, it is necessary to present the disturbance data and information PEDA would make available to protection engineers given the agent interactions and reasoning described in the preceding section.

It is anticipated that a PEDA user interface would collate and compile the disturbance information generated by PEDA into disturbance reports for perusal by a protection engineer. The information contained within the disturbance reports would be obtained via *inform* messages received by the PEDA user interface agent from IEI, FRR, FRI and PVD. However, given that the PEDA user interface has not advanced beyond the prototype stage, no actual disturbance reports have been generated for the case study. Nonetheless, two disturbance reports have been mocked up using the disturbance data and information produced by PEDA and collated by the prototype user interface: Figures 8-16 and 8-17. Although the look and feel of these disturbance reports will undoubtedly change slightly when a PEDA user interface is eventually implemented, the information contained within them will not.

Presented with the disturbance report in Figure 8-16, the protection engineer would immediately glean from the disturbance summary that the protection scheme on the SUBC4 / SUBA has begun an operating sequence at 14:20:38.97 and that an autoswitching sequence had been completed by 14:20:39.28. The PEDA user

interface would have received this information via an Incident *inform* message from IEI.

POST-FAULT DISTURBANCE REPORT	
Start: 14:20:38:97 **Finish:** 14:20:39.28	**SECOND MAIN PROT OPTD – SUBC4 / SUBA** **Autoswitching Sequence Complete**
Phases Affected:	Red-Yellow
Clearance Time:	69ms
High Level Protection Events	1st and 2nd Main Protection operated successfully at SUBC4 > SUBA 1st and 2nd Intertrips received at both ends 1st and 2nd Main Protection operated successfully at SUBA4 > SUBC
Results of Protection Validation & Diagnosis	All the components operated within the bounds predicted by the models
Detail on Protection Events	
14:20:38:97	2nd Main Protection Operated ON at SUBC4
14:20:38:98	1st Main Protection Operated ON at SUBC4
14:20:39:02	2nd Main Protection Operated OFF at SUBC4
14:20:39:02	SUBC4 Circuit Breaker CB1 OPEN
14:20:39:03	1st Intertrip Received ON at SUBC4 from SUBA
14:20:39:05	2nd Intertrip Received ON at SUBC4 from SUBA
14:20:39:05	Autoswitching in Progress at SUBC4 SUBA
14:20:39:07	2nd Main Protection Operated ON at SUBA4
14:20:39:07	1st Main Protection Operated ON at SUBA4
14:20:39:09	1st Main Protection Operated OFF at SUBC4
14:20:39:10	Autoswitching in Progress at SUBA4 CB2
14:20:39:11	1st Intertrip Received ON at SUBA4 from SUBC
14:20:39:13	SUBA4 Circuit Breaker CB2 OPEN
14:20:39:13	2nd Intertrip Received ON at SUBA4 from SUBC
14:20:39:15	2nd Main Protection Operated OFF at SUBA4
14:20:39:16	SUBA2 Circuit Breaker CB1 OPEN
14:20:39:16	SUBA2 Circuit Breaker CB3 OPEN
14:20:39:18	1st Main Protection Operated OFF at SUBA4
14:20:39:24	Autoswitching in Progress at SUBA4 SUBC
14:20:39:28	Autoswitching Complete at SUBA4 CB2
14:20:39:28	All tripped circuit breakers did NOT close
14:20:39:28	SUBC4 / SUBA circuit was not restored by end of incident. Time elapsed = 0m 0s 260ms
14:20:39:28	Autoswitching Sequence at SUBA4 CB2 took 0m 0s 180ms

Figure 8-16 Post-fault disturbance report for first disturbance at 14:20:38:97

POST-FAULT DISTURBANCE REPORT	
Start: 14:20:39:07 **Finish:** 14:20:39.28	**SECOND MAIN PROT OPTD – SUBA4 / SUBB** **Autoswitching Sequence Complete**
Phases Affected:	Yellow – Earth
Clearance Time:	66ms
High Level Protection Events	1st and 2nd Main Protection operated successfully at SUBA4 > SUBB 1st Main Protection operated successfully at SUBB4 > SUBA 1st and 2nd Intertrips received at both ends 2nd Main Protection at SUBB4 > SUBA failed to operate
Results of Protection Validation & Diagnosis	One or more protection scheme components may have malfunctioned. SUBA TR_1 may have malfunctioned.

Detail on Protection Events	
14:20:39:07	2nd Main Protection Operated ON at SUBA4
14:20:39:07	1st Main Protection Operated ON at SUBA4
14:20:39:09	1st Main Protection Operated ON at SUBB4
14:20:39:10	2nd Intertrip Received ON at SUBA4 from SUBB
14:20:39:10	Autoswitching in Progress at SUBA4 CB2
14:20:39.12	1st Intertrip Received ON at SUBA4 from SUBB
14:20:39:13	SUBA4 Circuit Breaker CB2 OPEN
14:20:39:16	2nd Main Protection Operated OFF at SUBA4
14:20:39:16	SUBA2 Circuit Breaker CB1 OPEN
14:20:39:16	SUBA2 Circuit Breaker CB3 OPEN
14:20:39:17	1st Main Protection Operated OFF at SUBA4
14:20:39:21	2nd Intertrip Received ON at SUBB4 from SUBA
14:20:39:21	1st Intertrip Received ON at SUBB4 from SUBA
14:20:39:23	Autoswitching in Progress at SUBB4 CB1
14:20:39:24	SUBB4 Circuit Breaker CB1 OPEN
14:20:39:25	1st Main Protection Operated OFF at SUBB
14:20:39:28	Autoswitching Complete at SUBA4 CB2
14:20:39:28	All tripped circuit breakers did NOT close
14:20:39:28	SUBA4 / SUBB circuit was not restored by end of incident. Time elapsed = 0m 0s 150ms
14:20:39:28	Autoswitching Sequence at SUBA4 CB2 took 0m 0s 180ms
14:20:39:07	2nd Main Protection Operated ON at SUBA4

Figure 8-17 Post-fault disturbance report for second disturbance at 14:20:39:07

Knowing that a protection operation had occurred, the protection engineer would next be interested in whether the protection operated in response to a fault or had operated without any fault on the circuit, i.e. a possible mal-operation. This can be determined from the next section in the disturbance report, derived by the PEDA user interface from InterpretedFaultRecord *inform* messages received from FRI.

The 'Phases Affected' and 'Clearance Time' sections indicate that a Red-Yellow phase fault had occurred and was cleared by the protection in 69ms confirming that a fault had indeed occurred. The protection engineer can now look to the next section of the disturbance report for a high-level summary of how the protection scheme responded to the fault.

Examining the high-level protection events the protection engineer would note that the protection scheme on the SUBA4 / SUBC circuit operated correctly, with both 1st and 2nd main protection operating at each circuit end and 1st and 2nd intertrips being receive at each circuit end. These events will have been received by the PEDA user interface from IEI as multiple Event *inform* messages.

At this stage, the protection engineer may decide not to progress any further with the disturbance report since the high-level event summaries indicate that everything within the protection scheme operated as expected. However, the 'Detail on Protection Events' section of the disturbance report would provide useful additional information on what protection scheme components actually operated and when. The event information contained within this section is derived from Event *inform* messages received by the PEDA user interface from IEI.

Looking at the events presented in the 'Detail on Protection Events' section of the report, a non-protection engineer may consider it a contradiction that autoswitching can be complete and all trip circuit breakers were not closed and the circuit not restored by the end of the incident. However, it must be remembered that, to a protection engineer, an incident is concluded when the protection scheme has completed its sequence and not when the circuit is restored. In the case of autoswitching, the circuit is not restored until 10-15 seconds have elapsed following completion of the protection and autoswitching sequence. The closure of the circuit

breakers are, therefore, not within the timeframe of the incident identified by the Telemetry Processor embedded within IEI.

On examination of the second disturbance report in Figure 8-17 the protection engineer would be immediately aware that the protection scheme on another feeder emanating from SUBA4, this time SUBA4 to SUBB, had operated 100ms after the first disturbance. Furthermore, the phases affected information would indicate that a Yellow-Earth fault had occurred, ruling out the possibility that the protection had mal-operated when no fault was present.

Examination of the high-level protection events for this second disturbance would also indicate that the 2nd main protection failed to operate at the SUBB4 circuit end. This is borne out by the absence of a "2nd Main Protection Operated ON at SUBB4" event in the detail on protection events section of the disturbance report. Given that the presence of a fault had been confirmed from the phases affected information, the protection engineer could now rule out the possibility of 1st main protection mal-operation instead reaching the conclusion that 2nd main protection at SUBB4 had indeed failed. This would be noted as requiring further investigation.

Moving onto the results of the protection validation and diagnosis, the protection engineer would be made aware that trip relay 1 (TR_1) at SUBA may have mal-functioned. The PVD agent would have generated this diagnosis by validating how the protection scheme should have operated against how it actually operated as recorded in the interpreted fault records retrieved from FRI by PVD.

It is highly likely that the protection engineer would now turn to the PEDA user interface and view the fault records pertaining to the disturbance (re: Figures 8-9 and Figures 8-10). Analysis of the fault records from both substations would not only confirm that the 2nd main protection had failed to operate at SUBB4 (re: Figure 8-10) but also that trip relay 1 at SUBA had indeed failed to operate (re: the SUB_B 1MPTR digital channel in Figure 8-9).

Returning to the disturbance report, the protection engineer can view a record of the pertinent protection scheme operations in the 'Detailed Protection Events' section of the report. Examination of these events would confirm the conclusions reached by the high-level event summaries.

In summary, from the disturbance information gathered by PEDA, it would be clear to the protection engineer that two almost simultaneous disturbances occurred and that the protection scheme on the SUBA4 / SUBB circuit did not operate as would be expected. Although the protection scheme did operate and clear the fault, it highlights a potential problem which casts doubt on whether the protection scheme would correctly isolate a future fault on the circuit. The protection engineer can use the disturbance information gathered by PEDA as a basis for further investigations into the failure of the 2nd main protection at SUBB and trip relay 1(TR1) at SUBA.

It should also be noted that during a storm in excess of 100 disturbances may occur and require analysis by the protection engineer. Without PEDA, the protection engineer would have to resort to a manual approach with the increased risk of overlooking the protection failures identified in the case study. As a consequence, the network is at greater risk of a significant blackout, e.g. if the failure of trip relay 1 at SUBA4 had gone unnoticed and there was a subsequent failure of trip relay 2 during another disturbance, tripping of the circuit breakers at SUBA4 would not be possible and a larger network area would need to be isolated to clear the fault.

8.6.4 Performance Assessment

Evaluation of PEDA's disturbance analysis capabilities is only truly complete, if the real-time performance of PEDA is assessed. To this end, the PEDA architecture was deployed within a laboratory environment in a number of deployment configurations ranging from all agents being distributed across different PCs to all agents running on the same PC. In each configuration, the real-time performance of PEDA was assessed using the case study presented in section 8.6.1.

To simulate the real-time feed of SCADA alarms, a software tool was developed to input the case study SCADA alarms into the alarm database monitored by the IEI agent at the same rate as they would arrive in an operational context. Simulation of the fault record retrieval was, however, more difficult to achieve since the prototype version of FRR did not have the communications functionality necessary to dialup and communicate with remote DFRs. Nevertheless, the prototype FRR agent was capable of simulating differing retrieval times by imposing delays of 30 seconds to 2

minutes between retrieval being initiated and the case study fault record being available within the FRR agent as a FaultRecord fact.

Running the case study through PEDA for each deployment configuration it was found that the Telemetry Processor wrapped within IEI would generate an Incident summary and make it available for IEI to disseminate to other agents less than 1 second after the incident had been concluded. Furthermore, the subscription process necessary for other agents to obtain updates of Incident, FaultRecord and InterpretedFaultRecord information took, on average 162ms. Finally, the time taken between the first incident being identified by IEI and PVD generating the second protection validation report, thereby completing the disturbance analysis for the case study, was found to range from 1 minute 17 seconds to 3 minutes 23 seconds. The variations in disturbance analysis time can be attributed to: differing levels of network traffic delaying the communications between agents, differing loadings of CPU's depending on how many agents were running on the one PC and the differing simulated delays in fault record retrieval. All these factors would also be present in a real-time operational environment.

If PEDA is to function in a real-time operational environment it must be flexible enough to cope with loss of communications to agents and agents being stopped and restarted by system administrators to facilitate reconfiguration of the architecture and maintenance of PCs. To assess the flexibility of the architecture, communications between the IEI and FRI agents were temporarily interrupted by physically disconnecting and reconnecting the network cables between the PCs running each agent. Monitoring of the communications prior, during and post communications interruption showed that both the IEI and FRI agents continued to function and that, although incident *inform* messages failed to be sent during the communications interruption, those sent after restoration of communications were successfully received by FRI. An improvement to each agent so it can make several attempts at sending messages is scheduled for the next stage in PEDA development.

Flexibility was also assessed by stopping FRI running and restarting it on another machine. This test proved successful with the new instance of FRI subscribing to IEI and IEI sending Incident *inform* messages to the new agent.

To assess the extensibility of PEDA three instances of the FRR agent were created and started at different times on different PCs increasing the number of PEDA agents to eight and mimicking the introduction of agents into the architecture. Each agent successfully started, registered with the Nameserver, identified from the Facilitator a provider of Incident information and subscribed to IEI for Incident updates. IEI also successfully informed each of the three FRR agents about new Incident information. This approach not only proved that PEDA was indeed extensible but also represented a realistic scenario where multiple FRR agents, each dedicated to the retrieval of fault records from a particular network area, could be used to reduce the fault record retrieval burden on the present FRR agent. The use of multiple FRR agents is seen as one of the next major enhancements to the PEDA architecture.

8.7 Discussion

The preceding sections have described how a MAS for automating post-fault disturbance analysis has been realised through implementation of the agents identified in the PEDA specification. The disturbance analysis capabilities and performance of the PEDA MAS have also been evaluated.

Using the agent shells provided by the Zeus agent building toolkit and through the addition of reactive and algorithmic functions existing standalone software tools have been given autonomy. These tools, previously used by protection engineers during disturbance analysis and now operating as agents within PEDA, can now collaborate and work collectively to monitor for disturbances on the transmission network and generate the disturbance information pertinent to protection engineers. This is a significant advance from the previous manual approach to disturbance analysis for a number of reasons:

- Fault record retrieval is prioritised, thereby reducing the risk of critical disturbance related fault records being overwritten during storm conditions.
- Protection engineers no longer require knowledge of how each individual software tool operates.

- Protection engineers no longer need to transfer retrieved disturbance data or information generated by a tool to another one for further analysis.
- As shown by the performance evaluation in section 8.6.3, all disturbance data is retrieved, interpreted and archived less than five minutes after the disturbance has occurred, ready for the protection engineer to begin post-fault disturbance analysis.

Although the underlying PEDA architecture is flexible, extensible and capable of fast automated post-fault disturbance analysis there is no means of informing a protection engineer that disturbances have occurred. The prototype user interface does gather the disturbance information and the final implementation of the user interface will generate disturbance reports. However, the protection engineer is only made aware of disturbances either through regular checking of the PEDA user interface or by conversations with other company personnel such as control engineers and field engineers. Significant delays can therefore result between a disturbance occurring and the protection engineer viewing the available disturbance information. The level of disturbance diagnosis assistance provided to protection engineers would be greatly enhanced if the PEDA user interface agent could manage the pro-active notification and dissemination of generated disturbance information to interested parties, principally protection engineers.

This new enhanced PEDA user interface agent would have to maintain a user profile for each protection engineer specifying their contact details, preferred notification method and types of data and information of interest. A more efficient approach would be to assign each protection engineer a unique agent devoted to the gathering and notification of disturbance information pertinent to the individual. This is similar to the concept of personal assistant agents proposed by Maes [7].

Personal assistant agents are another class of agents that act semi-autonomously for and on the behalf of a user, modelling the interests of the user and providing services to the user as and when required [8]. They are intended primarily to act as virtual support staff, accomplishing routine support tasks allowing the user to concentrate on their principal job function. These support tasks may relate to automatic filtering of

email messages or to the monitoring of Internet news sites for news of interest to the user. The generic model for a personal assistant agent is illustrated in Figure 8-18.

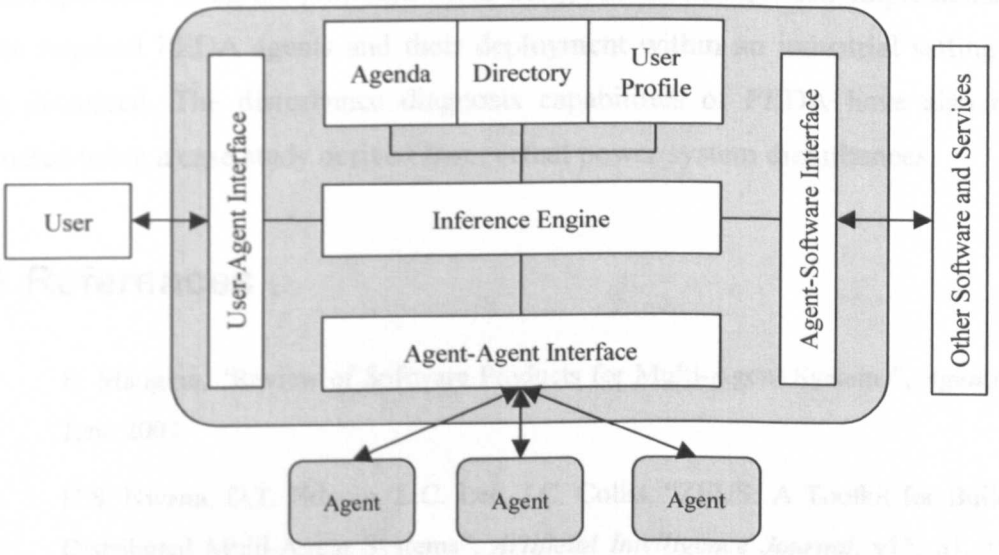


Figure 8-18 FIPA Personal Assistant Model [8]

Within PEDA the personal assistant agents will more than likely take a similar form to the engineering assistant agents proposed by Mangina [9] and used within the COMMAS MAS. These engineering assistant agents provide each engineer with a user interface to facilitate communications between the user and the COMMAS software system, in an intelligent manner, so that appropriate information is given to the users based on their profiles. The PEDA engineering assistant agents would perform the same functionality and gather disturbance information directly from the other PEDA agents through interactions.

Critical to the success of engineering assistant agents is the ability of a MAS, such as PEDA, to accommodate the introduction of engineering assistant agents to the architecture during runtime. Only with a flexible and extensible architecture, can new engineering assistant agents be introduced with ease and the utility of the architecture increased. The flexible and extensible architecture offered by PEDA is ideal for accommodating engineering assistant agents. The research and development of these agents is seen as one of the next major steps in PEDA development.

8.8 Chapter Summary

This chapter has described the realisation of the PEDA multi-agent engineering system specified using the proposed MAS design methodology. The implementation of the required PEDA agents and their deployment within an industrial setting has been discussed. The disturbance diagnosis capabilities of PEDA have also been evaluated using a case study derived from actual power system disturbances.

8.9 References

[1]. E. Mangina, "Review of Software Products for Multi-Agent Systems", *AgentLink*, June 2002.

[2]. H.S. Nwana, D.T. Ndumu, L.C. Lee, J.C. Collis, "ZEUS: A Toolkit for Building Distributed Multi-Agent Systems", *Artificial Intelligence Journal*, v13, n1, 1999, pp129-186.

[3]. Jack Intelligent Agents, http://www.agent-software.co.uk

[4]. FIPA-OS, http://fipa-os.sourceforge.net/index.htm

[5]. "FIPA Communicative Act Library Specification", XC00037H, [Online], Available: http://www.fipa.org/repository/index.html

[6]. "FIPA SL Content Language Specification", XC00008G, [Online], Available: http://www.fipa.org/repository/index.html

[7]. P. Maes, "Agents that Reduce Work and Information Overload", Communications of the ACM, v37, n7, pp 31-40, 1994

[8]. "FIPA Personal Assistant Specification", XC00083B, [Online], Available: http://www.fipa.org/repository/index.html

[9]. E. Mangina, "Agent-Based Approach for Intelligent Data Interpretation within Monitoring Applications", PhD thesis, University of Strathclyde, Department of Electronic and Electrical Engineering, 2002.

Chapter 9: Conclusions and Future Work

9.1 Conclusions

The research presented in this thesis has dealt with applying a multi-agent approach to the automation of post-fault disturbance analysis in order to provide levels of decision support beyond those currently experienced by ScottishPower protection engineers. Such decision support enhancements are required for a number of reasons:

- The number of protection engineers skilled in the analysis of power system disturbances has been reduced due to the rationalisation of operational functions within the power industry. This emphasises the need for sophisticated data interpretation systems to simulate the protection engineers' approach to post-fault disturbance analysis.
- The availability of data from transmission network monitoring devices is increasing in quantity, quality and scope resulting in the few remaining protection engineers experiencing data overload.
- With the deployment of new network monitoring technologies comes new data retrieval and analysis software which the protection engineer must be adept at using. This places additional strains on the limited pool of protection engineers.
- Although decision support tools have been developed, none focus on the post-fault disturbance analysis task in its entirety. Those decision support tools that do exist focus on interpretation of data from a particular source, e.g. SCADA, and have been adapted from control room applications. The protection engineer must therefore further interpret the output of these systems to generate information pertinent to the post-fault disturbance analysis task at hand.

A particular problem with the manual post-fault disturbance analysis process, as detailed in chapter two, is that it is both time-consuming and laborious. Automation of this manual process can realise true engineering benefits and savings:

- Automation eliminates the human error inherent in manual analysis which can result in incipient protection failures being missed, particularly where large data volumes are interpreted, e.g. following storms. The risk of protection mal-

operations being overlooked and becoming actual protection failures is therefore reduced and the overall transmission network security enhanced.

- Disturbances are not predictable and the scheduled protection design, commissioning and maintenance tasks performed by engineers often have to be rescheduled to enable post-fault disturbance analysis. Automating the data retrieval, interpretation and collation process lifts this burden from the protection engineers allowing the engineers to spend more time on other activities.
- Using an automated approach to disturbance analysis disturbance reports can be generated providing detail on the disturbance duration, network area affected and plant involved. Other utility personnel can benefit from these disturbance reports, such as control engineers, asset managers and customer services personnel.

As described in chapter two, post-fault disturbance analysis requires the retrieval, interpretation and collation of data from a number of disparate network monitoring technologies. A number of software tools are available to assist the protection engineers with the retrieval and interpretation of data from these devices and have been described in chapter three. Although useful, the existing software tools only address a small sub-set of tasks required to complete post-fault disturbance analysis. In order to assist the protection engineer with the entire post-fault disturbance analysis process these existing tools must be augmented with new decision support tools and integrated into an integrated decision support architecture.

Within this thesis multi-agent systems (MAS) have been advocated as an effective means of realising such integrated architectures and, consequently, automated post-fault disturbance analysis. Adopting a multi-agent approach is of real engineering benefit for a number of reasons:

- Existing software systems need not be rewritten to be accommodated within a multi-agent architecture. Instead developers can create software 'wrappers' that encapsulate the legacy application and add the agent-based functionality. Where wrapping is not feasible, transducers can be developed to act as an interface between the existing system and the multi-agent environment.

- The multi-agent architecture is inherently modular facilitating the introduction and removal of software components without extensive modifications to the individual software components or the architecture itself. This facilitates incremental development and extensibility of the architecture thereby slowing obsolescence and prolonging the useful life of the decision support architecture.
- The multi-agent architecture also benefits from flexibility since communications are dynamic and there is no hard-wiring of software components to one another. The configuration of the architecture can, therefore, be changed and agents redeployed without any need for software modifications.
- The individual software components, or agents, can be deployed across a range of platforms and end-users can access the decision support information provided via industry accepted internet protocols from their desktops.

Given the real engineering benefits of adopting a multi-agent approach to systems integration, a methodology had to be created to enable the specification and design of MAS for decision support within the power industry. To this end, chapter five described a new methodology which not only encompassed the traditional aspects of MAS design, such as task decomposition and specification of agent interactions, but also addressed the engineering problems of designing a MAS for decision support within the power industry, i.e. reuse of legacy systems and specification of the agent behaviour functions necessary to achieve systems automation.

A key outcome of the reported research work is the successful application of the new methodology to the design of the novel PEDA multi-agent architecture for automated post-fault disturbance analysis. Chapter six has described how the methodology was used to specify the agents required with PEDA, the social abilities of each agent and the agent behaviour functions necessary to wrap the existing decision support tools with agent functionality. By following the methodology it was recognised that automated disturbance analysis could be realised by four core disturbance analysis agents performing the following functions: automated SCADA interpretation, prioritised retrieval of disturbance related fault records, prioritised interpretation of disturbance related fault records and automated validation of a protection schemes response to a disturbance.

In all but the agent responsible for prioritised fault record retrieval, the disturbance analysis capabilities of the PEDA agents have been realised through the reuse of existing software systems. The agent responsible for identifying disturbances through automated SCADA interpretation is of particular significance since it uses a novel alarm processor developed as part of the research presented in this thesis.

The Telemetry Processor, described in chapter four, has been developed as a standalone intelligent system specifically aimed at automating the post-fault interpretation of SCADA alarms. The novel reasoning architecture developed to emulate the multi-stage approach to post-fault SCADA interpretation followed by protection engineers has been described in detail and the performance of the architecture evaluated. The development of this novel reasoning architecture and its encapsulation within the Telemetry Processor provides protection engineers with rapid disturbance analysis.

A functional prototype of the PEDA architecture has also been implemented and its disturbance analysis capabilities assessed using actual power systems disturbances. Chapter eight has described the implementation of the architecture and the performance of PEDA for one particular case study. The performance evaluation presented in chapter eight not only demonstrated the speed by which PEDA can perform post-fault disturbance analysis but also described the flexibility and scalability in the architecture.

In summary then, the main contributions of this thesis include:

1. *The design, development and implementation of an intelligent system focussed on assisting protection engineers with post-fault SCADA interpretation.* Based on the alarm processing requirements of ScottishPower protection engineers a reasoning architecture capable of emulating the multi-stage reasoning process followed during post-fault SCADA interpretation has been developed. This architecture has been shown to function effectively and has been encapsulated within the Telemetry Processor deployed at ScottishPower providing real-time automated post-fault SCADA analysis decision support in an industrial setting.
2. *The creation of a methodology for the specification of MAS for decision support within the power industry using multi-agent technology.* Following a critique of

existing MAS design methodologies and assessment of their suitability for developing MAS for decision support within the power industry a new methodology has been created. By following this methodology a developer can ensure the reuse of legacy decision support tools and specify flexible and scalable decision support architectures.

3. *The design of a multi-agent architecture for providing post-fault disturbance analysis decision support assistance to protection engineers through integration and automation of existing software systems.* The creation of the new methodology facilitated the specification and design of a multi-agent architecture where the overall post-fault disturbance analysis task is decomposed into sub-tasks, which can be performed by intelligent agents. Based on the requirements of protection engineers and assessment of the existing decision support tools a specification for the PEDA multi-agent architecture was created.

4. *The implementation of a multi-agent architecture for automating post-fault disturbance analysis using hybrid-data interpretation techniques across a number of intelligent agents.* Using the PEDA specification, a prototype implementation of the PEDA multi-agent architecture has been created and the disturbance analysis functionality within each agent implemented. Across these agents data interpretation techniques appropriate to the disturbance analysis task at hand are utilised, mainly through reuse of existing decision support tools.

5. *Demonstration of the benefits adopting a multi-agent approach can bring to the integration of decision support systems through the evaluation of a multi-agent architecture using power system data generated from actual disturbances.* The prototype PEDA architecture has been evaluated using a complex case study derived from actual power system disturbances. It has been demonstrated that PEDA can retrieve, interpret, collate and archive all disturbance data within five minutes of a disturbance occurring ready for the protection engineers to begin post-fault disturbance analysis. Furthermore, the PEDA architecture has also been shown to be flexible and extensible.

The significant outcomes of this research project along with the benefits and opportunities associated with the development of decision support architectures using

multi-agent technology have been listed above. However, there are still a number of potential avenues for further work building upon the results of the PhD research.

9.2 Future Work

Potential further work required to extend and refine the research presented in this thesis, which in the author's opinion will prove to be particularly productive, includes:

- Introduction of the fully functional FRR agent and deployment of PEDA within an industrial setting. This would facilitate an extended evaluation period of both PEDA's disturbance analysis capabilities and its performance in a real-time operational environment.
- Application of the methodology to other areas of power engineering decision support, such as assisting control engineers with fault management and asset managers with targeting network investment. It may also be fruitful to assess the suitability of the methodology for specifying MAS for non-decision support applications such as condition monitoring.
- At present the PEDA architecture is dependent on the IEI agent's ability to provide the incident information necessary to prioritise fault record retrieval and interpretation and automate protection validation. To mitigate the risk of a failure of either IEI or the SCADA archive leading to loss of disturbance analysis capabilities additional IEI agents could be deployed and distributed across the power system, each capable of retrieving SCADA directly from RTUs.
- As it stands, the single FRR agent is a bottleneck within PEDA since it can only handle a limited number of simultaneous retrievals increasing the time taken to complete disturbance analysis. Multiple, more localised, FRR agents would solve this problem since their retrieval workload would be significantly less making them able to respond to fault record retrieval requests quicker than a single FRR agent. Splitting the fault record retrieval task assigned to FRR amongst a number of agents distributed across the power system would clearly be a worthwhile exercise.

- Extending the post-fault disturbance analysis capabilities of PEDA through the introduction of new PEDA agents designed to handle the retrieval and interpretation of data from other devices which provide useful information to the protection engineer, e.g. Travelling Wave Fault Locators, weather monitoring equipment and lightning detection equipment. Furthermore, the possibility of integrating PEDA with other power systems MAS, such as COMMAS, to further extend its disturbance analysis capabilities should also be explored.
- Advancing the PEDA user interface beyond the prototype stage with the creation of fully functional engineering assistant agents. As detailed in chapter eight, each engineer requiring access to PEDA would benefit from an engineering assistant agent dedicated to facilitating access to and displaying the disturbance information of interest to the particular user. Furthermore, the functionality of the engineering assistant agent could be extended beyond that of traditional user interfaces to include the automated notification and dissemination of new disturbance information to its user.

Appendix A: Telemetry Processor Rulebase

This appendix contains the rules derived from the alarm interpretation knowledge elicited from protection engineers and used by the Telemetry Processor to interpret SCADA alarms.

Only those rules used to generate the incidents and events in the case study in chapter four are presented. The total numbers of rules for each rulebase used by the current Telemetry Processor facility at ScottishPower PowerSystems are presented below:

- o Incident Start Rulebase: 1
- o Incident Conclusion Rulebase: 5
- o Low-level Event Rulebase: 57
- o High-level Event Rulebase: 16

A.1 Incident Start Identification Rules

The JESS rules present in the Incident Start Rulebase are as follows:

; * Rule IS_1 – incident_start**

```
(defrule incident_start
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)
          (date ?date01) (time ?time01) (ms_since1970 ?ms1970_01) (hundredths ?ms_01)
          (legend ?legend01&:(or (prot_alarm ?legend01) (general_inter ?legend01) (trip_alarm
          ?legend01)))
          (alarmstatus ?status01&:(alarm_on ?status01)))
    (alarm (substation ?sub02) (feeder ?feeder02)
          (date ?date) (time ?time) (ms_since1970 ?ms1970_02) (hundredths ?ms_02)
          (plantid ?plantid02) (plantstatus ?plantstatus02))
   (test (cb_open ?plantid02 ?plantstatus02))
   (test (eq (sub-string 1 3 ?sub01) (sub-string 1 3 ?sub02)))
   (test (before ?ms1970_01 ?ms_01 ?ms1970_02 ?ms_02))
   (test (limit ?ms1970_01 ?ms_01 ?ms1970_02 ?ms_02 1000))
   (test (call (fetch PROT_BUFFER) find_earliest_prot_trip ?sub01 ?circ101 ?circ201 ?feeder01
                                                          ?ms1970_01 ?ms01))
=>
   (call (fetch INC_START) add(new TelemetryProcessor.CoreClasses.IncidentIdentifier ?date01
   ?time01 ?ms1970_01 ?ms_01 ?sub01 ?circ101 ?circ201 ?feeder01 ?legend01)))
```

A.2 Incident Conclusion Identification Rules

The JESS rules present in the Incident Conclusion Rulebase are as follows:

```
; *** Rule IC_1 – autoswitching_complete
(defrule autoswitching_complete
  (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01) (date
        ?date01) (time ?time01) (ms_since1970 ?ms1970_01) (hundredths ?ms_01)
        (legend ?legend01&:(auto_prog ?legend01))
        (alarmstatus ?status01&:(alarm_on ?status01)))
  (alarm (substation ?sub02) (circuit1 ?circ102) (circuit2 ?circ202) (feeder ?feeder02)
        (date ?date02) (time ?time02) (ms_since1970 ?ms1970_02) (hundredths ?ms_02)
        (plantid ?plantid02) (plantstatus ?plantstatus02))
  (test (cb_open ?plantid02 ?plantstatus02))
  (test (call (fetch PLANT_OPS) earliest ?date02 ?time02 ?ms1970_02 ?ms_02 ?sub02
                                        ?plantid02 ?plantstatus02))
  (timing (date ?date_x) (time ?time_x) (ms1970 ?ms1970_x) (hundredths ?ms_x))
  (test (wait ?ms1970_02 ?ms_02  ?ms1970_x ?ms_x  60000))
  (alarm (substation ?sub03) (circuit1 ?circ103) (circuit2 ?circ203) (feeder ?feeder03) (date
        ?date03) (time ?time03) (ms_since1970 ?ms1970_03) (hundredths ?ms_03)
        (legend?legend03&:(auto_complete ?legend03))
        (alarmstatus ?status03&:(alarm_off ?status03)))
 (test (call (fetch PROT_OPS) latest_comp ?date03 ?time03 ?ms1970_03 ?ms_03 ?sub03
                                        ?circ1_03 ?circ2_03 ?legend_03 ?status_03))
  (test (limit ?ms1970_02 ?ms_02 ?ms1970_03 ?ms_03 60000))
=>
  (store CLOSED_STATUS true)
  (bind ?summary (str-cat (call (fetch IDENTIFIER) getStarter) " - " (call (fetch
                IDENTIFIER) getSub) " / " (call (fetch IDENTIFIER) getCirc1) "
                Autoswitching Sequence Complete"))
  (store FINISH_RULE ?summary))
```

A.3 Low-Level Event Rules

The JESS rules present in the Low-level Events Rulebase are as follows:

; *** Rule LE_1 – First Main Protection Operation ON

```
(defrule 1st_first_prot_on
   (first)
   (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) (circuit2 ?circ201)
          (feeder ?feeder01) (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
          (ms_since1970 ?ms1970_01) (legend ?legend01&:(prot_first ?legend01))
          (alarmstatus ?status01&:(alarm_on ?status01)))
=>
   (bind ?rule_summary "1st Main Protection Operated ON")
   (bind ?conclusion (str-cat "1st Main Protection Operated ON at " ?sub01))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
                       (pass_id 1) (rule_id "1-1") (voltage ?volt01) (summary ?rule_summary)
                       (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                    ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01
                                    ?hundredths01 ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion
                                    ?rule_summary "1-1" 1 1)))
```

; *** Rule LE_2 – First Main Protection Operation OFF

```
(defrule 1st_first_prot_off
   (first)
   (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) (circuit2 ?circ201)
          (feeder ?feeder01) (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
          (ms_since1970 ?ms1970_01) (legend ?legend01&:(prot_first ?legend01))
          (alarmstatus ?status01&:(alarm_off ?status01)))
=>
   (bind ?rule_summary "1st Main Protection Operated OFF")
   (bind ?conclusion (str-cat "1st Main Protection Operated OFF at " ?sub01))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
                       (pass_id 1) (rule_id "1-2") (voltage ?volt01) (summary ?rule_summary)
                       (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                    ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01
                                    ?hundredths01 ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion
                                    ?rule_summary "1-2" 1 1)))
```

; *** Rule LE_3 – Second Main Protection Operation ON

```
(defrule 1st_second_prot_on
    (first)
    (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) circuit2 ?circ201)
           (feeder ?feeder01) (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
           (ms_since1970 ?ms1970_01) (legend ?legend01&:(prot_first ?legend01))
           (alarmstatus ?status01&:(alarm_on ?status01)))
 =>
    (bind ?rule_summary "2nd Main Protection Operated ON")
    (bind ?conclusion (str-cat "2nd Main Protection Operated ON at " ?sub01))
    (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                        (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
                        (pass_id 1) (rule_id "1-3") (voltage ?volt01) (summary ?rule_summary)
                        (conclusion ?conclusion)))
    (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                     ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01
                                     ?hundredths01 ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion
                                     ?rule_summary "1-3" 1 1)))
```

; *** Rule LE_4 – Second Main Protection Operation OFF

```
(defrule 1st_second_prot_off
    (first)
    (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) (circuit2 ?circ201)
           (feeder ?feeder01) (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
           (ms_since1970 ?ms1970_01) (legend ?legend01&:(prot_first ?legend01))
           (alarmstatus ?status01&:(alarm_off ?status01)))
=>
    (bind ?rule_summary "2nd Main Protection Operated OFF")
    (bind ?conclusion (str-cat "2nd Main Protection Operated OFF at " ?sub01))
    (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01) (sub
                        ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 1)
                        (rule_id "1-4") (voltage ?volt01) (summary ?rule_summary) (conclusion
                        ?conclusion)))
    (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                     ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01
                                     ?hundredths01 ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion
                                     ?rule_summary "1-4" 1 1)))
```

; *** Rule LE_5 – First Intertrip ON

```
(defrule 1st_first_intertrip_on
    (first)
    (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)
           (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
           (ms_since1970 ?ms1970_01) (legend ?legend01&:(first_inter ?legend01))
           (alarmstatus ?status01&:(alarm_on ?status01)))
=>
    (bind ?rule_summary (str-cat "1st Intertrip Received ON"))
    (bind ?conclusion (str-cat "1st Intertrip Received ON at " ?sub01 " from " ?circ101))
    (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01)
                        (ms1970 ?ms1970_01) (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id
                        1) (rule_id "1-5") (voltage ?volt01)
                        (summary ?rule_summary) (conclusion ?conclusion)))
    (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                     ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01 ?ms_01
                                     ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion ?rule_summary
                                     "1-5" 1 1)))
```

; *** Rule LE_6 – First Intertrip OFF

```
(defrule 1st_first_intertrip_off
   (first)
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)
          (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
          (ms_since1970 ?ms1970_01) (legend ?legend01&:(first_inter ?legend01))
          (alarmstatus ?status01&:(alarm_off ?status01)))
=>
   (bind ?rule_summary (str-cat "1st Intertrip Received OFF"))
   (bind ?conclusion (str-cat "1st Intertrip Received OFF at " ?sub01 " from " ?circ101))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 1) (rule_id "1-6")
                       (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                               ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01
                               ?circ101 ?circ201 ?volt01 x ?conclusion ?rule_summary "1-6" 1 1)))
```

; *** Rule LE_7 – Second Intertrip ON

```
(defrule 1st_second_intertrip_on
   (first)
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)
          (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
          (ms_since1970 ?ms1970_01) (legend ?legend01&:(first_inter ?legend01))
          (alarmstatus ?status01&:(alarm_on ?status01)))
=>
   (bind ?rule_summary (str-cat "2nd Intertrip Received ON"))
   (bind ?conclusion (str-cat "2nd Intertrip Received ON at " ?sub01 " from " ?circ101))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 1) (rule_id "1-7")
                       (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                               ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01
                               ?circ101 ?circ201 ?volt01 x ?conclusion ?rule_summary "1-7" 1 1)))
```

; *** Rule LE_8– Second Intertrip OFF

```
(defrule 1st_second_intertrip_off
   (first)
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)
          (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
          (ms_since1970 ?ms1970_01) (legend ?legend01&:(first_inter ?legend01))
          (alarmstatus ?status01&:(alarm_off ?status01)))
=>
   (bind ?rule_summary (str-cat "2nd Intertrip Received OFF"))
   (bind ?conclusion (str-cat "2nd Intertrip Received OFF at " ?sub01 " from " ?circ101))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 1) (rule_id "1-7")
                       (voltage ?volt01)  (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                               ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01
                               ?circ101 ?circ201 ?volt01 x ?conclusion ?rule_summary "1-7" 1 1)))
```

```
; *** Rule LE_8 – Autoswitching in Progress ON

(defrule 1st_auto_prog_on
   (first)
(alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)

(voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01) (plantid ?plantid01)

(ms_since1970 ?ms1970_01) (legend ?legend01&:(auto_prog ?legend01))

(alarmstatus ?status01&:(alarm_on ?status01)))

=>
   (bind ?rule_summary (str-cat "Autoswitching in Progress ON"))
   (bind ?conclusion (str-cat "Autoswitching in Progress at " ?sub01 " " ?circ101 " " ?plantid01))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 1) (rule_id "1-8")
                       (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                ?ms1970_01 ?hundredths01 ?date01 ?time01 ?ms1970_01 ?ms_01
                                ?sub01 ?circ101 ?circ201 ?volt01 ?plantid01 ?conclusion
                                ?rule_summary "1-8" 1 1)))


; *** Rule LE_9 – Autoswitching Complete OFF

(defrule 1st_auto_complete_off
   (first)
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)
          (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01) (plantid ?plantid01)
          (ms_since1970 ?ms1970_01) (legend ?legend01&:(auto_complete ?legend01))
          (alarmstatus ?status01&:(alarm_off ?status01)))
=>
   (bind ?rule_summary (str-cat "Autoswitching in Complete OFF"))
   (bind ?conclusion (str-cat "Autoswitching in Complete at " ?sub01 " " ?circ101 " " ?plantid01))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 1) (rule_id "1-9")
                       (voltage ?volt01)  (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                ?ms1970_01 ?hundredths01 ?date01 ?time01 ?ms1970_01 ?ms_01
                                ?sub01 ?circ101 ?circ201 ?volt01 ?plantid01 ?conclusion
                                ?rule_summary "1-9" 1 1)))
```

; *** Rule LE_10 – Circuit Breaker OPEN

```
(defrule 1st_cb_open_alarm
   (first)
   (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) (circuit2 ?circ201)
          (feeder ?feeder01) (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
          (ms_since1970 ?ms1970_01) (legend ?legend01) (plantid ?plantid01)
          (plantstatus ?plantstatus01))
   (test (cb_open ?plantid01 ?plantstatus01))
=>
   (bind ?rule_summary (str-cat "Circuit Breaker OPEN"))
   (bind ?conclusion (str-cat ?sub01 "  Circuit Breaker " ?plantid01 " OPEN"))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
                       (plantid ?plantid01) (plantstatus ?plantstatus01) (pass_id 1) (rule_id "1-10")
                       (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                ?ms1970_01 ?hundredths01 ?date01 ?time01 ?ms1970_01 ?ms_01
                                ?sub01 ?circ101 ?circ201 ?volt01 ?plantid01 ?conclusion
                                ?rule_summary "1-10" 1 1)))
```

; *** Rule LE_11 – Circuit Breaker CLOSED

```
(defrule 1st_cb_closed_alarm
   (first)
   (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) (circuit2 ?circ201)
         (feeder ?feeder01) (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
         (ms_since1970 ?ms1970_01) (legend ?legend01) (plantid ?plantid01)
         (plantstatus ?plantstatus01))
   (test (cb_closed ?plantid01 ?plantstatus01))
=>
   (bind ?rule_summary (str-cat "Circuit Breaker CLOSED"))
   (bind ?conclusion (str-cat ?sub01 "  Circuit Breaker " ?plantid01 " CLOSED"))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                       (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
                       (plantid ?plantid01) (plantstatus ?plantstatus01) (pass_id 1) (rule_id "1-11")
                       (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                       ?ms1970_01 ?hundredths01 ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01
                       ?circ101 ?circ201 ?volt01 ?plantid01 ?conclusion ?rule_summary "1-11" 1 1)))
```

; * Rule LE_12– All tripped circuit breakers did not close**

```
(defrule 1st_tripped_cb_closure_incomplete
   (first)
   (test (call (fetch PLANT_OPS) switchout_partial))
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01)
          (voltage ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01)
          (ms_since1970 ?ms1970_01) (plantid ?plantid01) (plantstatus ?plantstatus01))
   (test (cb_open ?plantid01 ?plantstatus01))
   (test (call (fetch PLANT_OPS) earliest ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01 ?plantid01
                                          ?plantstatus01))
=>
   (bind ?rule_summary (str-cat "Tripped_cb_Closure_incomplete"))
   (bind ?conclusion (str-cat "All tripped circuit breakers did NOT close"))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
             (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (plantid ?plantid01) (voltage ?volt01)
             (pass_id 1) (rule_id "1-12") (summary ?rule_summary) (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event (fetch
         FINISH_DATE) (fetch FINISH_TIME) (fetch FINISH_MS1970) (fetch FINISH_MS) (fetch
         FINISH_DATE) (fetch FINISH_TIME) (fetch FINISH_MS1970) (fetch FINISH_MS)
         ?sub01 ?circ101 ?circ201 ?volt01 ?plantid01 ?conclusion ?rule_summary "1-12" 1 1)))
```

; * Rule LE_13– Autoswitching sequence time period**

```
(defrule 1st_auto_switching_period
   (first)
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01) (voltage
          ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01) (ms_since1970 ?ms1970_01)
          (plantid ?plantid01)
          (legend ?legend01&:(auto_prog ?legend01))
          (alarmstatus ?status01&:(alarm_on ?status01)))
(alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ202) (feeder ?feeder02) (voltage ?volt02)
(date ?date02) (time ?time02) (hundredths ?ms_02) (ms_since1970 ?ms1970_02) (plantid ?plantid02)

          (legend ?legend01&:(auto_complete ?legend02))
          (alarmstatus ?status02&:(alarm_off ?status02)))
=>
   (bind ?rule_summary (str-cat "Autoswitching Sequence"))
   (bind ?conclusion (str-cat "Autoswitching Sequence at " ?sub01 " " ?circ101 " took " (call (fetch
                     PLANT_OPS) switchout_period ?ms1970_01 ?ms_01 ?ms1970_02 ?ms_02)))
   (assert (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01) (sub
                     ?sub01) (circ1 ?circ101) (circ2 ?circ201) (plantid ?plantid01) (pass_id 1)
                     (rule_id "1-13") (voltage ?volt01) (summary ?rule_summary) (conclusion
                     ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date02 ?time02
                                ?ms1970_02 ?ms_02 ?date02 ?time02 ?ms1970_02 ?ms_02
                                ?sub02 ?circ102 ?circ202 ?volt02 ?plantid02 ?conclusion
                                ?rule_summary "1-13" 1 1)))
```

; * Rule LE_14– Circuit was not restored by end of incident**

```
(defrule 1st_partial_switchout_time
   (first)
   (test (call (fetch PLANT_OPS) switchout_partial))
   (alarm (substation ?sub01) (circuit1 ?circ101) (circuit2 ?circ201) (feeder ?feeder01) (voltage
          ?volt01) (date ?date01) (time ?time01) (hundredths ?ms_01) (ms_since1970 ?ms1970_01)
          (plantid ?plantid01) (plantstatus ?plantstatus01))
   (test (cb_open ?plantid01 ?plantstatus01))
   (test (call (fetch PLANT_OPS) earliest ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01 ?plantid01
                                          ?plantstatus01))
=>
   (bind ?rule_summary (str-cat "Restoration Incomplete"))
   (bind ?conclusion (str-cat (call (fetch IDENTIFIER) getSub) " / "(call (fetch IDENTIFIER)
                    getCirc1) " " (call (fetch IDENTIFIER) getCirc2) " circuit was not restored by
                    end of incident. Time elapsed = " (call (fetch PLANT_OPS) switchout_period
                    ?ms1970_01 ?hundredths01 (fetch FINISH_MS1970) (fetch FINISH_MS))))
   (assert (first_pass (date (fetch FINISH_DATE)) (time (fetch FINISH_TIME)) (hundredths (fetch
                     FINISH_MS)) (ms1970 (fetch FINISH_MS1970)) (sub ?sub01) (circ1
                     ?circ101) (circ2 ?circ201) (plantid ?plantid01) (plantstatus ?plantstatus01)
                     (pass_id 1) (rule_id "1-14") (voltage ?volt01)  (summary ?rule_summary)
                     (conclusion ?conclusion)))
   (call (fetch FIRST_PASS) add(new TelemetryProcessor.CoreClasses.Event
                              (fetch FINISH_DATE) (fetch FINISH_TIME)
                              (fetch FINISH_MS1970) (fetch FINISH_MS)
                              (fetch FINISH_DATE) (fetch FINISH_TIME)
                              (fetch FINISH_MS1970) (fetch FINISH_MS) ?sub01 ?circ101 ?circ201
                              ?volt01 ?plantid01 ?conclusion  ?rule_summary "1-14" 1 1)))
```

A.4 High-Level Event Rules

The JESS rules present in the High-level Events Rulebase are as follows:

; * Rule HE_1– Successful First Main Protection Operation**

```
(defrule successful_first_main_prot
   (second)
   (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
               (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
               (voltage ?volt01) (summary "1st Main Protection Operated ON"))
   (first_pass (date ?date02) (time ?time02) (hundredths ?hundredths02) (ms1970 ?ms1970_02)
               (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
               (voltage ?volt02) (summary "1st Main Protection Operated OFF"))
   (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) (circuit2 ?circ201)
          (voltage ?volt03) (date ?date03) (time ?time03) (hundredths ?ms_03)
          (ms_since1970 ?ms1970_03) (legend ?legend03&:(trip_alarm ?legend03)
          (alarmstatus ?status03&:(alarm_on ?status03))
   (test (eq ?sub_nov01 ?sub_nov02 ?sub_nov03)) (test (eq ?circ101 ?circ102 ?circ103))
   (second_pass (sub_novolt ?sub_nov03) (summary "CBs that opened") (conclusion ?conclusion04)
                (date ?date04) (time ?time04) (hundredths ?ms_04) (ms1970 ?ms1970_04)))
=>
   (bind ?rule_summary (str-cat "Successful 1st Main Protection Operation"))
   (bind ?conclusion (str-cat "1st Main Protection operated successfully at " ?sub01 " -> " ?circ101))
   (assert (second_pass (date ?date01) (time ?time01) (hundredths ?hundredths01)
          (ms1970 ?ms1970_01) (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 2)
          (rule_id "2-1") (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)
          (event_object (new TelemetryProcessor.CoreClasses.Event ?date01 ?time01 ?ms1970_01
          ?hundredths01 ?date01 ?time01 ?ms1970_01 ?hundredths01 ?sub01?circ101 ?circ201
          ?volt01 x ?conclusion ?rule_summary "2-1" 2 1))))
   (call (fetch SECOND_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                ?ms1970_01 ?hundredths01 ?date01 ?time01 ?ms1970_01
                                ?hundredths01 ?sub01 ?circ101 ?circ201 ?volt01 x
                                ?conclusion ?rule_summary "2-1" 2 1)))
```

; *** Rule HE_2– Successful Second Main Protection Operation

```
(defrule successful_second_main_prot
    (second)
    (first_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
                (voltage ?volt01) (summary "2nd Main Protection Operated ON"))
    (first_pass (date ?date02) (time ?time02) (hundredths ?ms_02) (ms1970 ?ms1970_02)
                (sub ?sub01) (sub_novolt ?sub_nov01) (circ1 ?circ101) (circ2 ?circ201)
                (voltage ?volt02) (summary "2nd Main Protection Operated OFF"))
    (alarm (substation ?sub01) (sub_novolt ?sub_nov01) (circuit1 ?circ101) (circuit2 ?circ201)
           (voltage ?volt03) (date ?date03) (time ?time03) (hundredths ?ms_03)
           (ms_since1970 ?ms1970_03) (legend ?legend03&:(trip_alarm ?legend03)
           (alarmstatus ?status03&:(alarm_on ?status03))
    (test (eq ?sub_nov01 ?sub_nov02 ?sub_nov03)) (test (eq ?circ101 ?circ102 ?circ103))
    (second_pass (sub_novolt ?sub_nov03) (summary "CBs that opened") (conclusion ?conclusion04)
                 (date ?date04) (time ?time04) (hundredths ?ms_04) (ms1970 ?ms1970_04)))
=>
    (bind ?rule_summary (str-cat "Successful 2nd Main Protection Operation"))
    (bind ?conclusion (str-cat "2nd Main Protection operated successfully at " ?sub01 " -> " ?circ101))
    (assert (second_pass (date ?date01) (time ?time01) (hundredths ?hundredths01)
           (ms1970 ?ms1970_01) (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 2)
           (rule_id "2-2") (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)
           (event_object (new TelemetryProcessor.CoreClasses.Event ?date01 ?time01 ?ms1970_01
                         ?hundredths01 ?date01 ?time01 ?ms1970_01 ?hundredths01 ?sub01?circ101
                         ?circ201 ?volt01 x ?conclusion ?rule_summary "2-2" 2 1))))
    (call (fetch SECOND_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                     ?ms1970_01 ?hundredths01 ?date01 ?time01 ?ms1970_01
                                     ?hundredths01 ?sub01 ?circ101 ?circ201 ?volt01 x
                                     ?conclusion ?rule_summary "2-2" 2 1)))
```

; *** Rule HE_3– First Main Protection Failed to Operate

```
(defrule 1st_main_failed_to_operate
    (second)

    (first_pass (date ?date01) (time ?time01) (hundredths ?hundredths01) (ms1970 ?ms1970_01)
        (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt01)
        (summary ?rule_summary01&:(eq ?rule_summary01 "2nd Main Protection Operated ON")))
    (not (first_pass (date ?date02) (time ?time02) (hundredths ?ms_02) (ms1970 ?ms1970_02)
         (sub ?sub02&:(eq ?sub02 ?sub01)) (circ1 ?circ102&:(eq ?circ101 ?circ102))
         (circ2 ?circ202&:(eq ?circ201 ?circ202)) (voltage ?volt02)
         (summary ?rule_summary02&:(eq ?rule_summary02 "1st Main Protection Operated ON"))))
=>
    (bind ?rule_summary (str-cat "Failed 1st Main Protection Operation"))
    (bind ?conclusion (str-cat "1st Main Protection at " ?sub01 " -> " ?circ101 " failed to operate"))
    (assert (second_pass (date (fetch FINISH_DATE)) (time (fetch FINISH_TIME)) (hundredths
           (fetch FINISH_MS)) (ms1970 (fetch FINISH_MS1970)) (sub ?sub01) (circ1 ?circ101)
           (circ2 ?circ201) (pass_id 2) (rule_id "2-3") (voltage ?volt01) (summary ?rule_summary)
           (conclusion ?conclusion)))
    (call (fetch SECOND_PASS) add(new TelemetryProcessor.CoreClasses.Event
                                    (fetch FINISH_DATE) (fetch FINISH_TIME)
                                     (fetch FINISH_MS1970) (fetch FINISH_MS)
                                    (fetch FINISH_DATE) (fetch FINISH_TIME)
                                    (fetch FINISH_MS1970) (fetch FINISH_MS)
                                    ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion ?rule_summary
                                    "2-3" 2 1)))
```

; *** Rule HE_4– Second Main Protection Failed to Operate

```
(defrule 2nd_main_failed_to_operate
    (second)
    (first_pass (date ?date01) (time ?time01) (hundredths ?hundredths01) (ms1970 ?ms1970_01)
          (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt01)
          (summary ?rule_summary01&:(eq ?rule_summary01 "1st Main Protection Operated ON")))
    (not (first_pass (date ?date02) (time ?time02) (hundredths ?ms_02) (ms1970 ?ms1970_02)
          (sub ?sub02&:(eq ?sub02 ?sub01)) (circ1 ?circ102&:(eq ?circ101 ?circ102))
          (circ2 ?circ202&:(eq ?circ201 ?circ202)) (voltage ?volt02)
          (summary ?rule_summary02&:(eq ?rule_summary02 "2nd Main Protection Operated
          ON"))))
=>
    (bind ?rule_summary (str-cat "Failed 2nd Main Protection Operation"))
    (bind ?conclusion (str-cat "2nd Main Protection at " ?sub01 " -> " ?circ101 " failed to operate"))
    (assert (second_pass (date (fetch FINISH_DATE)) (time (fetch FINISH_TIME)) (hundredths
          (fetch FINISH_MS)) (ms1970 (fetch FINISH_MS1970)) (sub ?sub01) (circ1 ?circ101)
          (circ2 ?circ201) (pass_id 2) (rule_id "2-4") (voltage ?volt01) (summary ?rule_summary)
          (conclusion ?conclusion)))
    (call (fetch SECOND_PASS) add(new TelemetryProcessor.CoreClasses.Event
                                  (fetch FINISH_DATE) (fetch FINISH_TIME)
                                  (fetch FINISH_MS1970) (fetch FINISH_MS)
                                  (fetch FINISH_DATE) (fetch FINISH_TIME)
                                  (fetch FINISH_MS1970) (fetch FINISH_MS) ?sub01 ?circ101
                                  ?circ201 ?volt01 x ?conclusion ?rule_summary "2-4" 2 1)))
```

; *** Rule HE_5– First and Second Main Protection Operating Successfully

```
(defrule 1st_2nd_main_successful
    (second)
    (second_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
                 (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt01)
                 (summary ?rule_summary01&:(eq ?rule_summary01 "Successful 1st Main
                 Protection Operation")) (event_object ?event01))
    (second_pass (date ?date02) (time ?time02) (hundredths ?ms_02) (ms1970 ?ms1970_02)
                 (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt02) (summary
                 ?rule_summary02&:(eq ?rule_summary02 "Successful 2nd Main Protection
                 Operation")) (event_object ?event02))
    (not (second_pass (sub ?sub03&:(eq ?sub01 ?sub03)) (circ1 ?circ103&:(eq ?circ101 ?circ103))
                      (summary ?summary&:(eq ?summary "Successful 1st and 2nd Main
                      Protection Operation"))))
=>
    (bind ?rule_summary (str-cat "Successful 1st and 2nd Main Protection Operation"))
    (bind ?conclusion (str-cat "1st and 2nd Main Protection operated successfully at " ?sub01 " -> "
                      ?circ101))
    (assert (second_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
          (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (pass_id 2) (rule_id "2-5")
          (voltage ?volt01) (summary ?rule_summary) (conclusion ?conclusion)
          (event_object (new TelemetryProcessor.CoreClasses.Event ?date01 ?time01 ?ms1970_01
                        ?ms_01 ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01 ?circ101 ?circ201
                        ?volt01 x ?conclusion ?rule_summary "2-5" 2 1))))
    (call (fetch SECOND_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                  ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01
                                  ?circ101 ?circ201 ?volt01 x ?conclusion ?rule_summary "2-5" 2 1))
    (eventmanager (fetch SECOND_PASS) ?event01)
    (eventmanager (fetch SECOND_PASS) ?event02))
```

; *** Rule HE_6– First and Second Intertrips Operated at Substation

```
(defrule 1st_2nd_intertrip_at_sub
   (second)
   (first pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
            (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt01)
            (summary ?rule_summary01&:(eq ?rule_summary01 "1st Intertrip Received ON")))
   (first pass (date ?date02) (time ?time02) (hundredths ?ms_02) (ms1970 ?ms1970_02)
            (sub ?sub01) (circ1 ?circ102) (circ2 ?circ202) (voltage ?volt02)
            (summary ?rule_summary02&:(eq ?rule_summary02 "2nd Intertrip Received ON")))
=>
   (bind ?rule_summary (str-cat "1st and 2nd Intertrip received at substation"))
   (bind ?conclusion (str-cat "1st and 2nd Intertrips received at " ?sub01 " from " ?circ101))
   (assert (second_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
          (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt01) (summary
          ?rule_summary) (conclusion ?conclusion) (pass_id 2) (rule_id "2-9")
          (event_object(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01 ?ms1970_01
                    ?ms_01?date01 ?time01 ?ms1970_01 ?ms_01 ?sub01 ?circ101 ?circ201
                    ?volt01 x ?conclusion ?rule_summary "2-6" 2 1))))
   (call (fetch SECOND_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                ?ms1970_01 ?ms_01 ?date01 ?time01 ?ms1970_01 ?ms_01
                                ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion
                                ?rule_summary "2-6" 2 1)))
```

; *** Rule HE_7– First and Second Intertrips Operated at Both Ends

```
(defrule 1st_2nd_intertrip_at_both_ends
   (second)
   (second_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
            (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt01)
            (summary ?rule_summary01&:(eq ?rule_summary01 "1st and 2nd Intertrip
            received at substation")) (event_object ?event01))
   (second_pass (date ?date02) (time ?time02) (hundredths ?ms_02) (ms1970 ?ms1970_02)
            (sub ?sub02) (circ1 ?circ102) (circ2 ?circ202) (voltage ?volt02)
            (summary ?rule_summary02&:(eq ?rule_summary02 "1st and 2nd Intertrip
            received at substation")) (event_object ?event02))
   (test (eq (sub-string 1 3 ?sub02) (sub-string 1 3 ?circ101)))
=>
   (bind ?rule_summary (str-cat "1st and 2nd Intertrip received at both ends"))
   (bind ?conclusion (str-cat "1st and 2nd Intertrips received at both ends"))
   (assert (second_pass (date ?date01) (time ?time01) (hundredths ?ms_01) (ms1970 ?ms1970_01)
          (sub ?sub01) (circ1 ?circ101) (circ2 ?circ201) (voltage ?volt01)
          (summary ?rule_summary) (conclusion ?conclusion) (pass_id 2) (rule_id "2-7")))
   (call (fetch SECOND_PASS) add(new TelemetryProcessor.CoreClasses.Event ?date01 ?time01
                                ?ms1970_01 ?ms_01 ?date02 ?time02 ?ms1970_02 ?ms_02
                                ?sub01 ?circ101 ?circ201 ?volt01 x ?conclusion
                                ?rule_summary "2-7" 2 1))
   (eventmanager (fetch SECOND_PASS) ?event01)
   (eventmanager (fetch SECOND_PASS) ?event02))
```

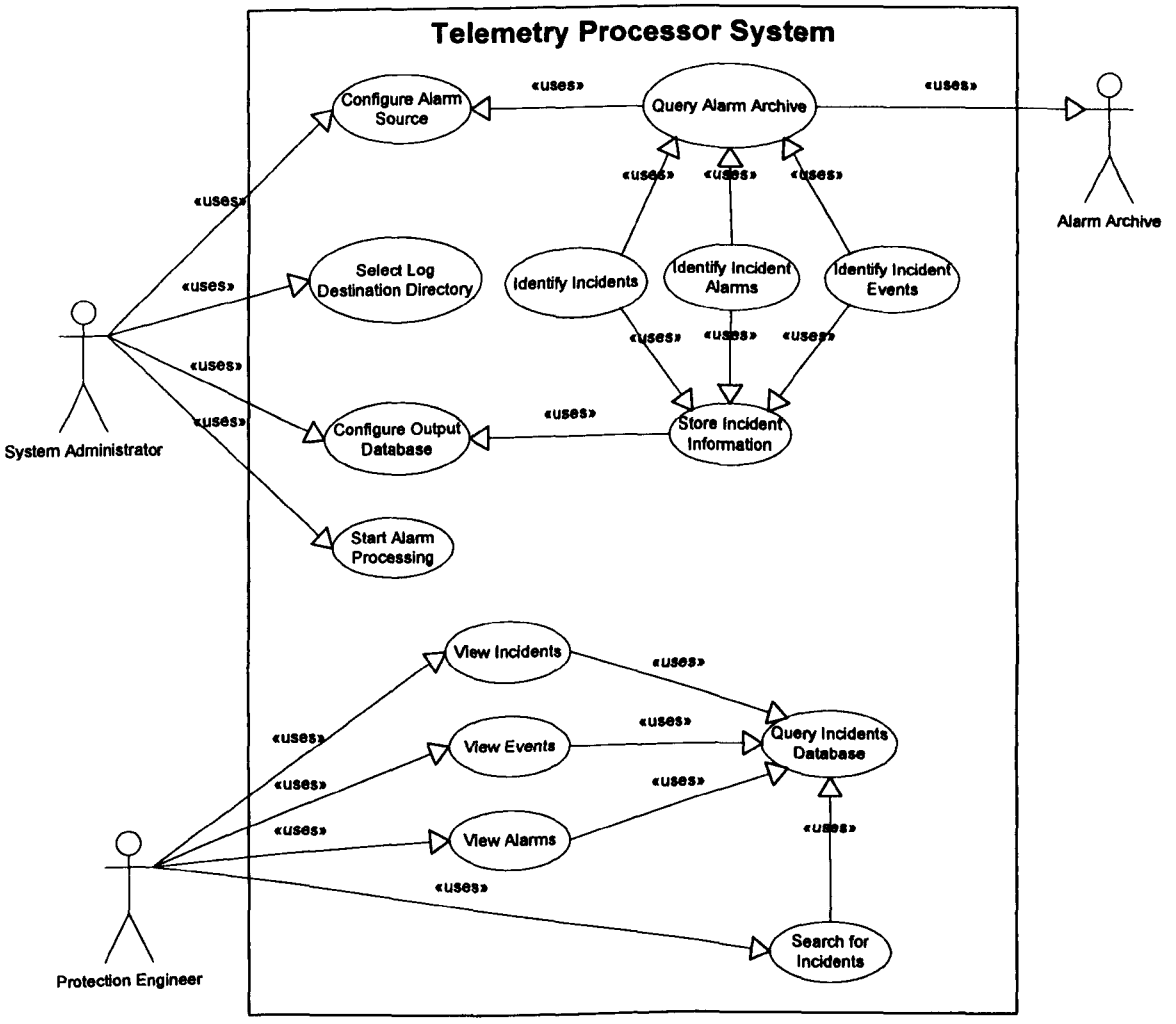
Appendix B: PEDA Use Cases

This appendix contains the use cases created during the Requirements and Knowledge Capture stage of PEDA design described in section 7.3 of the thesis.

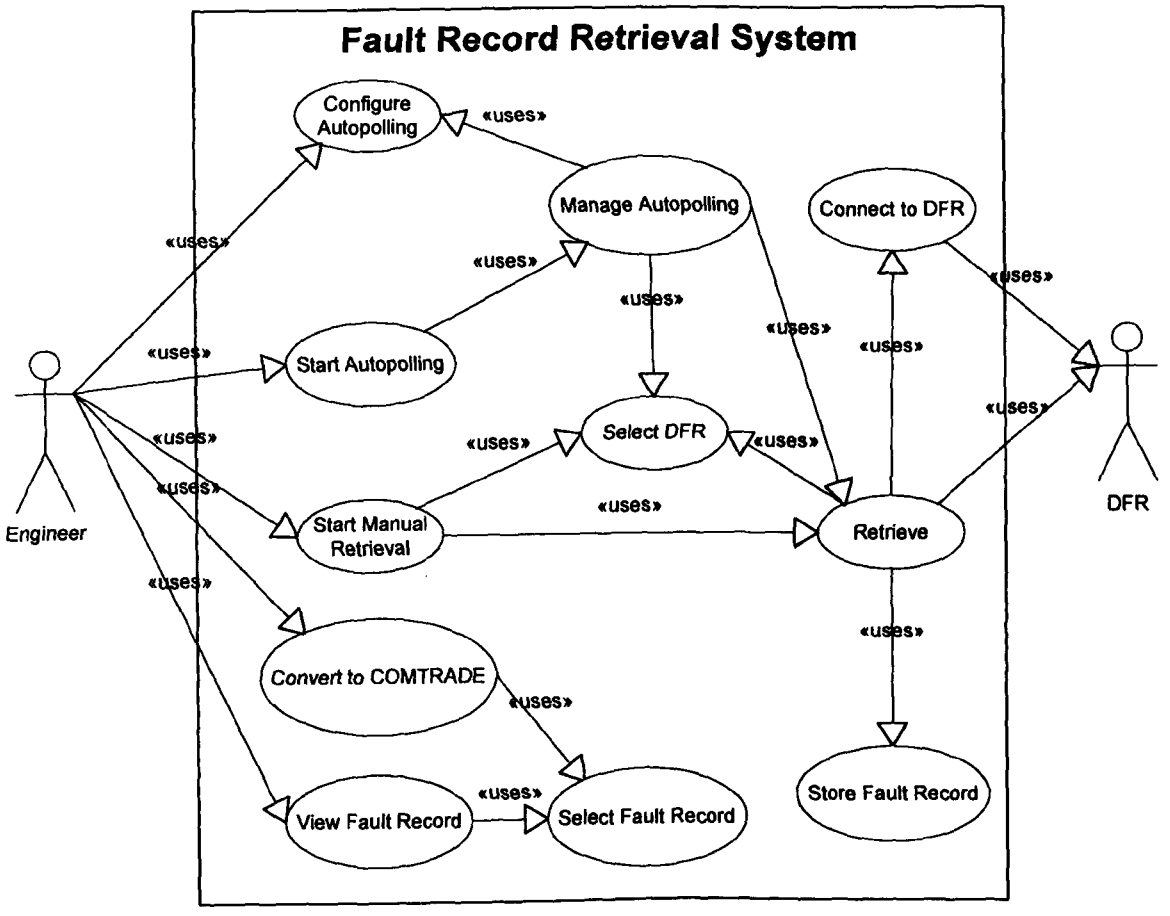
The use cases within this appendix are listed below:

- Telemetry Processor System
- Fault Record Retrieval System
- Fault Record Interpretation System
- Protection Validation Toolkit

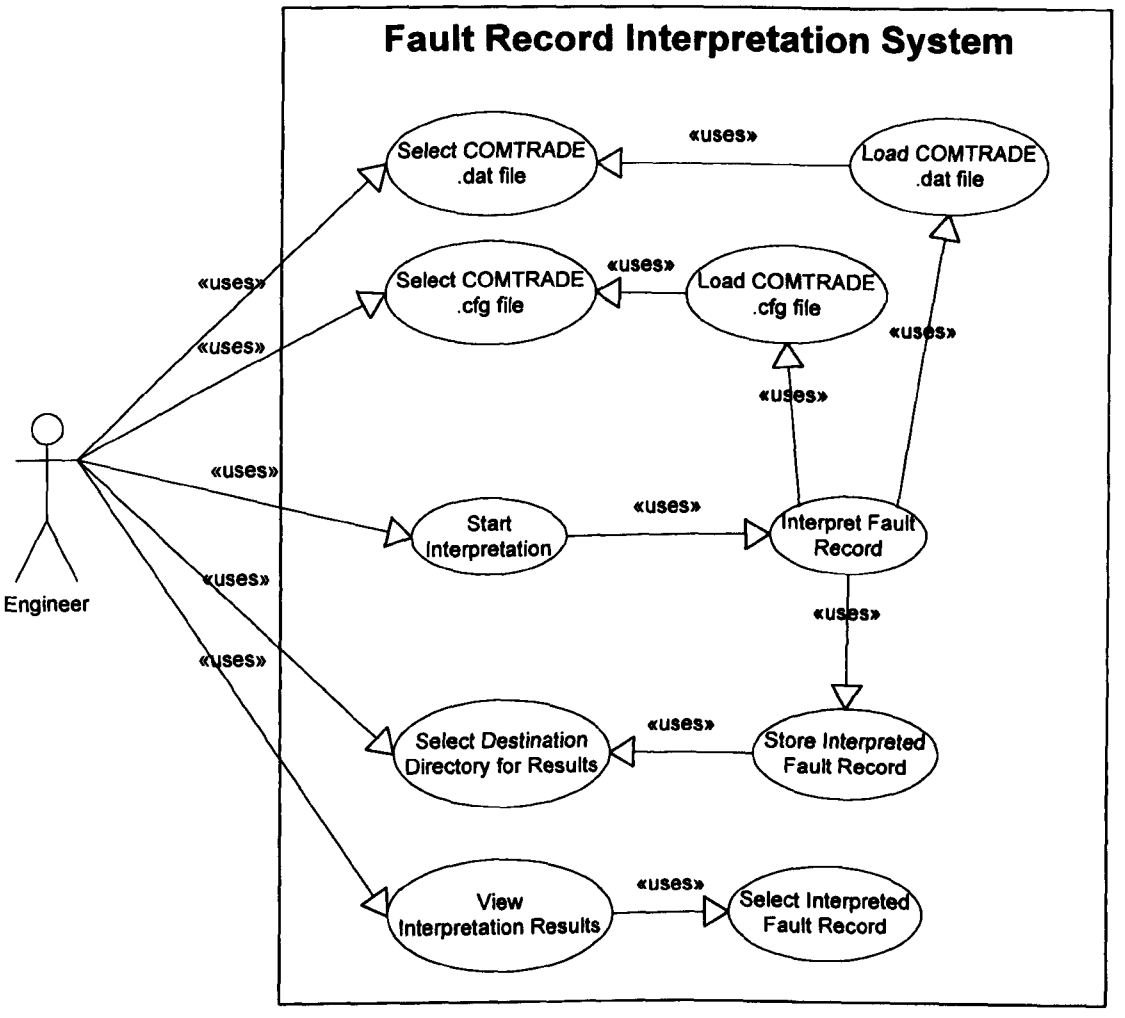
B.1 Use Case - Telemetry Processor System



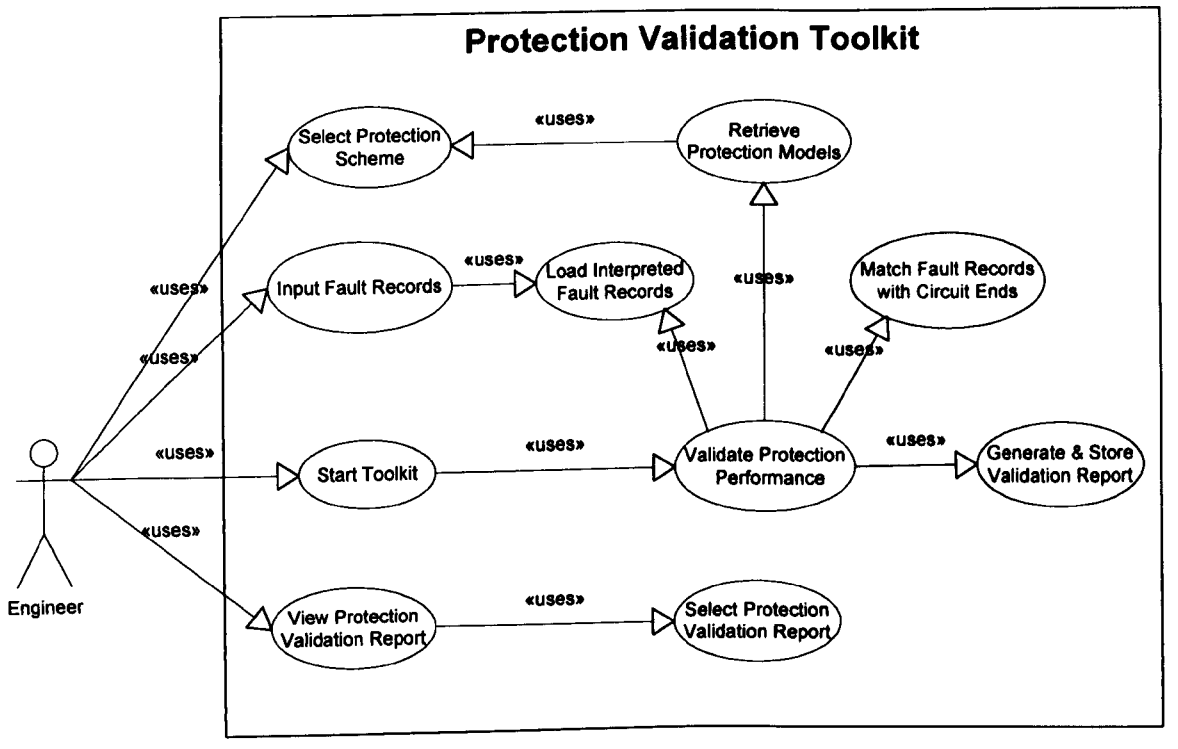
B.2 Use Case – Fault Record Retrieval System



B.3 Use Case – Fault Record Interpretation System



B.4 PEDA Use Case – Protection Validation Toolkit



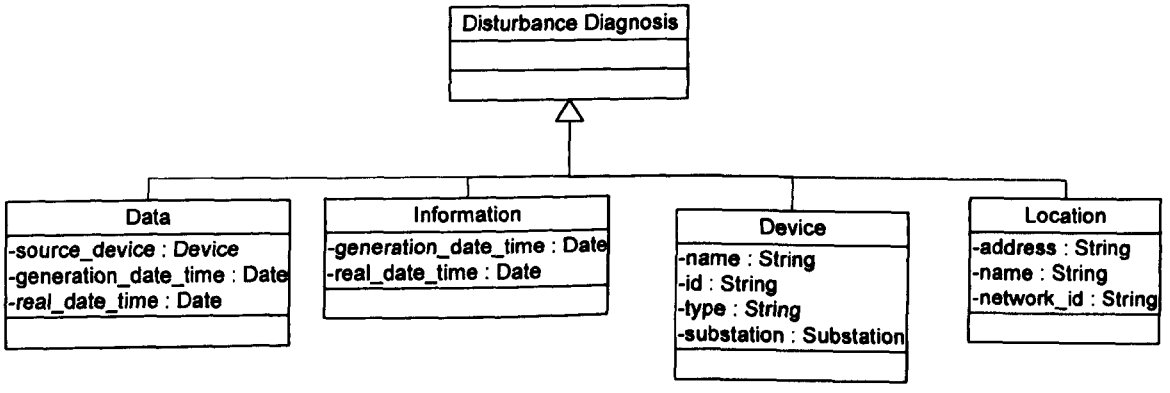
Appendix C: PEDA Ontology

This appendix contains the class hierarchy diagrams for the Disturbance Diagnosis ontology created during the 'Ontology Design' stage of PEDA specification described in section 7.5 of the thesis.

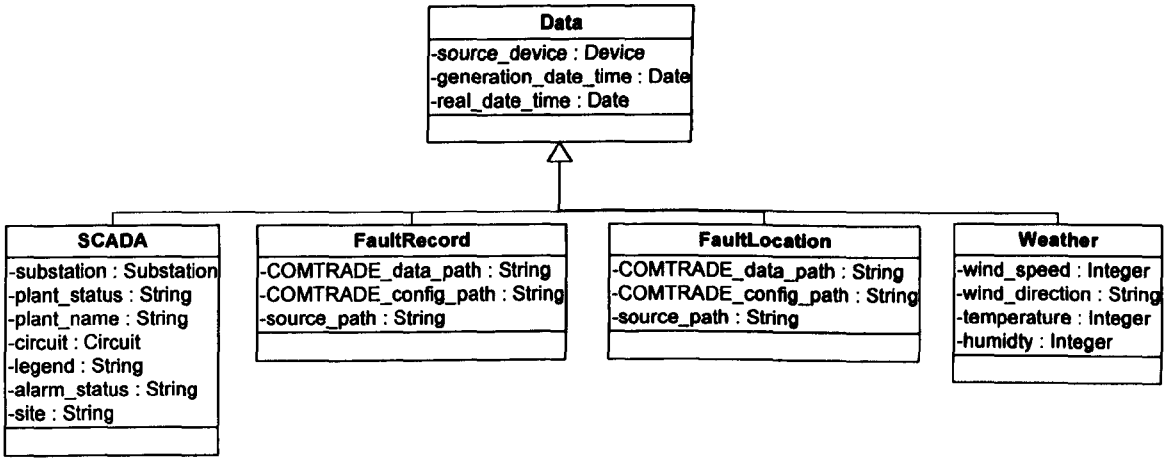
The class hierarchies within this appendix are listed below:

- Disturbance Diagnosis Subclasses
- Data Subclasses
- Information Subclasses
- Location Subclasses
- Device Subclasses

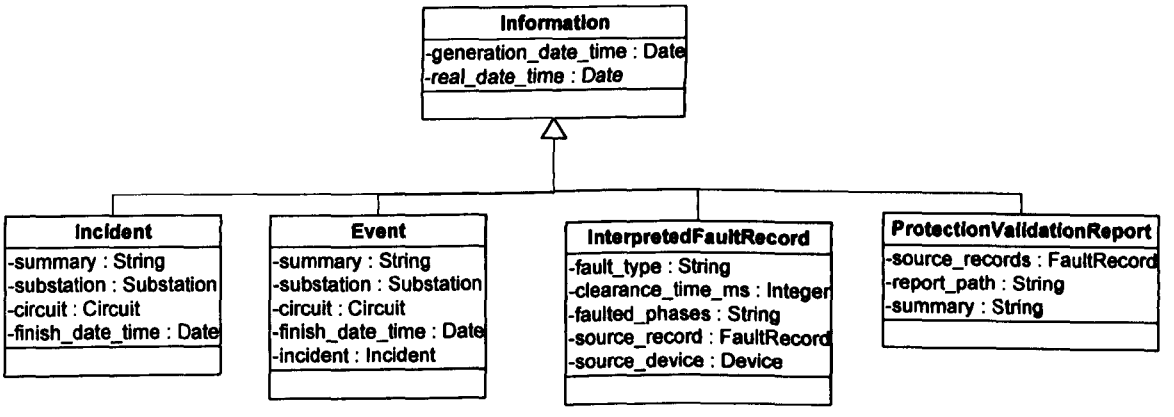
C.1 Disturbance Diagnosis Subclasses



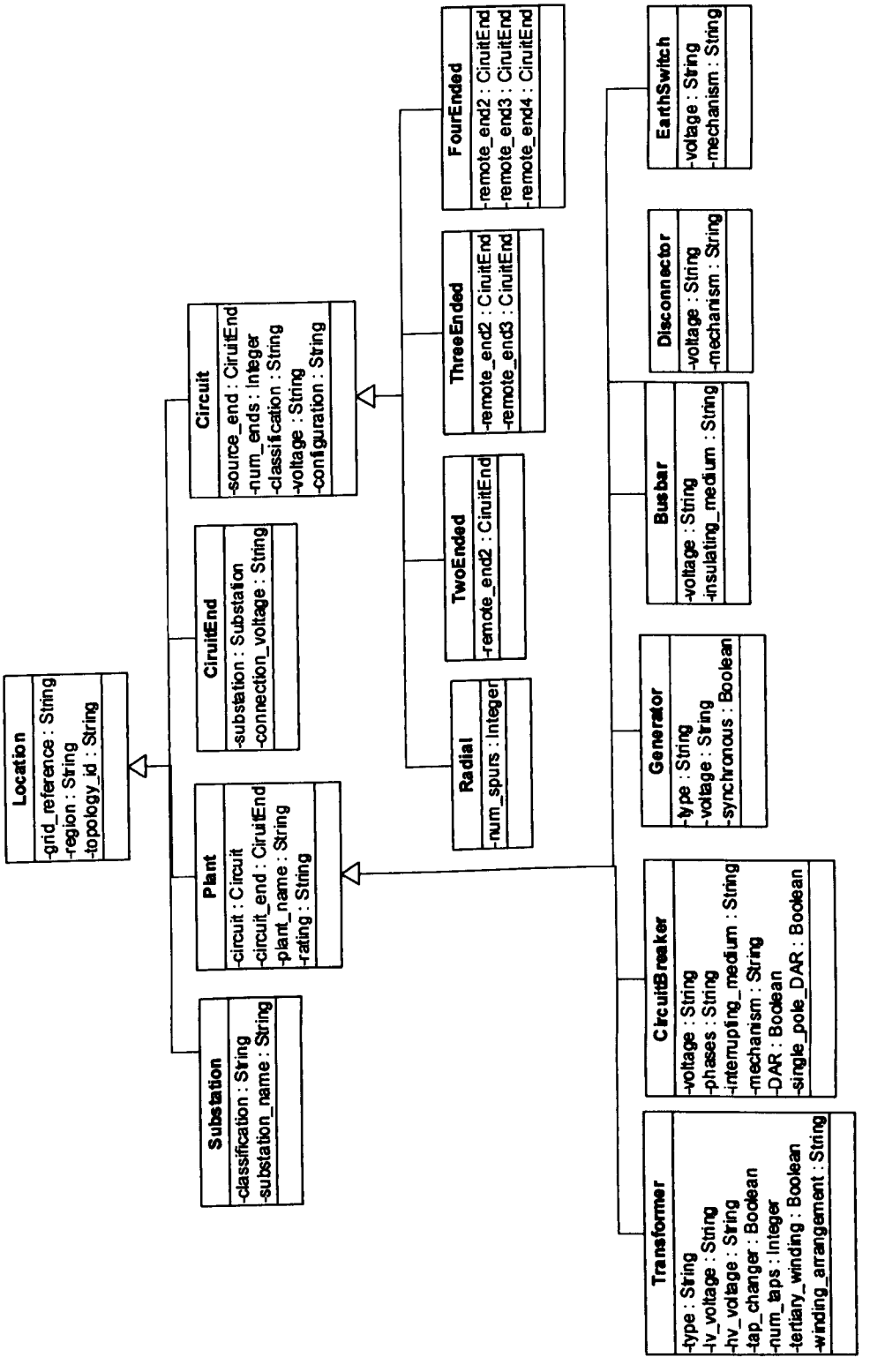
C.2 Data Subclasses



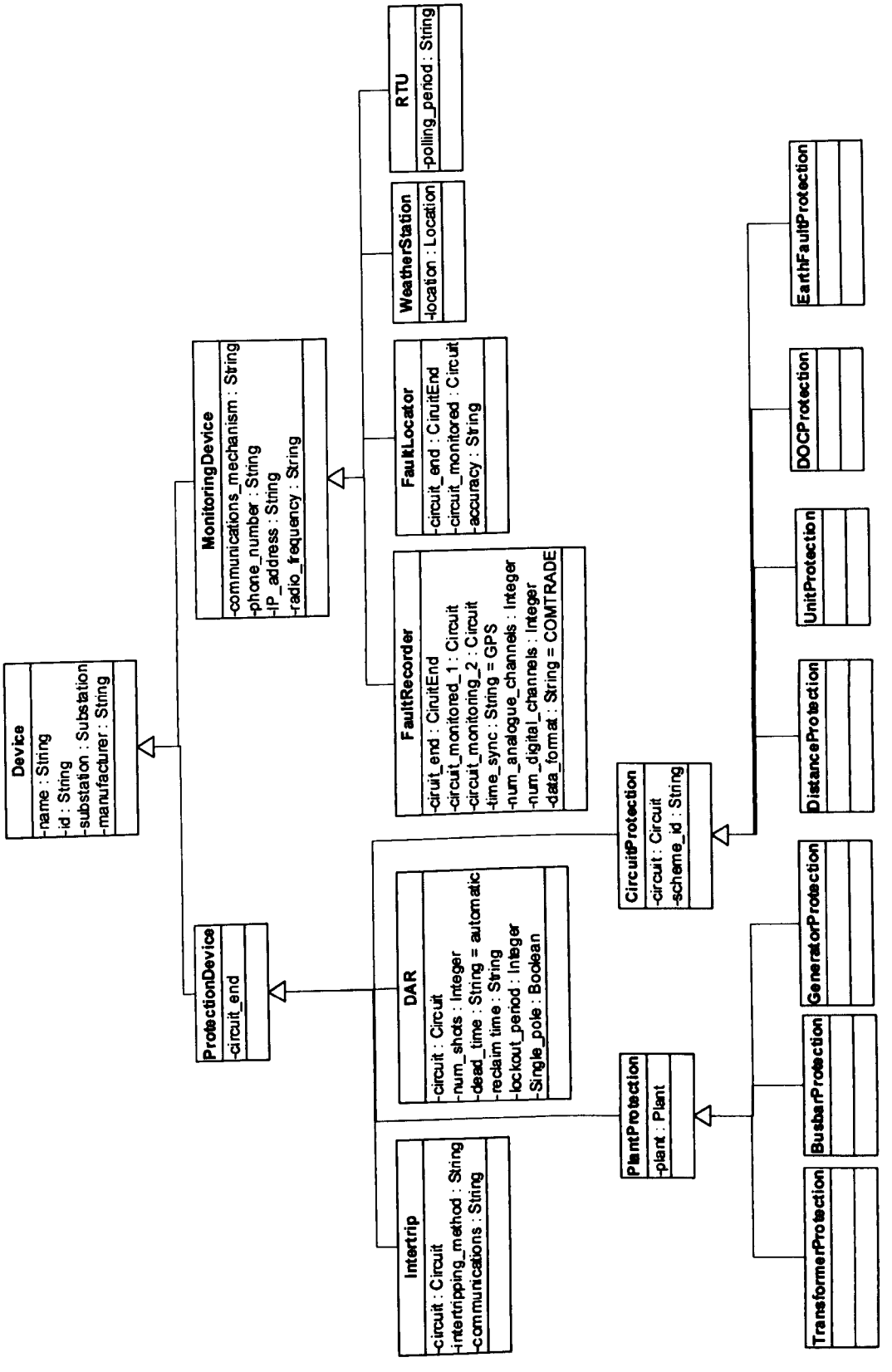
C.3 Information Subclasses



C.4 Location Subclasses



C.5 Device Subclasses



Appendix D: Legacy System Assessment Templates

This appendix contains the templates used to record the assessment of legacy system capabilities during stage four of the PEDA specification process described in section 7.6 of the thesis.

D.1 Telemetry Processor Software Assessment

Software Resource Assessment:	Telemetry Processor				
Resource Description:	Online expert system for identifying transmission network incidents and events.				
Source Code Available?	*Y*	**API?**	*Y*	**Language:**	*Java*
Control Requirements	• User intervention to upload a configuration file. • Telemetry Processor is started via a user interface. • No further user intervention is required once started.				

Functional Capabilities:

Function	Description	Possible Task Mapping
Query alarm archive	Telemetry Processor queries the alarm archive at fixed intervals and retrieves any new alarms.	Retrieve SCADA
Identify incidents	Interprets alarms looking for an incident start and conclusion.	Interpret SCADA for Incidents
Identify incident alarms	Searches for alarms occurring on the incident circuit between incident start and conclusion.	Group Incident Related SCADA
Identify incident events	Interprets incident alarms for pertinent protection events.	Interpret Incident SCADA for Events
Search for incidents	Allows user to search for historical incidents via a web-based user interface.	*Not required in task hierarchy since PEDA will operate online and will not require historical searches.*

Data Input Requirements:

Data	Mechanism	Ontology Mapping
Alarms	Automatic alarm archive query	SCADA

Data Provision Capabilities:

Data	Mechanism	Ontology Mapping
Incident Report	Database query	Incident
Incident Event	Database query	Event
Alarm	Database query	SCADA

D.2 Fault Record Retrieval Software Assessment

Software Resource Assessment:	Proprietary Fault Record Retrieval Software
Resource Description:	Software developed by DFR manufacturers to manage remote DFRs. Users can initiate manual retrieval of fault records from remote devices. Time scheduled non-prioritised autopolling is also possible.

Source Code Available?	N	**API?**	N	**Language:**	Unknown

Control Requirements	• User intervention to configure autopolling. • User intervention to identify DFR to initiate retrieval from. • User intervention to initiate retrieval.

Functional Capabilities:

Function	Description	Possible Task Mapping
Configurable Autopolling	User can select which DFRs to initiate retrieval from and when. Software then manages the autopolling.	Create Autopolling Schedule
DFR selection	Software or user can select the next DFR to retrieve fault records from and obtain the configuration details necessary to begin retrieval.	Select Next Retrieval
Request retrieval of new fault records from a DFR	Software or user can begin retrieval once connected to the DFR.	Retrieve
Monitor device communications	Software will log any device communications problems.	Monitor Device Availability
COMTRADE conversion	User can request fault record conversion to COMTRADE.	*Not required in task hierarchy since all fault records will already be available in COMTRADE format.*
Fault record viewing and analysis.	User can view retrieved records and perform routine analysis using the tools provided by the software.	*Not required in task hierarchy since fault records can still be viewed using proprietary software.*

Data Input Requirements:

Data	Mechanism	Ontology Mapping
DFR	User selects from list of available DFRs	FaultRecorder

Data Provision Capabilities:

Data	Mechanism	Ontology Mapping
Fault records	Stored as a file in a directory structure. File can be accessed via software or via Windows functions, e.g. My Computer	FaultRecord

D.3 Fault Record Interpretation Software Assessment

Software Resource Assessment:	Fault Record Interpretation Software
Resource Description:	Software developed primarily to prepare faults records for protection validation. Requires fault records in COMTRADE format. Identifies protection operating times, faulted phases and fault clearance time.

Source Code Available?	Y	**API?**	Y	**Language:**	Java and C

Control Requirements	• User intervention to start software. • User intervention to upload COMTRADE .dat and .cfg files. • User intervention to start interpretation.

Functional Capabilities:

Function	Description	Possible Task Mapping
Upload a COMTRADE fault record	User selects the fault record .cfg and .dat files from the directory structure.	*No mapping since PEDA will need to schedule and automate the selection of fault records for upload*
Select destination directory for results	User selects the destination directory for the interpretation results.	*Not required in task hierarchy since PEDA will be set to use a default directory.*
Interpret fault record	Software loads in the user selected .dat and .cfg files and interprets the fault record.	Interpret Fault Record
View interpretation results	User can either view the interpretation results on the user interface or open the results file.	*Not required in task hierarchy since PEDA is only responsible for disturbance diagnosis automation.*

Data Input Requirements:

Data	Mechanism	Ontology Mapping
COMTRADE fault record files	Manual selection and input of both the .dat and .cfg COMTRADE files associated with the original record	FaultRecord

Data Provision Capabilities:

Data	Mechanism	Ontology Mapping
Interpreted fault record	Output by the software as a text file and stored in a directory structure.	InterpretedFaultRecord

D.4 PV Toolkit Software Assessment

Software Resource Assessment:	Protection Validation Toolkit
Resource Description:	An intelligent systems using MBD to validate performance of transmission protection schemes. Requires models of the protection scheme being validated and interpreted fault records from each circuit end.

Source Code Available?	Y	**API?**	Y	**Language:**	Java, XML and C

Control Requirements	• User intervention to start software. • User selects protection scheme to be validated. • User selects input fault records from each circuit end. • User intervention to start protection validation

Functional Capabilities:

Function	Description	Possible Task Mapping
Select protection scheme	User selects the protection scheme to be validated from a library. The software then retrieves and loads the models.	Select Protection Scheme
Upload COMTRADE fault records	User selects the .cfg and .dat files from the directory structure for each fault record.	*No mapping since PEDA will need to schedule and automate the selection of fault records for upload*
Upload interpreted fault records	User selects the interpreted fault record files for each circuit end from the directory structure.	*No mapping since PEDA will need to schedule and automate the selection of interpreted fault records for upload*
Validate protection performance	User initiates protection validation which reads the uploaded information and runs the protection models.	Run Protection Models
View protection validation report	User can either view protection validation report on the user interface or open the report file.	*Not required in task hierarchy since PEDA is only responsible for disturbance diagnosis automation.*

Data Input Requirements:

Data	Mechanism	Ontology Mapping
COMTRADE fault record files	Manual selection and input of both the .dat and .cfg COMTRADE fault record files for each circuit end.	FaultRecord
Interpreted fault record files	Manual selection of the interpreted fault records from each circuit end.	InterpretedFaultRecord

Data Provision Capabilities:

Data	Mechanism	Ontology Mapping
Protection validation report	Output by the software as a text file and stored in a directory structure.	ProtectionValidationReport

Appendix E: Agent Modelling Templates

This appendix contains the agent modelling templates generated during stage eight of the PEDA specification process described in section 7.10 of the thesis.

E.1 Incident and Event Identification (IEI) Agent

AGENT NAME:	Incident and Event Identification (IEI)	
AGENT ROLE:	Automated interpretation of transmission SCADA alarms and the provision of SCADA data, and incident and event information to agents.	
Functional Tasks:	**Task Realisation Method**	
Retrieve SCADA	Existing system: Telemetry Processor	
Interpret SCADA for Incidents	Existing system: Telemetry Processor	
Group Incident related SCADA	Existing system: Telemetry Processor	
Interpret Incident SCADA for Events	Existing system: Telemetry Processor	
Interaction Tasks:	**Interaction Type**	**Exchanged Resource**
Provide SCADA	subscribe, query-ref	SCADA
Provide Incidents	subscribe, query-ref	Incident
Provide Events	subscribe, query-ref	Event

E.2 Fault Record Retrieval (FRR) Agent

AGENT NAME:	Fault Record Retrieval (FRR)	
AGENT ROLE:	Automated and prioritised retrieval of fault records and the provision of fault records to agents.	
Functional Tasks:	**Task Realisation Method**	
Select Next Retrieval	Algorithmic Code	
Create Autopolling Schedule	Algorithmic Code	
Reschedule Retrieval	Rules, Algorithmic Code	
Retrieve	Algorithmic Code	
Monitor device availability	Algorithmic Code	
Interaction Tasks:	**Interaction Type**	**Exchanged Resource**
Provide Fault Records	subscribe, query-ref, request	FaultRecord
Obtain Identified Incidents	subscribe	Incident

E.3 Fault Record Interpretation (FRI) Agent

AGENT NAME:	Fault Record Interpretation (FRI)	
AGENT ROLE:	Automated and prioritised interpretation of fault records and the provision of interpreted fault records to agents.	
Functional Tasks:	**Task Realisation Method**	
Select Next Fault Record	Algorithmic Code	
Schedule Intepretation	Rules, Algorithmic Code	
Interpret Fault Record	Existing system: Fault Record Interpretation Software	
Interaction Tasks:	**Interaction Type**	**Exchanged Resource**
Provide Interpreted Fault Records	subscribe, query-ref, request	InterpretedFaultRecord
Obtain Identified Incidents	subscribe	Incident
Obtain Retrieved Fault Records	subscribe, query-ref, request	FaultRecord

E.4 Protection Validation & Diagnosis (PVD) Agent

AGENT NAME:	Protection Validation and Diagnosis (PVD)
AGENT ROLE:	Validation of protection performance and diagnosis of protection failures and the provision of protection validation reports to agents.

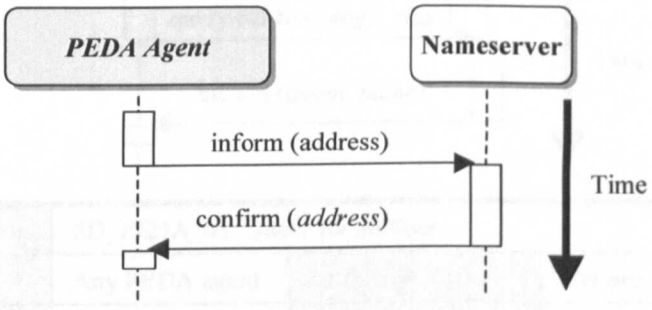
Functional Tasks:	**Task Realisation Method**
Develop Validation Schedule	Rules, Algorithmic Code
Reschedule Validation	Rules, Algorithmic Code
Select Next Validation	Algorithmic Code
Select Protection Scheme	Existing system: Protection Validation Toolkit
Run Protection Models	Existing system: Protection Validation Toolkit

Interaction Tasks:	**Interaction Type**	**Exchanged Resource**
Obtain Identified Incidents	subscribe	Incident
Obtain Interpreted Fault Records	query-ref, request	InterpretedFaultRecord
Provide Protection Validation Report	subscribe, query-ref	ProtectionValidationReport

Appendix F: PEDA Sequence Diagrams

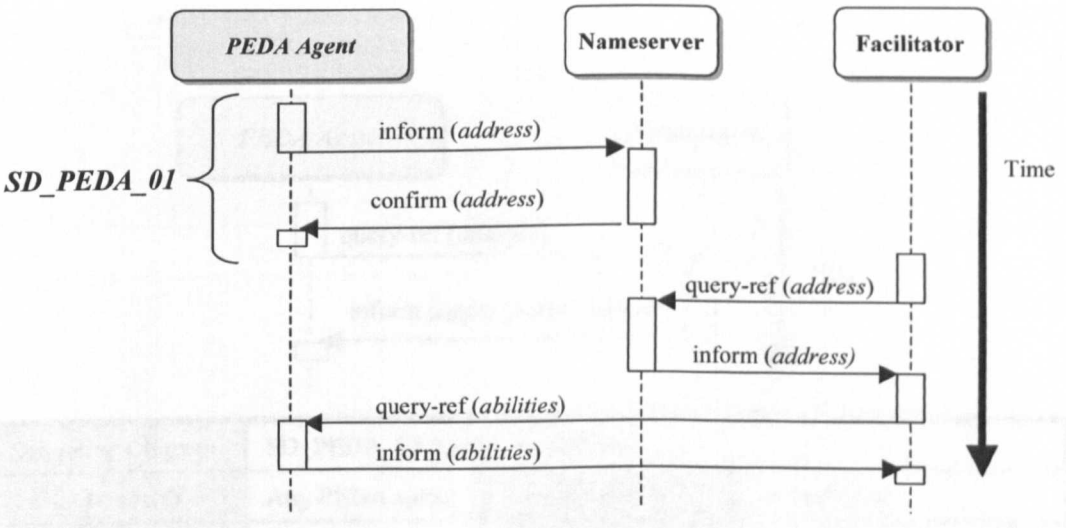
This appendix contains the sequence diagrams created during the agent interactions stage of the PEDA specification process described in section 7.12 of the thesis.

F.1 SD_PEDA_01 = Nameserver registration



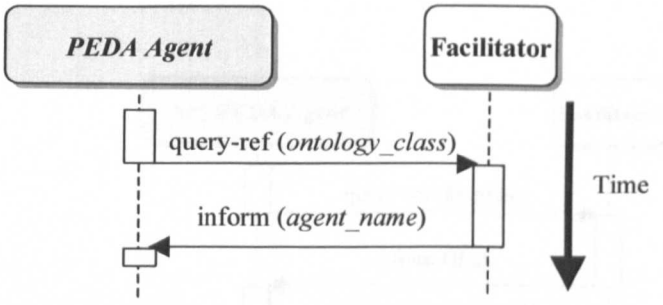
Sequence Diagram	SD_PEDA_01: Nameserver registration		
Task Owner(s)	Any PEDA agent	**Initiating Task**	*Register location*
Task Owner(s)	Nameserver	**Responding Task**	Acknowledge Registration
Other participants	None	**Responding Task**	N/A

F.2 SD_PEDA_02 = Provide Abilities



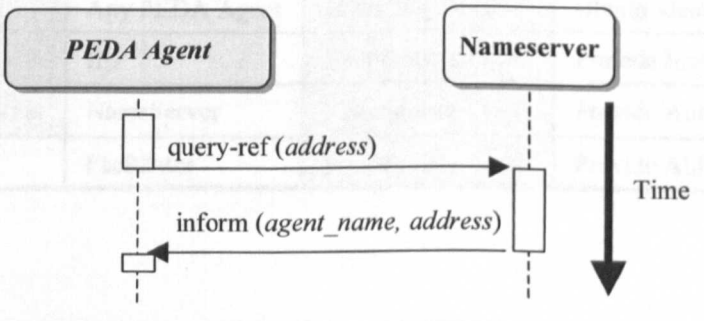
Sequence Diagram	SD_PEDA_02: Provide Abilities		
Task Owner(s)	Facilitator	**Initiating Task**	Request Abilities
Task Owner(s)	Any PEDA agent	**Responding Task**	*Provide Abilities*
Other participants	None	**Responding Task**	N/A

F.3 SD_PEDA_03 = Query for abilities



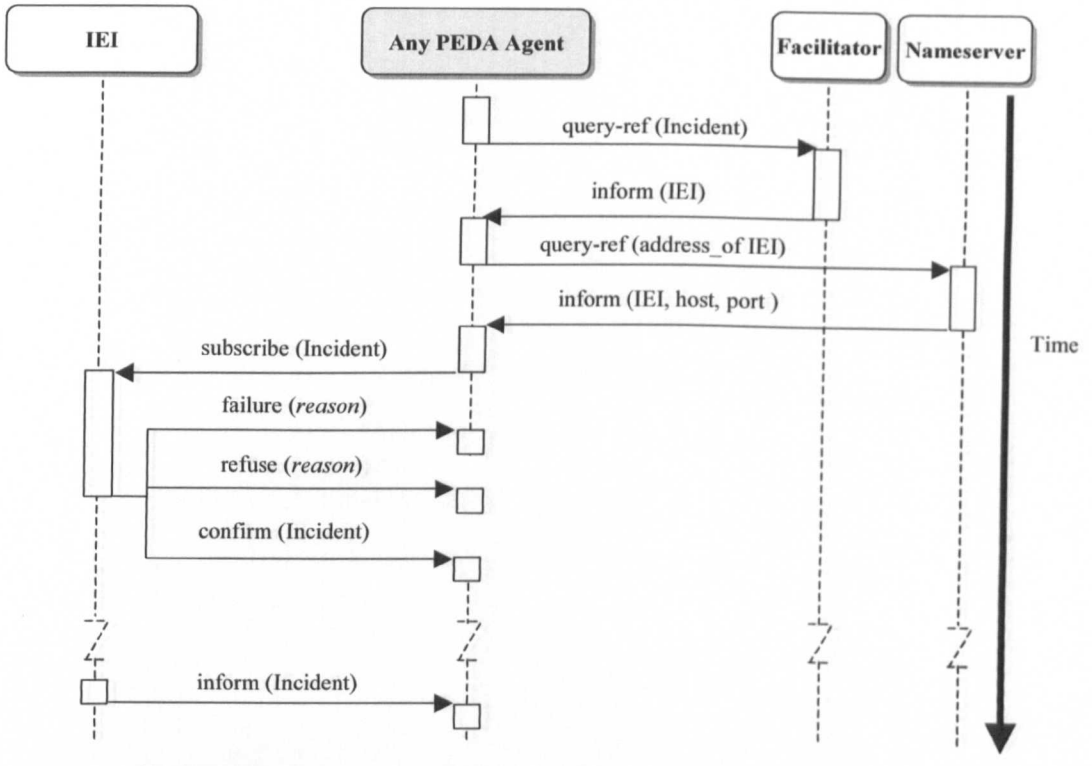
Sequence Diagram	SD_PEDA_03: Query for abilities		
Task Owner(s)	Any PEDA agent	**Initiating Task**	*Get provider*
Task Owner(s)	Facilitator	**Responding Task**	Provide Abilities
Other participants	None	**Responding Task**	N/A

F.4 SD_PEDA_04 = Query for address



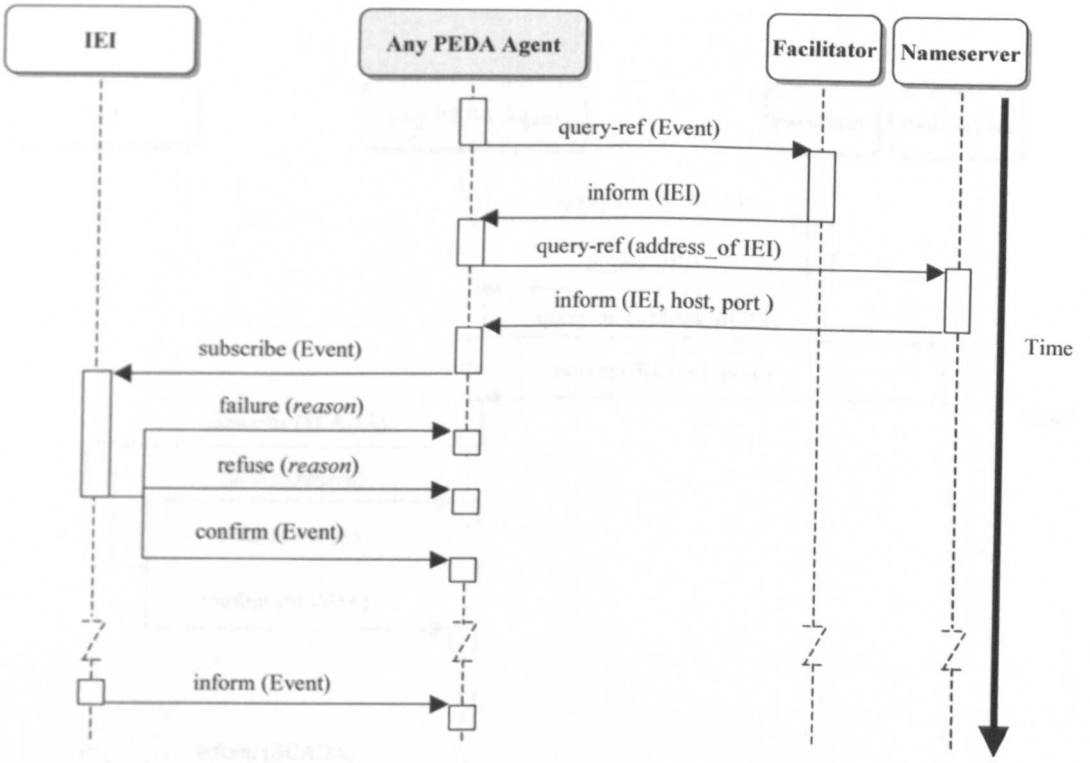
Sequence Diagram	SD_PEDA_04: Query for address		
Task Owner(s)	Any PEDA agent	**Initiating Task**	*Get address*
Task Owner(s)	Nameserver	**Responding Task**	Provide Address
Other participants	None	**Responding Task**	N/A

F.5 SD_PEDA_05 = Subscribe for Incident updates



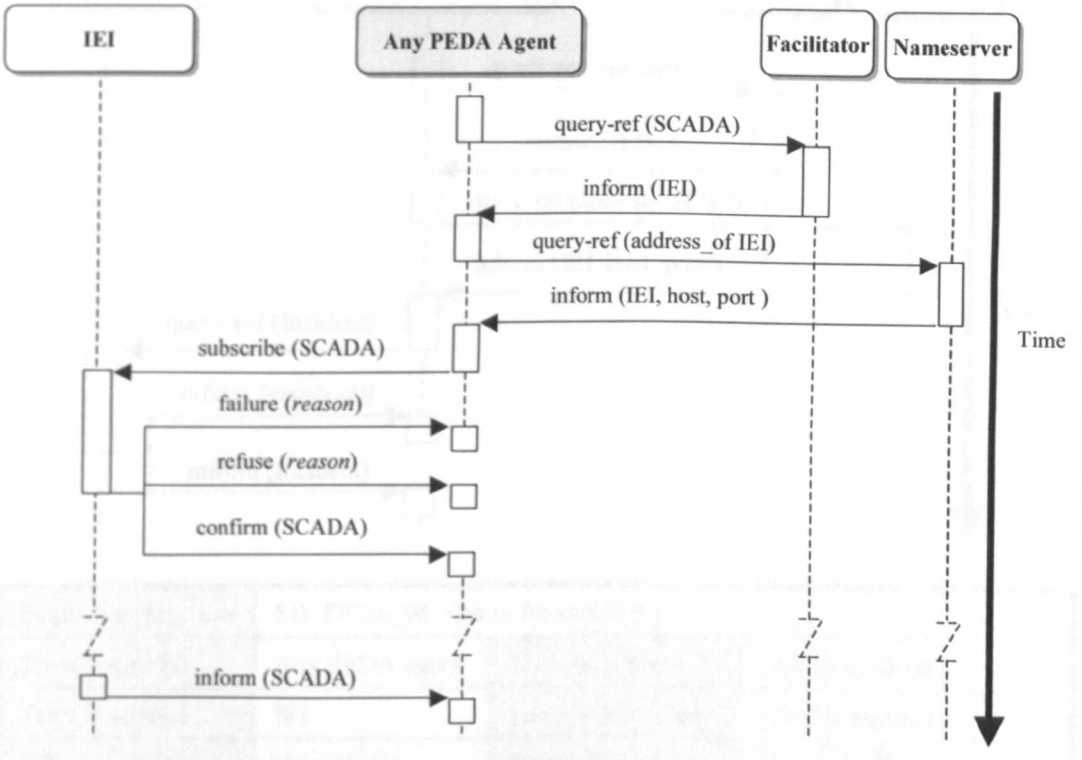
Sequence Diagram	SD_PEDA_05: Subscribe for Incident updates		
Task Owner(s)	Any PEDA Agent	**Initiating Task**	Obtain Identified Incidents
Task Owner(s)	IEI	**Responding Task**	Provide Incidents
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.6 SD_PEDA_06 = Subscribe for Event updates



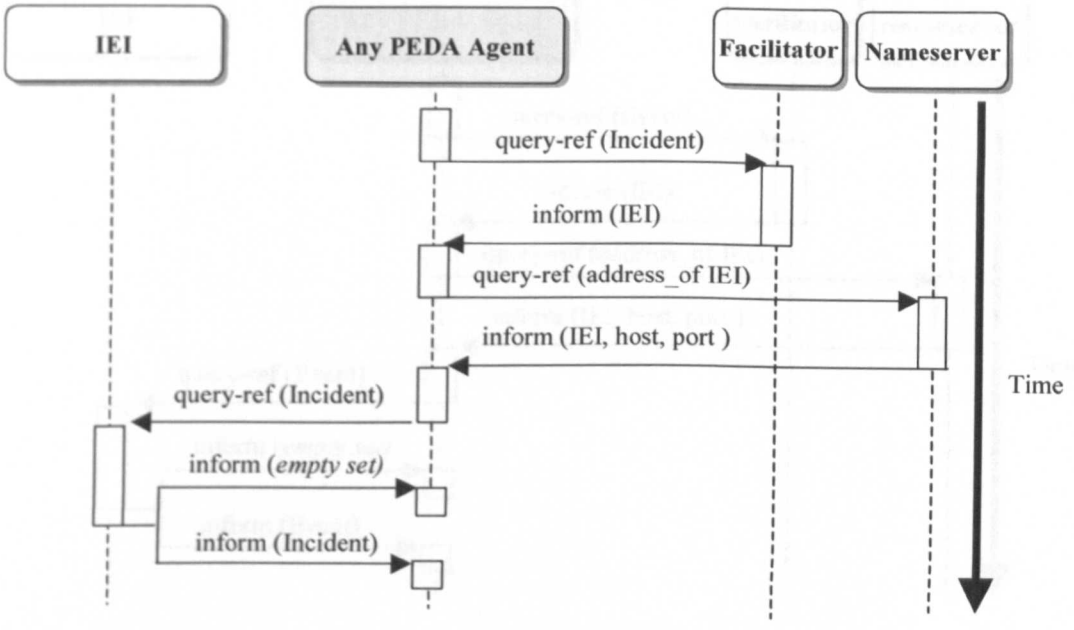
Sequence Diagram	SD_PEDA_09: Query for Events(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Identified Events
Task Owner(s)	IEI	**Responding Task**	Provide Events
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.7 SD_PEDA_07 = Subscribe for SCADA Alarm updates



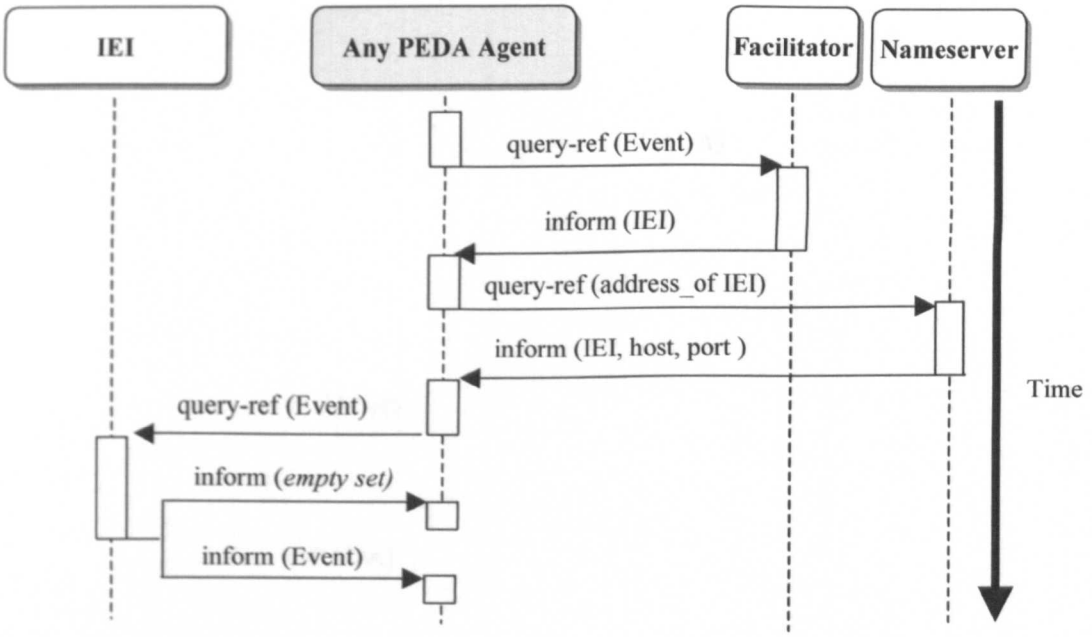
Sequence Diagram	SD_PEDA_07: Subscribe for SCADA Alarm updates		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain SCADA
Task Owner(s)	IEI	**Responding Task**	Provide SCADA
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.8 SD_PEDA_08 = Query for Incident(s)



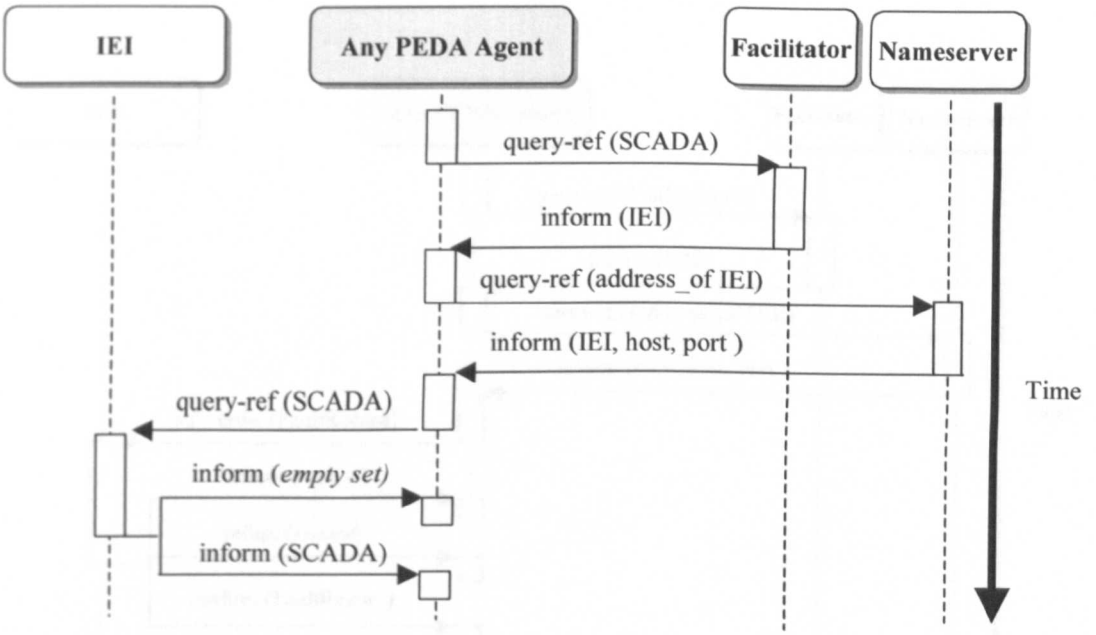
Sequence Diagram	SD_PEDA_08: Query for Incident(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Incidents
Task Owner(s)	IEI	**Responding Task**	Provide Incidents
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.9 SD_PEDA_09 = Query for Event(s)



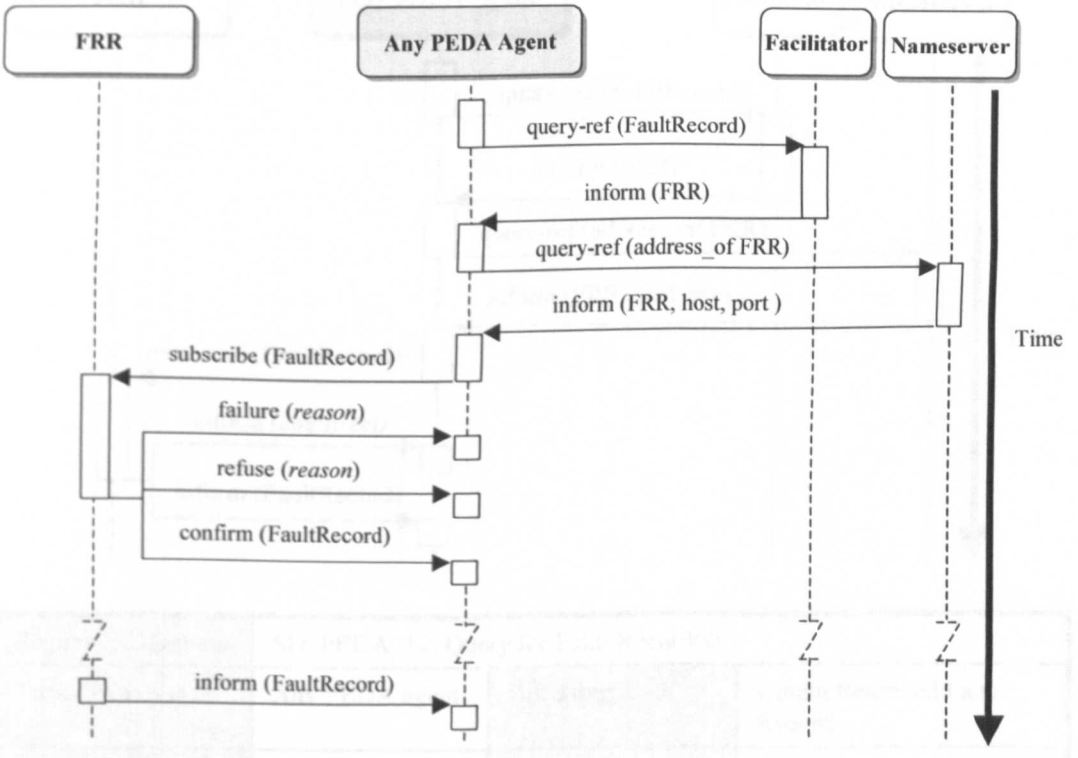
Sequence Diagram	SD_PEDA_09: Query for Events(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Events
Task Owner(s)	IEI	**Responding Task**	Provide Events
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.10 SD_PEDA_10 = Query for SCADA Alarm(s)



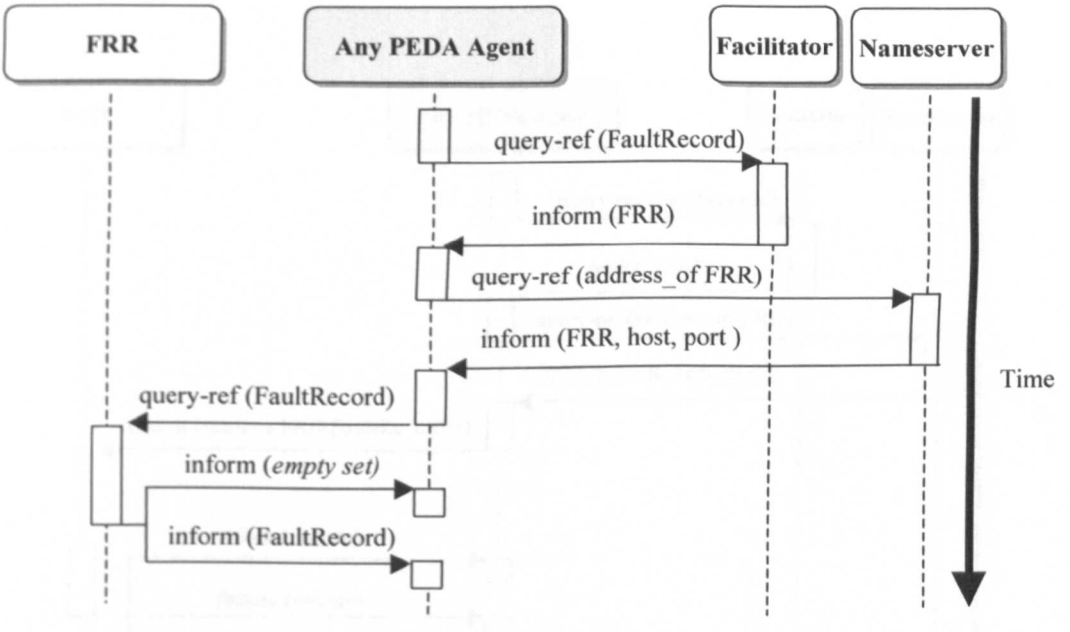
Sequence Diagram	SD_PEDA_10: Query for SCADA Alarm(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain SCADA
Task Owner(s)	IEI	**Responding Task**	Provide SCADA
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.11 SD_PEDA_11 = Subscribe for Fault Record updates



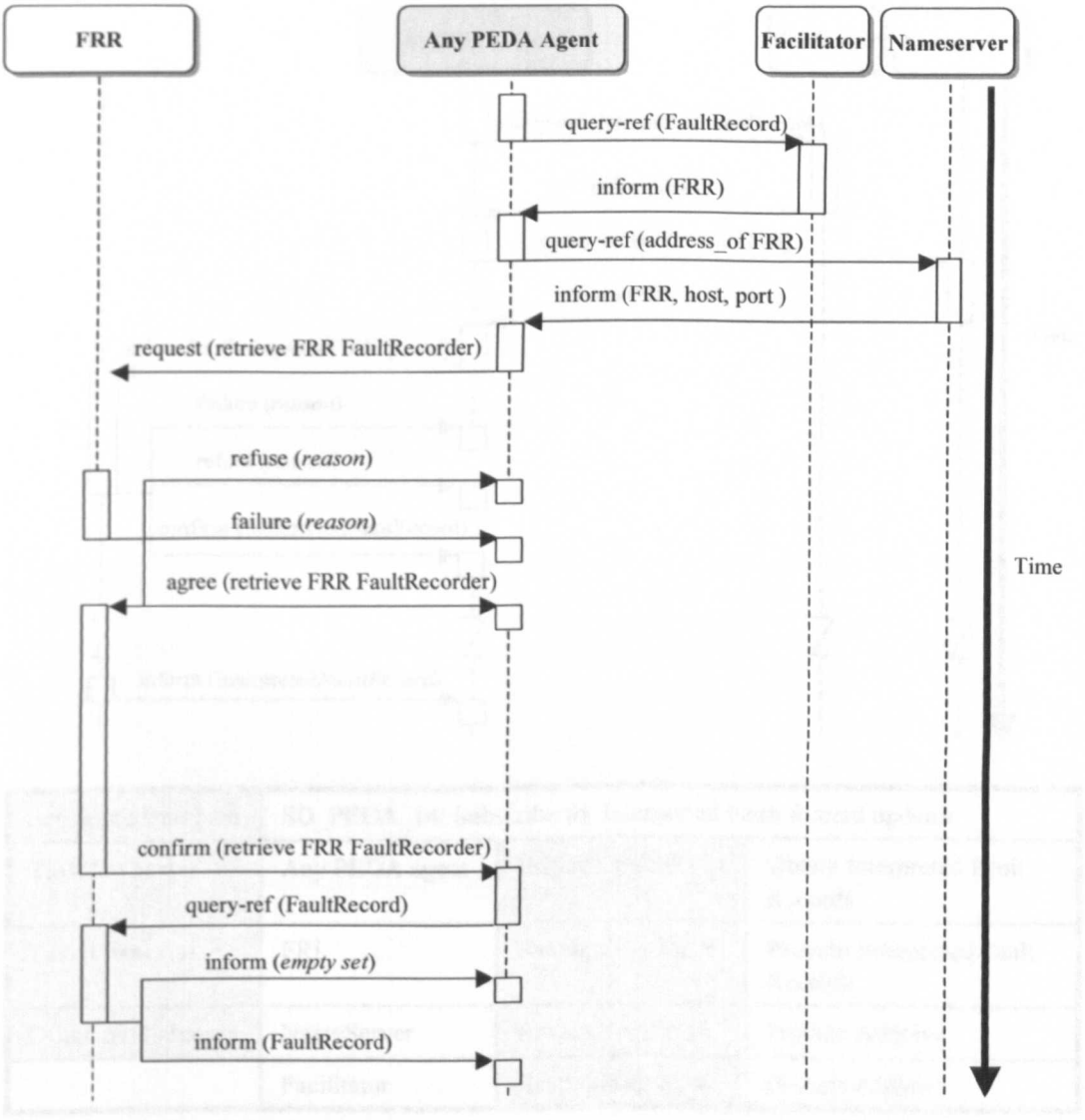
Sequence Diagram	SD_PEDA_11: Subscribe for Fault Record updates		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Retrieved Fault Records
Task Owner(s)	FRR	**Responding Task**	Provide Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.12 SD_PEDA_12 = Query for Fault Record(s)



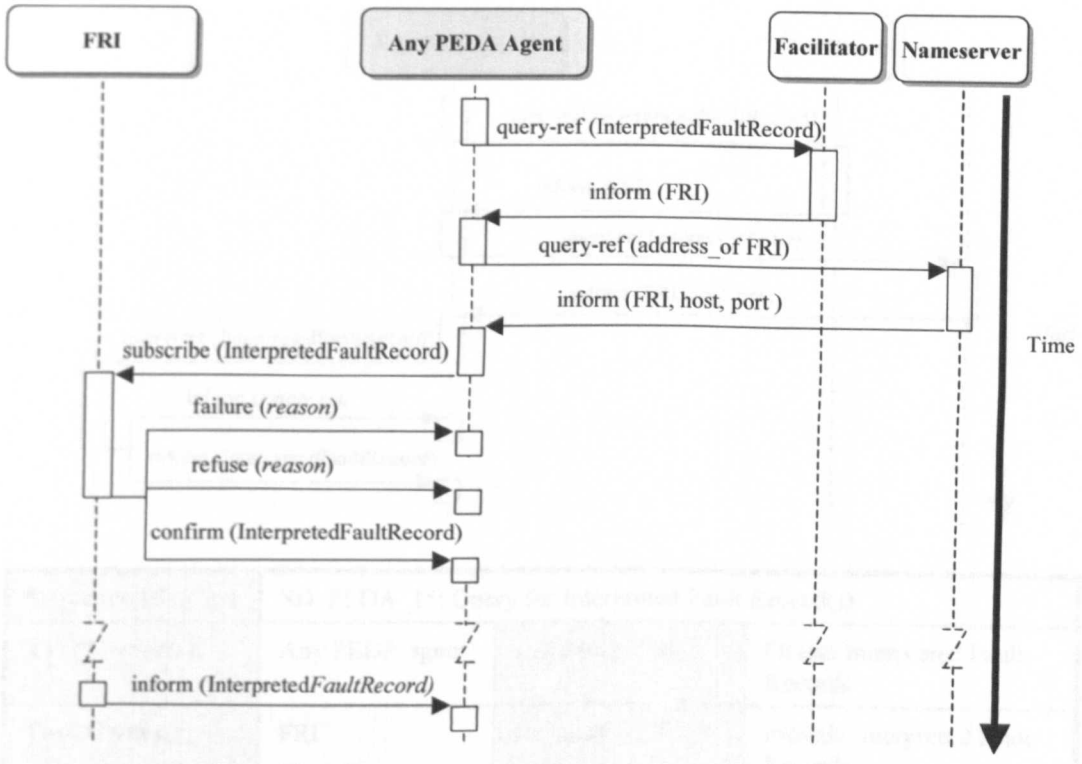
Sequence Diagram	SD_PEDA_12: Query for Fault Record(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Retrieved Fault Records
Task Owner(s)	FRR	**Responding Task**	Provide Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.13 SD_PEDA_13 = Request retrieval of Fault Record(s)



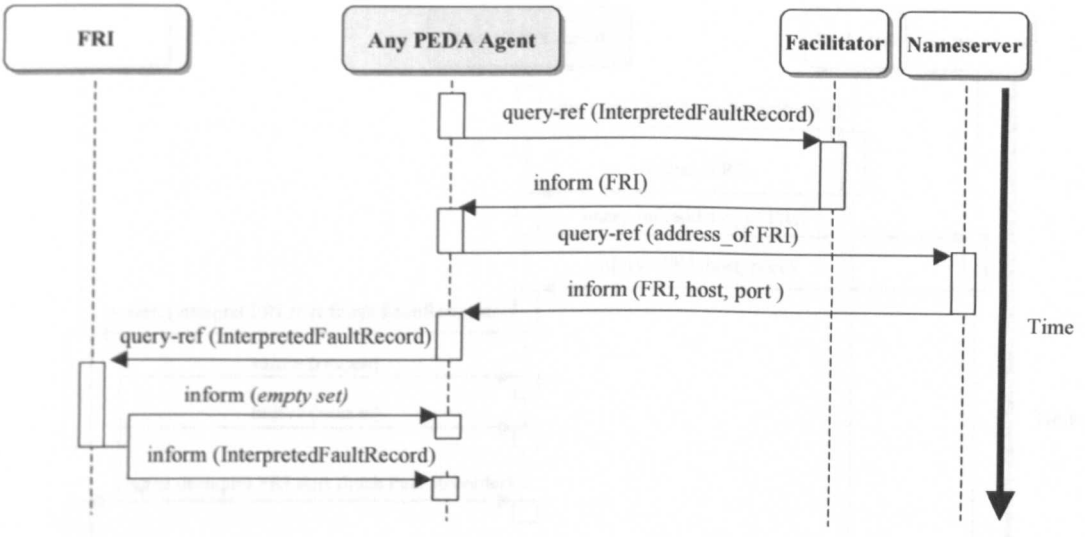
Sequence Diagram	SD_PEDA_13: Request Retrieval of Fault Record(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Retrieved Fault Records
Task Owner(s)	FRR	**Responding Task**	Provide Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.14 SD_PEDA_14 = Subscribe for Interpreted Fault Record updates



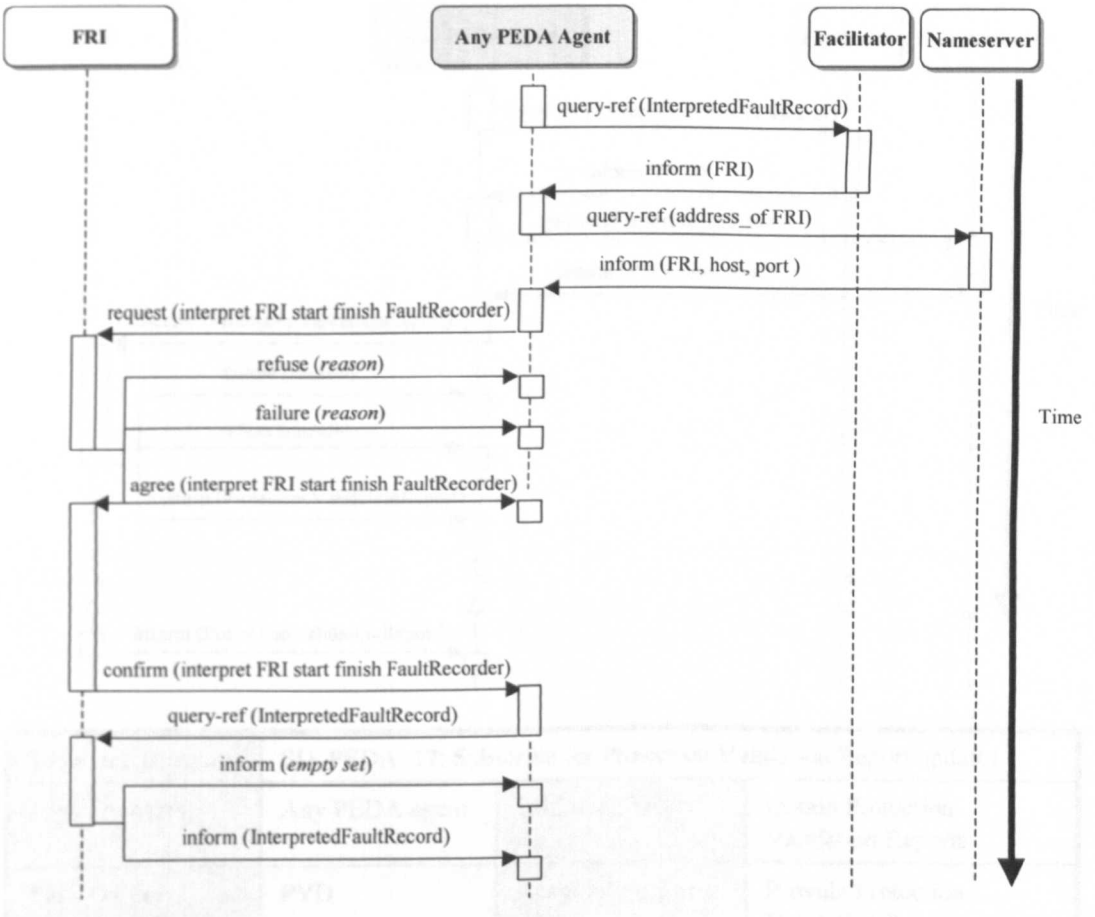
Sequence Diagram	SD_PEDA_14: Subscribe for Interpreted Fault Record updates		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Interpreted Fault Records
Task Owner(s)	FRI	**Responding Task**	Provide Interpreted Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.15 SD_PEDA_15 = Query for Interpreted Fault Record(s)



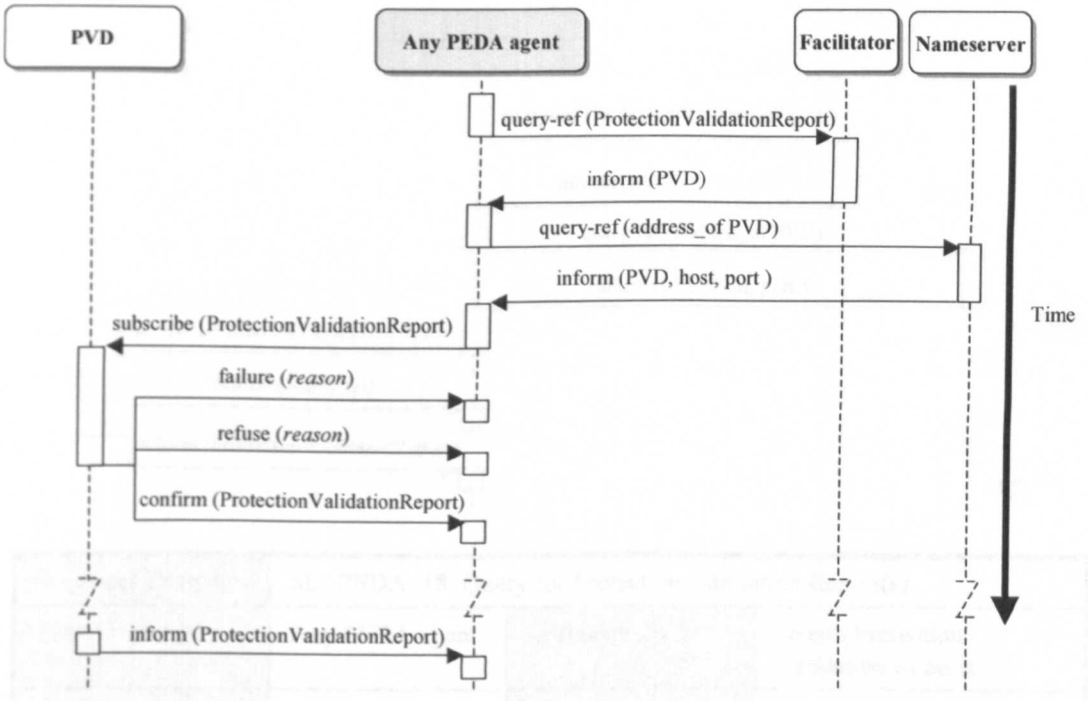
Sequence Diagram	SD_PEDA_15: Query for Interpreted Fault Record(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Interpreted Fault Records
Task Owner(s)	FRI	**Responding Task**	Provide Interpreted Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.16 SD_PEDA_16 = Request generation of Interpreted Fault Record(s)



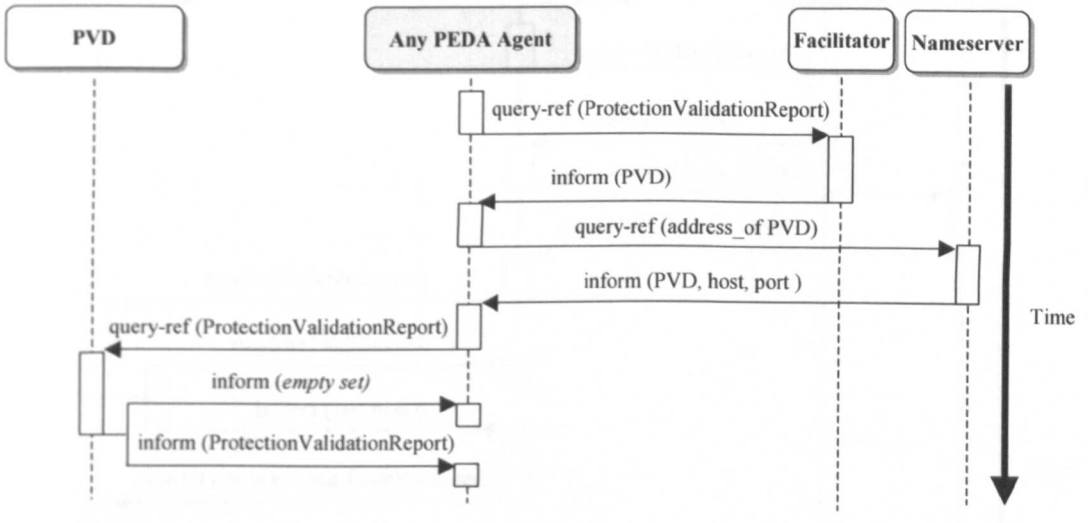
Sequence Diagram	SD_PEDA_16: Request Generation of Interpreted Fault Record(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Interpreted Fault Records
Task Owner(s)	FRI	**Responding Task**	Provide Interpreted Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.17 SD_PEDA_17 = Subscribe for Protection Validation Report updates



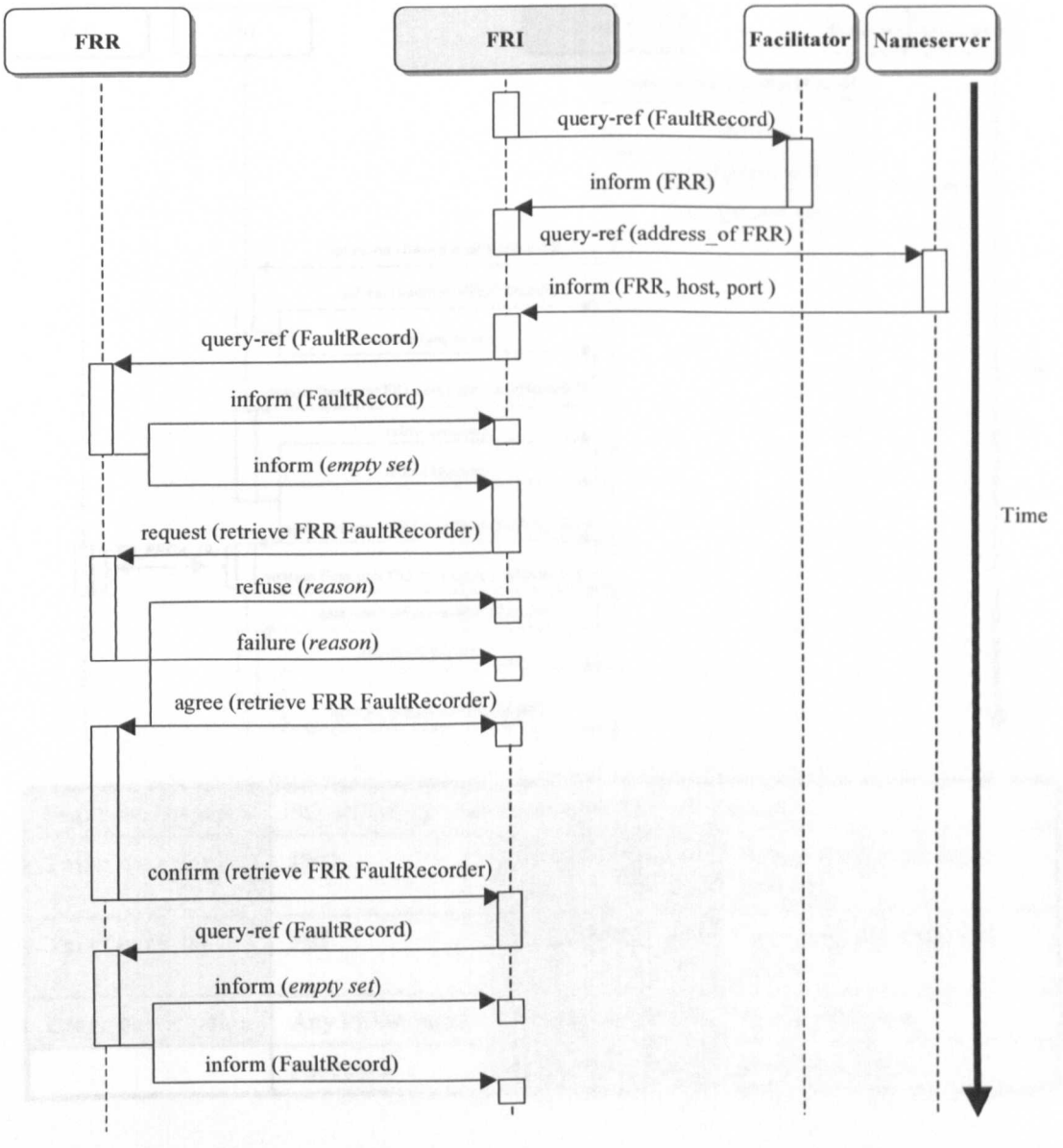
Sequence Diagram	SD_PEDA_17: Subscribe for Protection Validation Report updates		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Protection Validation Reports
Task Owner(s)	PVD	**Responding Task**	Provide Protection Validation Reports
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.18 SD_PEDA_18 = Query for Protection Validation Report(s)



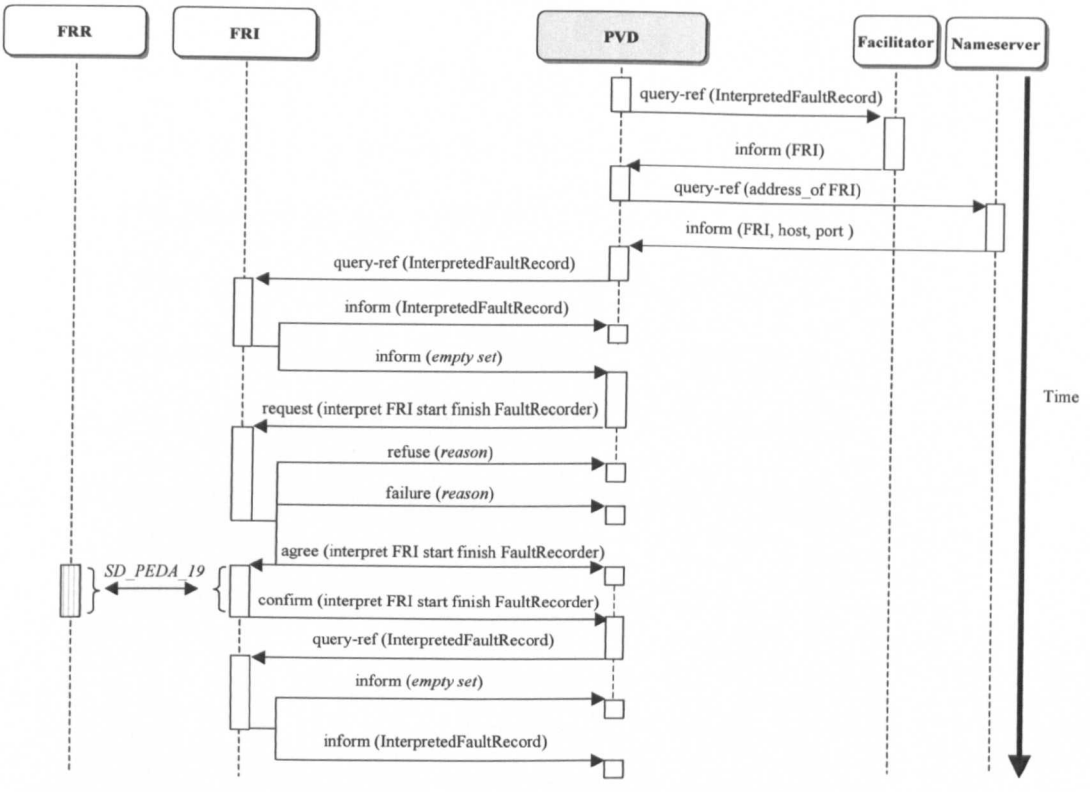
Sequence Diagram	SD_PEDA_18: Query for Protection Validation Report(s)		
Task Owner(s)	Any PEDA agent	**Initiating Task**	Obtain Protection Validation Reports
Task Owner(s)	PVD	**Responding Task**	Provide Protection Validation Reports
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.19 SD_PEDA_19 = Obtain Fault Records



Sequence Diagram	SD_PEDA_19: Obtain Fault Records		
Task Owner(s)	FRI	**Initiating Task**	Obtain Retrieved Fault Records
Task Owner(s)	FRR	**Responding Task**	Provide Fault Records
Other participants	NameServer	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

F.20 SD_PEDA_20 = Obtain Interpreted Fault Records



Sequence Diagram	SD_PEDA_20: Obtain Interpreted Fault Records		
Task Owner(s)	PVD	**Initiating Task**	Obtain Interpreted Fault Records
Task Owner(s)	FRI	**Responding Task**	Provide Interpreted Fault Records
Other participants	Any PEDA agent	**Responding Task**	Provide Address
	Facilitator	**Responding Task**	Provide Abilities

Appendix G: PEDA Message Handlers

This appendix contains the message handlers identified during the Agent Behaviour Modelling stage of the PEDA design process described in section 6.9.1 of this thesis.

The message handlers within the appendix are listed below:

G.1 IEI Reactive Behaviour: MH_IEI_01 to MH_IEI_06

G.2 IEI Proactive Behaviour: *None*

G.3 FRR Reactive Behaviour: MH_FRR_01 to MH_FRR_03

G.4 FRR Proactive Behaviour: MH_FRR_04 to MH_FRR_09

G.5 FRI Reactive Behaviour: MH_FRI_01 to MH_FRI_03

G.6 FRI Proactive Behaviour: MH_FRI_04 to MH_FRI_23

G.7 PVD Reactive Behaviour: MH_PVD_01 to MH_PVD_02

G.8 PVD Proactive Behaviour: MH_PVD_03 to MH_PVD_16

G.9 Message Handlers – IEI Agent

G.9.1 IEI Reactive Behaviour

Agent	IEI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_05 – Subscribe for Incident updates		
Responding Task	Provide Incidents		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_IEI_01	subscribe	Incident	• If subscribe message cannot be understood, then o Send failure (*reason*) to sending agent • If subscribing agent does not have correct access privileges, then o Send refuse (*reason*) to sending agent • If subscribing agent message can be understood and subscribing agent does have correct access privileges, then o Send confirm (Incident) to subscribing agent o Create a 'subscription rule' to monitor for new Incident data and automatically inform subscribed agent

Agent	IEI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_06 – Subscribe for Event updates		
Responding Task	Provide Events		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_IEI_02	subscribe	Event	• If subscribe message cannot be understood, then o Send failure (*reason*) to sending agent • If subscribing agent does not have correct access privileges, then o Send refuse (*reason*) to sending agent • If subscribing agent message can be understood and subscribing agent does have correct access privileges, then o Send confirm (Event) to subscribing agent o Create a 'subscription rule' to monitor for new Event data and automatically inform subscribed agent

Agent	IEI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_07 – Subscribe for SCADA Alarm updates		
Responding Task	Provide SCADA		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_IEI_03	subscribe	SCADA	• If subscribe message cannot be understood, then o Send failure (*reason*) to sending agent • If subscribing agent does not have correct access privileges, then o Send refuse (*reason*) to sending agent • If subscribing agent message can be understood and subscribing agent does have correct access privileges, then o Send confirm (SCADA) to subscribing agent o Create a 'subscription rule' to monitor for new SCADA data and automatically inform subscribed agent

Agent	IEI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_08 – Query for Incident(s)		
Responding Task	Provide Incidents		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_IEI_04	query-ref	Incident	• Check for Incident data matching query-ref message content. • If no matching Incident data, then o Send inform (empty set) message to querying agent • If matching Incident data found, then o Send inform (Incident) message to querying agent containing the matching data set.

Agent	IEI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_09 – Query for Event(s)		
Responding Task	Provide Events		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_IEI_05	query-ref	Event	• Check for Event data matching query-ref message content. • If no matching Event data, then o Send inform (empty set) message to querying agent • If matching Event data found, then o Send inform (Event) message to querying agent containing the matching data set.

Agent	IEI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_10 – Query for SCADA Alarm(s)		
Responding Task	Provide SCADA		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_IEI_06	query-ref	SCADA	• Check for SCADA data matching query-ref message content. • If no matching SCADA data, then o Send inform (empty set) message to querying agent • If matching SCADA data found, then o Send inform (SCADA) message to querying agent containing the matching data set.

G.9.2 IEI Proactive Behaviour

N/A - The IEI agent does not exhibit any proactive behaviour.

G.10 Message Handlers – FRR Agent

G.10.1 FRR Reactive Behaviour

Agent	FRR	**Behaviour**	Reactive
Sequence Diagram	SD_PEDA_11 – Subscribe for Fault Record updates		
Responding Task	Provide Fault Records		

Message Handler ID	**Incoming Message**		**Response**
	Type	**Content**	
MH_FRR_01	subscribe	FaultRecord	• If subscribe message cannot be understood, then o Send failure (*reason*) to sending agent • If subscribing agent does not have correct access privileges, then o Send refuse (*reason*) to sending agent • If subscribing agent message can be understood and subscribing agent does have correct access privileges, then o Send confirm (FaultRecord) to subscribing agent o Create a 'subscription rule' to monitor for new FaultRecord data and automatically inform subscribed agent

Agent	FRR	Behaviour	Reactive
Sequence Diagram	SD_PEDA_12 – Query for Fault Record(s)		
Responding Task	Provide Fault Records		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_FRR_02	query-ref	FaultRecord	• Check for FaultRecord data matching query-ref message content. • If no matching FaultRecord data, then o Send inform (empty set) message to querying agent • If matching FaultRecord data found, then o Send inform (FaultRecord) message to querying agent containing the matching data set.

Agent	FRR	Behaviour	Reactive
Sequence Diagram	SD_PEDA_13 – Request retrieval of Fault Record(s)		
Responding Task	Provide Fault Records		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_FRR_03	request	retrieve FRR FaultRecorder	• If request message cannot be understood OR FaultRecorder is unknown, then o Send failure (*reason*) to requesting agent. • If requesting agent does not have correct request privileges, then o Send refuse (*reason*) to sending agent • If request message can be understood AND FaultRecorder is known AND requesting agent has correct request privileges, then o Send agree (retrieve FRR FaultRecorder) to requesting agent o Trigger 'Reschedule Retrieval' task in FRR o Create a 'request completion' rule to monitor for completion of requested interpretation and send a confirm (retrieve FRR FaultRecorder) message to requesting agent upon completion.

Note:

- A query-ref message handler is not required to manage the query-ref message received from the requesting agent following the sending of confirm message (see SD_PEDA_13). A suitable message handler has already been created to handle query-ref messages in FRR, namely: MH_FRR_02.

G.10.2 FRR Proactive Behaviour

Agent	FRR	Behaviour	Proactive
Sequence Diagram	SD_PEDA_05 – Subscribe for Incident updates		
Initiating Task	Obtain Identified Incidents		

Message Handler ID	Incoming Message			Response
	Type	Content	In Reply To	
MH_FRR_04	inform	*agent name*	query-ref (Incident)	• Send query-ref (address_of *agent name*) to Nameserver • Create MH_FRR_05 to handle response
MH_FRR_05	inform	*agent address*	query-ref (address_of *agent name*)	• Send subscribe (Incident) to *agent name* • Create MH_FRR_06, MH_FRR_07 & MH_FRR_08 to handle response.
MH_FRR_06	failure	*reason*	subscribe (Incident)	• Log reason for failure and reattempt later
MH_FRR_07	refuse	*reason*	subscribe (Incident)	• Log reason for refusal and reattempt later
MH_FRR_08	confirm	Incident	subscribe (Incident)	• Log successful subscription • Create MH_FRR_09 to handle inform (Incident) messages • Terminate 'Obtain Identified Incidents' task
MH_FRR_09	Inform	Incident	subscribe (Incident)	• Add Incident to memory

G.11 Message Handlers – FRI Agent

G.11.1 FRI Reactive Behaviour

Agent	FRI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_14 – Subscribe for Interpreted Fault Record updates		
Responding Task	Provide Interpreted Fault Records		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_FRI_01	subscribe	InterpretedFaultRecord	• If subscribe message cannot be understood, then o Send failure (*reason*) to sending agent • If subscribing agent does not have correct access privileges, then o Send refuse (*reason*) to sending agent • If subscribing agent message can be understood and subscribing agent does have correct access privileges, then o Send confirm (InterpretedFaultRecord) to subscribing agent o Create a 'subscription rule' to monitor for new InterpretedFaultRecord data and automatically inform subscribed agent

Agent	FRI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_15 – Query for Interpreted Fault Record(s)		
Responding Task	Provide Interpreted Fault Records		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_FRI_02	query-ref	InterpretedFaultRecord	• Check for InterpretedFaultRecord data matching query-ref message content. • If no matching IntepretedFaultRecord data, then o Send inform (empty set) message to querying agent • If matching InterpretedFaultRecord data found, then o Send inform (InterpretedFaultRecord) message to querying agent containing the matching data set.

Agent	FRI	Behaviour	Reactive
Sequence Diagram	SD_PEDA_16 – Request generation of Interpreted Fault Record(s)		
Responding Task	Provide Interpreted Fault Records		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_FRI_03	request	interpret FRI start finish FaultRecorder	• If request message cannot be understood OR FaultRecorder is unknown, then o Send failure (*reason*) to requesting agent. • If requesting agent does not have correct request privileges, then o Send refuse (*reason*) to sending agent • If request message can be understood AND FaultRecorder is known AND requesting agent has correct request privileges, then o Send agree (interpret FRI start finish FaultRecorder) to requesting agent o Trigger 'Schedule Interpretation' task in FRI o Create a 'request completion' rule to monitor for completion of requested interpretation and send a confirm (interpret FRI start finish FaultRecorder) message to requesting agent upon completion.

Note:

- A query-ref message handler is not required to manage the query-ref message received from the requesting agent following the sending of confirm message (see SD_PEDA_16). A suitable message handler has already been created to handle query-ref messages in FRI, namely: MH_FRI_02.

G.11.2 FRI Proactive Behaviour

Agent	FRI	**Behaviour**	Proactive
Sequence Diagram	SD_PEDA_05 – Subscribe for Incident updates		
Initiating Task	Obtain Identified Incidents		

Message Handler ID	**Incoming Message**			**Response**
	Type	**Content**	**In Reply To**	
MH_FRI_04	inform	*agent name*	query-ref (Incident)	• Send query-ref (address_of *agent name*) to Nameserver • Create MH_FRI_05 to handle response
MH_FRI_05	inform	*agent address*	query-ref (address_of *agent name*)	• Send subscribe (Incident) to *agent name* • Create MH_FRI_06, MH_FRI_07 & MH_FRI_08 to handle response.
MH_FRI_06	failure	*reason*	subscribe (Incident)	• Log reason for failure and reattempt later
MH_FRI_07	refuse	*reason*	subscribe (Incident)	• Log reason for refusal and reattempt later
MH_FRI_08	confirm	Incident	subscribe (Incident)	• Log successful subscription • Create MH_FRI_09 to handle inform (Incident) messages • Terminate 'Obtain Identified Incidents' task
MH_FRI_09	Inform	Incident	subscribe (Incident)	• Add Incident to memory

Agent	FRI	Behaviour	Proactive
Sequence Diagram	SD_PEDA_11 – Subscribe for Fault Record updates		
Initiating Task	Obtain Retrieved Fault Records		

Message Handler ID	Incoming Message			Response
	Type	Content	In Reply To	
MH_FRI_10	inform	*agent name*	query-ref (FaultRecord)	• Send query-ref (address_of *agent name*) to Nameserver • Create MH_FRI_11 to handle response
MH_FRI_11	inform	*agent address*	query-ref (address_of *agent name*)	• Send subscribe (FaultRecord) to *agent name* • Create MH_FRI_12, MH_FRI_13 & MH_FRI_14 to handle response.
MH_FRI_12	failure	*reason*	subscribe (FaultRecord)	• Log reason for failure and reattempt later
MH_FRI_13	refuse	*reason*	subscribe (FaultRecord)	• Log reason for refusal and reattempt later
MH_FRI_14	confirm	FaultRecord	subscribe (FaultRecord)	• Log successful subscription • Create MH_FRI_15 to handle inform (FaultRecord) messages • Terminate 'Obtain Retrieved Fault Records' task
MH_FRI_15	inform	FaultRecord	subscribe (FaultRecord)	• Add FaultRecord to memory

Agent	FRI	Behaviour	Proactive
Sequence Diagram	SD_PEDA_19 – Obtain Fault Records		
Initiating Task	Obtain Retrieved Fault Records		

Message Handler ID	Incoming Message			Response
	Type	Content	In Reply To	
MH_FRI_16	inform	*agent name*	query-ref (FaultRecord)	• Send query-ref (address_of *agent name*) to Nameserver • Create MH_FRI_17 to handle response
MH_FRI_17	inform	*agent address*	query-ref (address_of *agent name*)	• Send query-ref (FaultRecord) to *agent name* • Create MH_FRI_18 to handle response.
MH_FRI_18	inform	FaultRecord | *empty set*	query-ref (FaultRecord)	• If content = FaultRecord, then o Add FaultRecord to memory o Terminate 'Obtain Retrieved Fault Records' task • If content = *empty set*, then o Send request (retrieve FRR FaultRecorder) to *agent_name* o Create MH_FRI_19, MH_FRI_20 & MH_FRI_21 to handle response
MH_FRI_19	refuse	*reason*	request (retrieve FRR FaultRecorder)	• Log reason for failure and terminate 'Obtain Retrieved Fault Records'
MH_FRI_20	failure	*reason*	request (retrieve FRR FaultRecorder)	• Log reason for refusal and terminate 'Obtain Retrieved Fault Records'
MH_FRI_21	agree	retrieve FRR FaultRecorder	request (retrieve FRR FaultRecorder)	• Create MH_FRI_22 to handle confirm message indicating completion
MH_FRI_22	confirm	retrieve FRR FaultRecorder	N/A	• Send query-ref (FaultRecord) to *agent_name* • Create MH_FRI_23 to handle response.
MH_FRI_23	inform	FaultRecord | *empty set*	query-ref (FaultRecord)	• If content = FaultRecord, then o Add FaultRecord to memory • If content = *empty set*, then o Log no matching FaultRecords are available. • Terminate 'Obtain Retrieved Fault Records' task

G.12 Message Handlers – PVD Agent

G.12.1 PVD Reactive Behaviour

Agent	PVD	**Behaviour**	Reactive
Sequence Diagram	SD_PEDA_17 – Subscribe for Protection Validation Report updates		
Responding Task	Provide Protection Validation Reports		

Message Handler ID	**Incoming Message**		**Response**
	Type	**Content**	
MH_PVD_01	subscribe	ProtectionValidationReport	• If subscribe message cannot be understood, then o Send failure (*reason*) to sending agent • If subscribing agent does not have correct access privileges, then o Send refuse (*reason*) to sending agent • If subscribing agent message can be understood and subscribing agent does have correct access privileges, then o Send confirm (ProtectionValidationReport) to subscribing agent o Create a 'subscription rule' to monitor for new ProtectionValidationReport data and automatically inform subscribed agent

Agent	PVD	Behaviour	Reactive
Sequence Diagram	SD_PEDA_18 – Query for Protection Validation Report(s)		
Responding Task	Provide Protection Validation Reports		

Message Handler ID	Incoming Message		Response
	Type	Content	
MH_PVD_02	query-ref	ProtectionValidationReport	• Check for ProtectionValidationReport data matching query-ref message content. • If no matching ProtectionValidationReport data, then o Send inform (empty set) message to querying agent • If matching ProtectionValidationReport data found, then o Send inform (ProtectionValidationReport) message to querying agent containing the matching data set.

G.12.2 PVD Proactive Behaviour

Agent	PVD	Behaviour	Proactive
Sequence Diagram	SD_PEDA_05 – Subscribe for Incident updates		
Initiating Task	Obtain Identified Incidents		

Message Handler ID	Incoming Message			Response
	Type	Content	In Reply To	
MH_PVD_03	inform	*agent name*	query-ref (Incident)	• Send query-ref (address_of *agent name*) to Nameserver • Create MH_PVD_04 to handle response
MH_ PVD_04	inform	*agent address*	query-ref (address_of *agent name*)	• Send subscribe (Incident) to *agent name* • Create MH_PVD_05, MH_PVD_06 & MH_PVD_07 to handle response.
MH_ PVD_05	failure	*reason*	subscribe (Incident)	• Log reason for failure and reattempt later
MH_ PVD_06	refuse	*reason*	subscribe (Incident)	• Log reason for refusal and reattempt later
MH_ PVD_07	confirm	Incident	subscribe (Incident)	• Log successful subscription • Create MH_PVD_08 to handle inform (Incident) messages • Terminate 'Obtain Identified Incidents' task
MH_ PVD_08	Inform	Incident	subscribe (Incident)	• Add Incident to memory

Agent	PVD	Behaviour	Proactive
Sequence Diagram	SD_PEDA_20 – Obtain Interpreted Fault Records		
Initiating Task	Obtain Interpreted Fault Records		

Message Handler ID	Incoming Message			Response
	Type	Content	In Reply To	
MH_PVD_09	inform	*agent name*	query-ref (InterpretedFaultRecord)	• Send query-ref (address_of *agent name*) to Nameserver • Create MH_PVD_10 to handle response
MH_ PVD _10	inform	*agent address*	query-ref (address_of *agent name*)	• Send query-ref (InterpretedFaultRecord) to *agent name* • Create MH_PVD_11 to handle response.
MH_ PVD _11	inform	ProtectionValidationReport | *empty set*	query-ref (ProtectionValidationReport)	• If content = ProtectionValidationReport, then o Add ProtectionValidationReport to memory o Terminate 'Obtain Interpreted Fault Records' task • If content = *empty set*, then o Send request (interpret FRI start finish FaultRecorder) to *agent_name* o Create MH_PVD_12, MH_PVD_13 & MH_PVD_14 to handle response
MH_ PVD _12	refuse	*reason*	request (interpret FRI start finish FaultRecorder)	• Log reason for failure and terminate 'Obtain Interpreted Fault Records'
MH_ PVD _13	failure	*reason*	request (interpret FRI start finish FaultRecorder)	• Log reason for refusal and terminate 'Obtain Interpreted Fault Records'
MH_ PVD _14	agree	interpret FRI start finish FaultRecorder	request (interpret FRI start finish FaultRecorder)	• Create MH_PVD_15 to handle confirm message indicating completion
MH_ PVD _15	confirm	retrieve FRR FaultRecorder	N/A	• Send query-ref (ProtectionValidationReport) to *agent_name* • Create MH_PVD_16 to handle response.
MH_ PVD _16	inform	ProtectionValidationReport | *empty set*	query-ref (ProtectionValidationReport)	• If content = ProtectionValidationReport, then o Add ProtectionValidationReport to memory • If content = *empty set*, then o Log no matching ProtectionValdiationReport data available. • Terminate 'Obtain Interpreted Fault Records' task

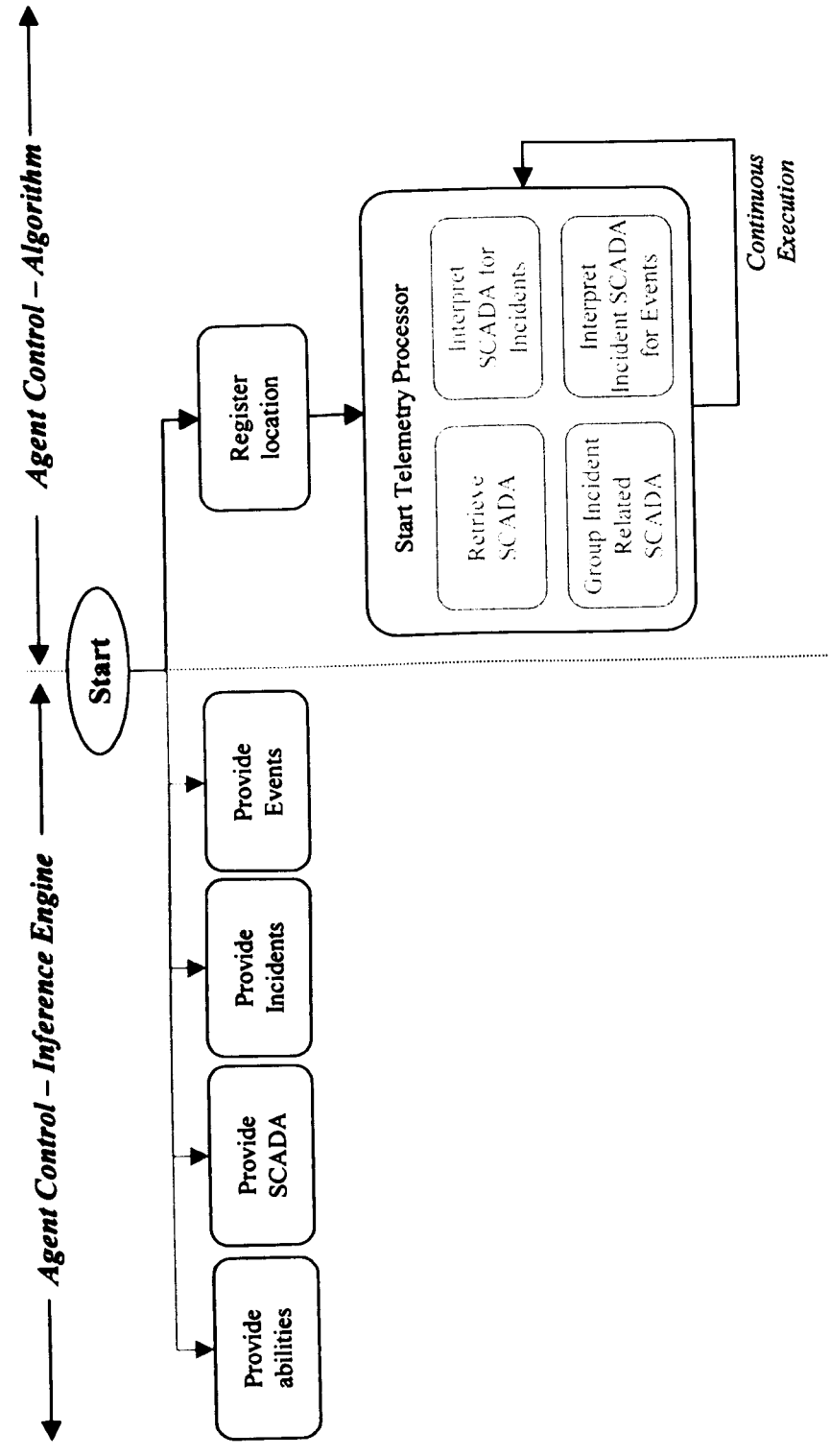
Appendix H: PEDA Agent Control Diagrams

This appendix contains the agent control diagrams created during the Agent Behaviour Modelling stage of the PEDA design process described in section 6.9.2 of this thesis.

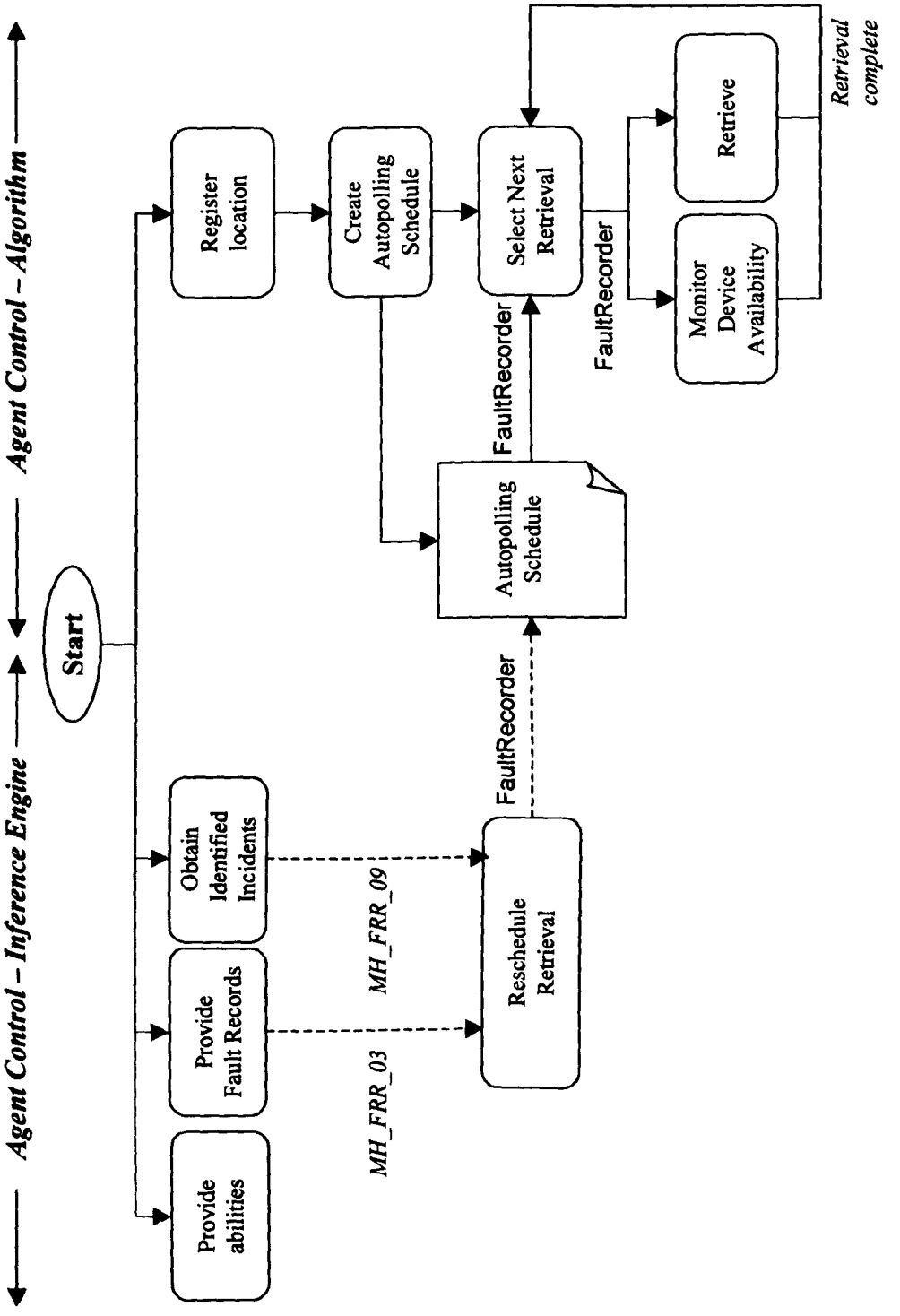
The agent control diagrams within the appendix are listed below:

- IEI Agent Control Diagram
- FRR Agent Control Diagram
- FRI Agent Control Diagram
- PVD Agent Control Diagram

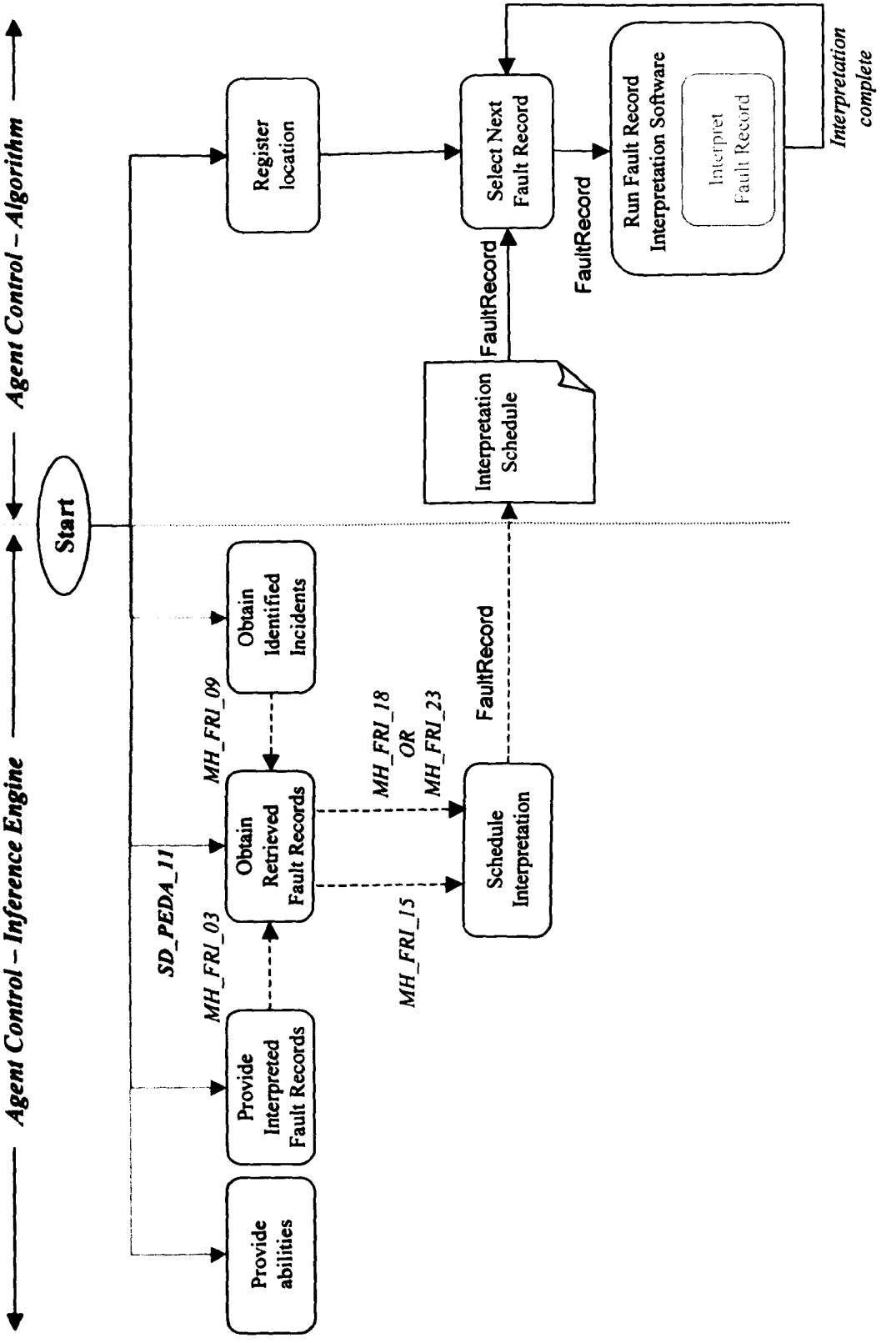
H.1 IEI Agent Control Diagram



H.2 FRR Agent Control Diagram



H.3 FRI Agent Control Diagram



H.4 PVD Agent Control Diagram