8-2008

Architecture for intelligent power systems management, optimization, and storage.

James Christopher Foreman 1968-
University of Louisville

ARCHITECTURE FOR INTELLIGENT POWER SYSTEMS MANAGEMENT, OPTIMIZATION, AND STORAGE

By

J. Chris Foreman
B.S. Electrical Engineering, University of Louisville, 1990
MENG Electrical Engineering, University of Louisville, 1996

A Dissertation
Submitted to the Faculty of the
Graduate School of the University of Louisville
In Partial Fulfillment of the Requirements
For the Degree of

Doctor of Philosophy

Department of Computer Science and Engineering
University of Louisville
Louisville, Kentucky

August 2008

ARCHITECTURE FOR INTELLIGENT POWER SYSTEMS MANAGEMENT, OPTIMIZATION AND STORAGE

By

J. Chris Foreman
B.S. Electrical Engineering, University of Louisville, 1990
MENG Electrical Engineering, University of Louisville, 1996

A Dissertation Approved on

August 2008

By the following Dissertation Committee:

____
Dissertation Director

____

____

____

____

ACKNOWLEDGEMENTS

The author would like to acknowledge Dr. Rammohan K. Ragade, who served as the dissertation advisor for his support throughout the research, publications, and final dissertation development. The committee members, Dr. Ellen G. Brehob, Dr. Ahmed H. Desoky, Dr. Adel S. Elmaghraby, and Dr. James H. Graham who have also given their support.

The author would like to acknowledge the University of Louisville Resources for Academic Achievement (REACH) Organization who provided a graduate assistantship that paid tuition, health care, and a stipend to enable the author to focus on and complete the dissertation. This graduate assistantship also provided the author with direct tutoring and mentoring experience in higher education that has been invaluable in building the skills to teach and train students.

The author acknowledges Duke Energy (then Cinergy) for trusting the author as lead engineer for four coal-fired neural network optimization projects, among other projects, with a budget in excess of $1.5MM that served as motivation for the author's development of a better power management architecture.

The author finally acknowledges his friends and family for their continued support and enthusiasm in completing the dissertation.

ABSTRACT

ARCHITECTURE FOR INTELLIGENT POWER SYSTEMS MANAGEMENT, OPTIMIZATION, AND STORAGE

J. Chris Foreman

August 2008

The management of power and the optimization of systems generating and using power are critical technologies. A new architecture is developed to advance the current state of the art by providing an intelligent and autonomous solution for power systems management. The architecture is two-layered and implements a decentralized approach by defining software objects, similar to software agents, which provide for local optimization of power devices such as power generating, storage, and load devices. These software device objects also provide an interface to a higher level of optimization. This higher level of optimization implements the second layer in a centralized approach by coordinating the individual software device objects with an intelligent expert system thus resulting in architecture for total system power management. In this way, the architecture acquires the benefits of both the decentralized and centralized approaches.

The architecture is designed to be portable, scalable, simple, and autonomous, with respect to devices and missions. Metrics for evaluating these characteristics are also defined. Decentralization achieves scalability and simplicity through modularization using software device objects that can be added and deleted as modules based on the devices of the power system are being optimized. Centralization coordinates these software device objects to bring autonomy and intelligence of the whole power system and mission to the architecture. The centralization approach is generic since it always coordinates software device objects; therefore it becomes another modular component of the architecture.

Three example implementations illustrate the evolution of this power management system architecture. The first implementation is a coal-fired power generating station that utilized a neural network optimization for the reduction of nitrogen oxide emissions. This illustrates the limitations of this type of black-box optimization and serves as a motivation for developing a more functional architecture. The second implementation is of a hydro-generating power station where a white-box, software agent approach illustrates some of the benefits and provides initial justification of moving towards the proposed architecture. The third implementation applies the architecture to a vehicle to grid application where the previous hydro-generating application is ported and a new hybrid vehicle application is defined. This demonstrates portability and scalability in the architecture, and linking these two applications demonstrates autonomy. The simplicity of building this application is also evaluated.

TABLE OF CONTENTS

PAGE

APPROVAL / SIGNATURES ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

LIST OF TABLES x

LIST OF FIGURES xii

CHAPTER I 1

INTRODUCTION 1

What are Power Management Systems? 1

Problem Description 2

Architecture Description as a Solution 3

Why is Power Management Important? 4

Motivation for the Architecture 5

Brief Outline of Dissertation 6

CHAPTER II 8

LITERATURE REVIEW 8

Research in Coal-Fired Power Plant Software Optimization 8

Commercial Software Optimization Products for Power Generating Plants 13

Case Studies at Various Coal-fired Generating Stations 19

Sample Cost Comparisons for Hardware vs. Software Nitrogen Oxide Reduction Efforts 26
Research in Hydro Power Generating Plants in Particular 30
Research in Enterprise-level and Business Solutions 32
Research in Vehicular Systems for Power Management 34
CHAPTER III 39
ARCHITECTURE 39
Evolution of the Architecture 40
Quantifying the Criteria of the Architecture 42
Quantifying Portability 43
Quantifying Scalability 45
Quantifying Simplicity 48
Quantifying Autonomy 50
A Layered Approach 55
The Device Layer 57
The System Layer 65
Integrating the Device and System Layers Together 70
Use Cases of the Architecture 73
Additional Layers of Enhancement 76
Data Mining Layer 76
Business Entities and the Enterprise-level Layer 77
Conflict Resolution in the Architecture 79
State Transitions of the Power System Utilizing this Architecture 81

Communications Timing in the Architecture 83

CHAPTER IV 85

IMPLEMENTATION 85

Power Generating Plants vs. Vehicular Systems 86

Considerations for Power Generating Plants 86

Proprietary Systems in Power Generation Control 88

Proportional Integral Differential Control - PID 90

Considerations for Safe and Reliable Operation 91

Considerations for Vehicular Systems 92

Motivations for Vehicular Power Management Systems 92

Classifications of Vehicles and Architectural Considerations 93

Vehicular Control Systems Environments 96

Example Application of Nitrogen Oxide Reduction for a Coal-fired Boiler 97

Plant description 97

Application Architecture 97

Applying the Metrics to the Application 99

Results and Closing Remarks for the Application 107

Example Application for a Hydro-generating Plant 108

Plant Description 109

Existing Control Scheme 110

Integration of the Software Agent 111

Use Cases of the Application 112

Development of the Rule Sets 114

Results 123
Applying the Metrics to the Application 128
Closing Remarks for the Application 135
The Architecture Applied to a Power Plant with Hybrid Vehicle Coupling 138
Overview 138
Hydro-Generating Plant Application 140
Hybrid Vehicle Application 147
Coupling the Applications' System Layers 155
Software Metrics of the Applications 161
Closing Remarks for the Applications 167
CHAPTER V 169
CONCLUSIONS AND FUTURE DIRECTIONS 169
The Architecture as a Solution 169
Scalability in the Applications 171
Security in Scalable Applications 172
Portability in the Applications 172
Simplicity in the Applications 173
Autonomy in the Applications 174
Future Directions 176
Closing the Loop in Consumer Power Generation 176
Application of the Architecture to a Picosatellite 177
REFERENCES 181
CURRICULUM VITAE 196

LIST OF TABLES

PAGE

Table 3.1. Scalability factors. 46

Table 3.2. Readability complexity. 49

Table 3.3. Operator independence. 51

Table 3.4. Self-preservation. 52

Table 3.5. Strategy. 52

Table 3.6. Coordination. 53

Table 4.1. Application controllables. 98

Table 4.2. Neural network application portability. 100

Table 4.3. Neural network application scalability. 101

Table 4.4. Neural network application difficulty. 103

Table 4.5. Neural network application scope 104

Table 4.6. Software agent application portability. 129

Table 4.7. Software agent application scalability. 131

Table 4.8. Software agent application difficulty 132

Table 4.9. Rule sets in the hydro-generating application 141

Table 4.10. Status method for the hydro-generating application. 143

Table 4.11. Neural network configuration of hydro-generating application. 145

Table 4.12. Power flow relative quantities. 150

Table 4.13. Neural network configuration of hybrid vehicle application. 151

Table 4.14. Modes of coordination. 156

Table 4.15. Coupled applications portability. 161

Table 4.16. Coupled applications scalability. 162

Table 4.17. Coupled applications difficulty. 163

LIST OF FIGURES

PAGE

Figure 2.1. Data flow in optimization software products. 19

Figure 2.2. Initial installation costs of compared technologies. 28

Figure 2.3. Annual operating costs of compared technologies. 29

Figure 2.4. Effectiveness of compared technologies at reducing nitrogen oxide. 29

Figure 2.5. Relative costs for the equivalent nitrogen oxide reduction. 30

Figure 2.6. Selection of best characteristics 37

Figure 3.1. The portability metric. 45

Figure 3.2. The scalability metric. 47

Figure 3.3. The autonomy metric. 54

Figure 3.4. Layered approach. 57

Figure 3.5. Software device object architecture 59

Figure 3.6. Software device object architecture expanded with optimization. 62

Figure 3.7. Software device object as a software agent. 64

Figure 3.8. System layer coordinating multiple software device objects. 67

Figure 3.9. System layer enhanced with neural network classification. 69

Figure 3.10. Use case for human users of the architecture. 73

Figure 3.11. Use case for power devices of the architecture. 74

Figure 3.12. Power generating enterprise example 78

Figure 3.13. Data mining and enterprise-level layers. 79

Figure 3.14. Sample power system state transition scenario. 82

Figure 3.15. Data pathway timing for industrial control system. 84

Figure 4.1. Industrial control system overview. 87

Figure 4.2. Functional architecture of DPU and PLC 88

Figure 4.3. Sample of function block diagram or SAMMA. 89

Figure 4.4. Sample of ladder logic diagram 90

Figure 4.5. Vehicle types and examples. 94

Figure 4.6. Neural network application architecture. 98

Figure 4.7. Neural network application autonomy. 107

Figure 4.8. Typical hydro unit with primary variables. 110

Figure 4.9. Existing control scheme. 111

Figure 4.10. Individual unit integration. 111

Figure 4.11. Modules in the software agent 112

Figure 4.12. Use case diagram for a single agent. 113

Figure 4.13. Multi-agent use case diagram. 114

Figure 4.14. Optimal points of operation 115

Figure 4.15. Hill curve of operating space and limits 120

Figure 4.16. Load vs. flow 124

Figure 4.17. Load vs. flow redistribution. 125

Figure 4.18. Response to trouble condition. 127

Figure 4.19. Software agent application autonomy metric. 135

Figure 4.20. Overview of hydro plant to hybrid vehicle coupling.................................. 140

Figure 4.21. Hybrid vehicle power flow.. 149

Figure 4.22. Neural network interpolative classification example................................ 152

Figure 4.23. User preferences interface for coupling application.................................. 157

Figure 4.24. Autonomy metric of the coupled applications.. 167

Figure 5.1. Autonomy in decision-making... 175

Figure 5.2. Closing the loop in consumer power generation... 177

Figure 5.3. Picosatellite power management system... 179

CHAPTER I

INTRODUCTION

What are Power Management Systems?

Management is a formalized approach to achieving the desired mission. Power management refers to the managing of the devices in a power system. Therefore, power management provides a formal approach to utilizing the power system to achieve its mission within the mission of the whole system or process. While there are many solutions to accomplish this, the desired path should be the optimal path in a responsible management approach. Optimization refers to finding the best-fit solution given a set of criteria. This is typically a balanced solution based on multiple, weighted criteria. Management supervises this optimizing process by collecting the criteria and boundary conditions from the users, application and environment to achieve a solution that most satisfies the overall mission. Management also includes the responsibilities of observing the status of the optimization to verify the solutions and handle unknown or trouble conditions. Therefore, power optimization is a tool of power management. Power management systems are the architecture implementing the management, optimization, and storage strategies.

Problem Description

There has been much work on optimization of power processes and the development of power management systems. The processes being optimized and the systems being managed include a diverse range of missions; however they share some common threads. Power needs to be generated as efficiently as possible to minimize costs and reduce negative environmental impacts. Power also needs to be used as efficiently as possible for these same reasons. Lastly, power needs to be stored for use in times when generation is limited or unavailable. Many devices have been introduced into power systems with hardware advancements occurring all the time. The dynamics of adding and removing these devices in a power system adds another dimension of complexity. Software-based management solutions have attempted to incorporate these devices to provide an optimal solution for the mission at hand.

The approaches thus far can be categorized into two groups, centralized and decentralized architectures. Centralized architectures know the whole system and have the benefit of superior coordination, but at the cost of being the most complex and specialized of solutions [Vahidi, 2007]. Decentralization attempts to break the problem into smaller pieces to achieve a simpler solution, but at the cost of coordination [Vahidi, 2007]. The preferred path has been to take the decentralized approach and attempt some form of coordination of the pieces to get back to a whole system solution. While there has been success in these attempts, limitations still exist.

The need is for an architecture that has the characteristics of: scalability – for growing with the application and the mission expands; portability – to apply the architecture to a wide range of devices and missions; autonomy – for missions where user interaction is limited; and simplicity – to enable the solution to be implemented by experts in the field and maintained by maintenance personnel. Metrics for quantifying these are defined in *Chapter III* and applied in *Chapter IV.*

Architecture Description as a Solution

Architecture is developed for power systems management. The architecture is realized in two layers. The first layer implements a decentralized approach by defining software objects, similar to software agents, which provide for local optimization of power devices such as power generating, storage, and load devices. These software device objects also provide an interface to a higher level of optimization. This higher level of optimization implements the second layer in a centralized approach by coordinating the individual software device objects with a rule-based expert system. This results in a solution that is intelligent for the whole power system while being constructed of modular pieces that are simple and distributed. In this way, the architecture acquires the benefits of both the de/centralized approaches.

Because the software objects in the first layer are only responsible for their single power system device, they can be quickly developed and are portable to other power management systems whose power systems utilize the same device. The scalable and portable aspects of the architecture also address the problem of adding and removing devices. Management is achieved by coordinating all software device objects in the whole power system. By utilizing a rule-based expert system, an intelligent and autonomous solution is achieved. Rules are a natural way for human experts to think about optimization and management and therefore simplify the implementation process. Rules are also modular themselves and can be added, modified, and deleted without significant change to the architecture.

Why is Power Management Important?

Power is a limited resource that is generated and utilized in many ventures. This generation and utilization provides certain benefits and comes at certain costs. In many cases, a mission is severely limited or not possible without an optimal management approach to balance these benefits and costs. Because of these, the importance of power management and optimization is directly proportional to the importance of the mission utilizing the power resource.

The proposed architecture is important because it provides a framework for implementing a power management system that enables optimal power management. The simplicity enables the architecture to be developed quickly and cheaply. The intelligence allows the architecture to be effective. The autonomy allows the architecture to function automatically without significant user guidance or interaction. These qualities are important because they become mission-enabling characteristics. For example, a small satellite operates in a severely power-limited environment with minimal opportunity for user interaction. A hybrid vehicle needs to provide long-range use, minimal environmental impact, high reliability, and low cost to be a marketable product. Power generating plants need to be operated at peak efficiency and again with minimal environmental impact, since the economies of scale make small gains or losses at these facilities result in huge benefits or costs. Without power management and power optimization, many of these missions would be difficult or impossible.

Motivation for the Architecture

There are several approaches to software-based power optimization and management. Much of this work in the power generation industry has been achieved with model predictive control or neural network optimization. These approaches require much work to implement and do not handle multiple goals or changing conditions well. The author has performed several neural network optimizations at power generating plants for emissions reductions and efficiency improvements. While good results were obtained, e.g. approximately 20% average reduction in nitrogen oxide emissions by software optimization alone, the implementation was difficult, requiring large training sets and much time spent validating process data patterns for these sets. Once the optimization was completed, changes in goals, in the process equipment, or other conditions were not handled well and required total retraining of the neural network. There had to be a better way. In vehicular power systems, newer approaches had been successfully implemented. These incorporated software agents and other object-oriented structures for autonomous and intelligent decision-making solutions. These served as an inspiration for the problems encountered in the power industry; however, these vehicular systems were designed for small mobile implementations. Taking the best components of each approach, a scalable architecture was developed in *Chapter III*, which used a layered approach to incorporate the mission as needed. A neural network was still used but only for pre-classification and of a smaller size. Decisions were made by a rule-based expert system to provide a white-box solution, which could be modified one rule at a time. At the lowest level, the concept of software device objects was created to provide a local software interface to the power system's hardware components. This resulted in a solution that was scalable, portable, and more autonomous than before while still being simple to understand and maintain. Once the architecture was in place, additional layers could be added for enterprise-level optimizations and beyond.

Brief Outline of Dissertation

Chapter II will review the literature for current work relating to the proposed work in power management systems. In addition to reviewing the literature, notes are made illustrating how the proposed work utilizes and enhances the current state of the art. *Chapter III* will define and develop the architecture and derive some methods by which the metrics of portability, scalability, simplicity, and autonomy can be comparably quantified. *Chapter IV* will discuss considerations for implementing the architecture in real-world systems. Three implementation cases are presented to illustrate the motivation, development, and application of the architecture. The first case is a coal-fired steam-boiler generating plant optimization for emission reduction utilizing a monolithic neural network. The limitations of this approach are discussed and this will serve as a motivation for developing the architecture. The second case is a hydro-generation plant optimization for efficiency using multiple software agents. This will introduce some aspects of the architecture developed in *Chapter III*. The third case is a vehicle to grid application using the hydro-generating plant coupled to a personal hybrid vehicle to demonstrate a full implementation of the architecture. *Chapter V* will provide final discussion of the architecture and suggest future directions.

CHAPTER II

LITERATURE REVIEW

Software optimization research for coal-fired power plants will first be discussed along with commercial applications and case studies. Hydro-generating plants in particular will then be discussed as a special topic to power generating plants. Vehicular power management systems will then be discussed to build on the power optimization and management theme of the dissertation. Finally, research in enterprise-level business entity software optimization and management systems are discussed briefly.

Research in Coal-Fired Power Plant Software Optimization

The major motivations for optimization at coal-fired generating plants are efficiency, emissions reductions, and availability. Efficiency typically refers to generating the maximum amount of power with the minimal input of fuel. The measure for this is *heat rate*, which is a ratio of power generation divided by fuel burned expressed in units of kilowatts per million BTU. Software optimizations for efficiency therefore attempt to burn fuel more completely and capture the heat released from the fuel more effectively. Reducing auxiliary loads are also included in this optimization. Emissions reduction has become an increasingly important topic. The combustion process releases several pollutants in the form of sulfur oxide, nitrogen oxide, and carbon dioxide as well as particulates and trace heavy metals. Software optimizations in these cases attempt to burn the fuel cleaner or affect combustion that produces fewer emissions. In fact, most software optimization implementations are justified and originated due to environmental concerns. The last efforts have been in increasing availability and reliability. The categories of preventative and predictive maintenance software optimization systems are included in this case as a means of keeping the plant operational for longer periods with reduced maintenance costs.

Software optimization in the power industry, as well as other industries, began as an outgrowth from computer-based performance monitoring and data archiving. Compared with hardware approaches that required large capital expenditures on equipment and maintenance, software became viewed as a very cost effective means to achieve improved performance with simple maintenance. With the advent of faster computers starting in the 1990's, a more active role for software optimization became possible. Initially, the complexity of the combustion process in terms of chaotic behavior as well as the large number of variables made neural networks a natural choice. In the last few years, however, limitations of neural networks have pushed the development of alternative schemes such as agent-based architectures. The current research in these optimization techniques are presented here.

Various types of artificial intelligence approaches have been surveyed for their application in power generation control and optimization [Viswanathan, 1999] [Oluwande, 2001]. Power plant control systems are dominantly based on the PID

(Proportional Integral Differential) algorithm [Astrom, 1995]. The PID controller is a single-input single-output controller and although quite effective and simple to use, it is limited in its application as most controllables are dependent on multiple variables. The next logical step was multi-variable controllers [Oluwande, 2001]. As the name implies, these built on the PID's weakness by taking multiple inputs to influence a single output or controllable. These were difficult to tune and still did not provide an intelligent solution. Among the first of the advanced algorithms was Model Predictive Control (MPC). Model predictive control, as the name implies, is an algorithm that uses an iterative model of the process being controlled to predict the values for the outputs (controllables) given a set of input variables. In this way, an optimal path of operation can be determined by selecting the inputs that produce the desired outputs based on user-defined criteria. Recent applications have had success; for example, [Havlena, 2002] and [Havlena, 2005]. In both of these, MPC is used to model a coal-fired boiler so that air and fuel control inputs can be selected to minimize nitrogen oxide emissions. Efficiency improvement in the form of reduced heat rate was also obtained through better combustion control. More cases are also given in the case studies later in this chapter. There are still limitations [Hugo, 2000] with MPC, however. MPC is a difficult technology to implement and tune. Most maintenance personnel cannot effectively maintain it in the field. It is not an intelligent solution and is typically implemented with a static model. MPC provides a local optimization solution and therefore is not expandable to enterprise-level optimizations. By its central dependence on a model of the process, MPC is not portable to other processes or even adaptable to configuration changes of the existing process [Hugo, 2000].

In an effort to address the limitations of MPC, intelligent algorithms began appearing in industrial control. Due to the large number of variables involved and the chaotic process of combustion, artificial neural networks seemed a logical choice. Neural networks can learn a model-based training with historical data, thus simplifying the model development process. Neural networks interpolate well and also allow some online retraining to handle process changes. In fact, several vendors produce off-the-shelf packages for neural network optimization [NeuCo, 2008] [Pavilion, 2008] [Pegasus, 2006], also discussed in the following section about commercial products and in the section on case studies. In particular, Booth and Roland [Booth, 1998] summarize the application of neural network software across eleven coal-fired boilers whose goals are reduced nitrogen oxide and efficiency improvements similar to the efforts for MPC above. Neural networks are still difficult for maintenance personnel to modify or tune, as they are a black-box approach. Similar to MPC, neural networks are developed as process specific and are therefore not portable. It is also difficult for neural networks to change goals or handle multiple goals such as those that would arise in an enterprise-level environment.

In recent years, software agents have begun to be applied to control systems. Software agents are a progression from object-oriented programming and attempt a modular, white-box approach to address these limitations in neural networks. Software agents are "an encapsulated computer system that is situated in some environment and can act flexibly and autonomously in that environment to meet its design objects."

[Woodridge, 1997] Software agents enable a problem to be broken into simpler pieces. Since these pieces are autonomous and can react to their environment, they can work together for an optimal solution. For these reasons, they are ideally suited for optimizing process control [Jennings, 2003]. Chang and Lee [Chang-1, 2003] [Chang-2, 2003] developed a multi-agent-based control system for a whole coal-fired boiler that illustrates the use and coordination of agents in feedback control for optimal and stable process control. The use of software agents also strengthens the ability of the optimization to handle enterprise-level solutions since the agents can also interact with outside users just as they do with elements of the combustion process. These enterprise-level applications of agents are discussed further in the section, *Research in Enterprise-level and Business Solutions*, later in this chapter. Further discussion on using software agents in the architecture is discussed in *Chapter III.*

Data mining has played a significant role in enhancing optimization efforts. While data mining itself is not an optimization algorithm, the algorithms of data mining discover many of the relations and other information that make advanced and intelligent optimization systems possible. Ogilvie et al describes using data mining as a precursor to optimization at gas and oil fired power plants [Ogilvie, 1998]. This also details how process data can be mined for such cases. Kusiak et al describes a specific application where data-mining techniques are used to detect events causing mill pluggage in fuel delivery for a coal-fired boiler [Kusiak, 2005]. In Kusiak and Song, a data-mining approach is then applied to the whole coal-fired boiler for optimization in great detail [Kusiak, 2006]. Also included is virtual testing of optimizations that is beneficial to any optimization system. In most cases, online testing is difficult since the power generation process is critical and can often not be risked during the uncertainties of software development. Online testing is also costly so virtual testing becomes an enabling technology in many cases and is needed to persuade management for project approval.

The approach of this dissertation is developed in *Chapter III* and demonstrated in *Chapter IV*. This approach creates an architecture that advances the state of the art with respect to the above. The architecture includes software objects and agents to achieve a modular, decentralized, and autonomous approach that are easy to develop quickly. The architecture also incorporates a coordinating component consisting of a rule-based expert system and neural network classifier. This achieves a centralized solution that is easily controllable by providing a managed interface point for outside users; and maintainable due to the use of rules, which is a natural way of thinking for maintenance personnel. The use of a smaller neural network for state classification only gains the benefits of this algorithm without the costs associated by having neural networks as the sole optimization engine. The architecture is designed to be portable, not only across multiple power generating applications but also in other systems such as vehicular power management and enterprise-level optimizations. In *Chapter III*, this architecture is developed in more detail.

Commercial Software Optimization Products for Power Generating Plants

Software optimization has been applied to various industrial processes for a number of years, but usually in an open loop or advisory mode system. In the 1990's, computer technology and control systems advanced to a point where closed-loop control became feasible. As a result, several vendor products are available with various approaches to optimization. These products typically consist of three main components. The first is preprocessing to check the validity of the input data as well as the health and communication status of the system. The second is the main analysis engine that processes inputs and determines outputs. Finally, a post-processing step is incorporated to check constraints or perform other functions before being sent to the control system as outputs. Many packages also include some data analysis software to view trends and compare data offline. Brief summaries of the most popular products are given below. A general software optimization data flow diagram is also given in figure 2.1 at the end of this section. The commercial platforms discussed here are:

- Pegasus Technologies, NeuSIGHT® Optimization Suite 2001
- Pavilion Technologies, Process Insights® and Process Perfecter®
- Ultramax Corporation, ULTRAMAX® Dynamic Optimization
- NeuCo, ProcessLink® Boiler Optimization Suite

Pegasus Technologies markets the NeuSIGHT Optimization Suite 2001, which is an artificial neural network based system. The hardware platform used to implement the optimization software is Sun Microsystems UNIX based Solaris running on their SPARC processor based systems. Interfaces to the generating unit's distributed control system (DCS) can be via Modbus (serial or Ethernet), OSI's PI Server (Ethernet), and OPC (Ethernet) which is Microsoft's OLE for Process Control. The neural network engine is developed by Computer Associates and can include functional expansions of inputs to produce a better model. The neural network gathers process data and uses this data to partially retrain or retune the model, every two hours. This allows for equipment condition changes (such as wear) or operational changes (such as fuel quality). NeuCo acquired Pegasus Technologies in 2006 and their product offerings have since been combined [Pegasus, 2006]. Discussion of Pegasus' previous product is included here for background information.

The general approach is for the software to calculate desired operating setpoints and bias a set of controllables in the DCS to obtain those setpoints. Process data is gathered typically every 30sec and then averaged over a 10-15min period to provide a statistical smoothing for data entered into the model. The model is designed to provide advisory values when in open-loop mode or directly entered control biases when in closed-loop mode every 10-15min cycle during steady load operation. The software incorporates a graphically user-programmable preprocessing and post-processing area to perform data processing functions on incoming process data or outgoing biases respectively. Constraints can be incorporated into the software to limit the influence over the DCS as well as assist in validity checking of process data. Included in the software suite is NeuWAVE® based on Visual Numeric's PV-WAVE® to provide 2D and 3D graphs and analysis tools for handling process data to aid in model building.

Pavilion Technologies' optimization suite includes: an offline analysis package known as Process Insights®; the main neural network optimization engine Process Perfecter® or Power Perfecter®, which is designed for the power generation industry;

and the RunTime Application Engine® for implementing the model and interfacing with the DCS. The computing platform typically used is Microsoft's Windows NT® on Intel's Pentium® class machines, although UNIX and OpenVMS® platforms have been available. Process Insights, as the name implies, is used to gain insight into the process being optimized. Process data is collected into a database and Process Insights provides statistical and graphical analysis tools to discover relevant variables and variable interaction that would assist in the design of the optimization model. An additional and very powerful feature is the software's ability to incorporate data from various sources in almost any format into a common database with relative ease. The software has the ability to correlate variables and build relations based on time. For example, the software can determine that a change in overfire air damper setpoint affects the nitrogen oxide emissions 45sec later, or that an increase in secondary airflow always precedes an increase in excess oxygen and/or decrease in opacity 1min later. When analysis is complete with Process Insights, enough information should be available to build a model and train it with the process data in the database. In addition to building and training a model, it is possible to overlay expert knowledge of the process to further enhance the capability and accuracy of the model. For example, the model can be built to inhibit decreasing excess oxygen when opacity is high; or create the relation that reducing burner shroud opening is a method to lower combustion temperatures which would result in a thermal nitrogen oxide reduction.

Optimization can be done on single or multiple parameters in a weighted balance allowing the best overall solution or trade-offs to be taken when appropriate. As in other products, user programmable data processing functions are available for validation, constraint, and other purposes. Process Perfecter has two modes of operation being online and offline. Online refers to interfacing the model with the DCS to gather process information as inputs and supply target setpoints as outputs. Process Perfecter is a dynamic model and will not only optimize a unit at steady load but optimize the transition periods as well. This can keep emissions under control while greatly increasing efficiency and stability during the most complex operating condition, being load-change. The offline mode allows the model to be simulated for verification by writing output setpoints to memory and predicting the resulting inputs. The RunTime Application Engine acts as a server for the model and provides an interface with the DCS. It is capable of monitoring and guiding the current optimization scheme.

Ultramax Corporation markets the ULTRAMAX Dynamic Optimization® software for process optimization. The computing platform utilized is typically Microsoft's Windows NT® on Intel's Pentium® class machines. In contrast with offerings by Pavilion Technologies and Pegasus Technologies, ULTRAMAX does not utilize a neural network based engine. Instead, the software employs an empirical modeling and optimization approach that is based on Bayesian statistics and multivariate, weighted-regression algorithms. In comparison with other mathematical methods, ULTRAMAX does not require running experiments, instead learning during the normal process. The software is less susceptible to noise in data and can compensate for disturbances in uncontrolled inputs. Neural networks require large training sets and numerous parametric tests that are not required with ULTRAMAX. The software also is much more capable at extrapolation to new operating states than neural networks, which typically interpolate between known operating states [Ultramax, 2008].

ULTRAMAX has a capacity of 10 control outputs to the DCS and 20 input variables from the DCS. As in other optimization products, single and multivariable optimization goals are possible with user programmable data processing and operating constraints capable of being specified. Included in the software are analysis tools providing: 2D, 3D, and contour graphs; model predictability and interpretation; historical performance and data reports; detected effects of outputs on inputs; and comparison of predicted versus actual inputs. The software can be run in stand-alone mode as an isolated system, linked to a control system to provide suggestion in advisory mode, and closed-loop mode to influence process control [Ultramax, 2008].

NeuCo's ProcessLink is another neural network based optimization product similar in overall architecture to products by Pavilion Technologies and Pegasus Technologies. The software is capable of validating data and retraining itself in real-time during optimization, thus allowing for changing equipment and operating conditions. ProcessLink can operate in both open-loop advisory and closed-loop control modes. NeuCo's Boiler Optimization Suite is actually a family of several products including: CombustionOpt for combustion optimization such as nitrogen oxide or opacity; PerformanceOpt for performance optimization such as heat rate; SCROpt, SNCROpt, FGDOpt, SootBlowingOpt for SCR, SNCR, FGD, and sootblowing systems optimization respectively; FuelOpt, ValueOpt, and ProfitOpt to optimize the goals of fuel, value, and profit respectively. The computing platform is Microsoft's Windows® running on Intel Pentium® class machines. The software enlists the standards of Active-X®, Visual C++®, Microsoft Office®, Visual BASIC®, and Open Database Connectivity® (ODBC) allowing for simple integration and future growth [NeuCo, 2008].

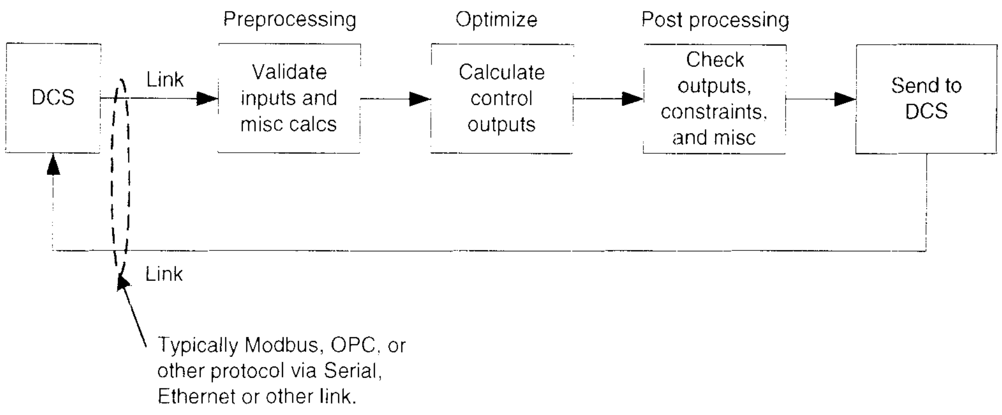

Preprocessing
Optimize
Post processing
DCS
Link
Validate inputs and misc calcs
Calculate control outputs
Check outputs, constraints, and misc
Send to DCS
Link
Typically Modbus, OPC, or other protocol via Serial, Ethernet or other link.

Figure 2.1. Data flow in optimization software products.

Case Studies at Various Coal-fired Generating Stations

Software optimization has been applied for several years to various industries and the quantity of research is extensive. Offline data analysis techniques such as computational fluid dynamic modeling have been used as well as advisory mode neural network based systems to suggest the best mode of unit operation. It is only recently with advances in computing power have process industries begun to utilize optimization schemes in their online control system.

In an optimization at Illinois Power [McVay, 1998], the Ultramax optimization software is discussed for the purposes of nitrogen oxide reduction and efficiency improvements at the Baldwin Generating Station and others. As is the case with most plants performing such optimizations, the goal was to provide a low-cost solution for reducing nitrogen oxide as part of the company's Phase II Clean Air Act Amendments compliance plan without adversely affecting operation of the generating unit.

The decision was made to proceed with optimization at Baldwin based on the success at Hennepin, another Illinois Power generating station. In both cases, the distributed control system utilized at the plant was the Westinghouse WDPF II with data archiving provided by OSI's PI Server system. Hennepin unit 2 was able to achieve improvements of 3% in operating efficiency while reducing nitrogen oxide by 20% at full load. The solution was known to work with the existing control system and had acceptance by the operating staff. Hennepin unit 2 has a tangentially fired twin-furnace boiler rated at 235MW. The greatest effects came from lowering excess oxygen and tightening upper wind box dampers [McVay, 1998].

Baldwin Units 1 and 2 are 575MW B&W cyclone boilers and unit 3 is an ABB-CE tangentially fired 595MW boiler. The Ultramax system was interfaced to the PI Server at this site to obtain process information and communicate recommended settings to the operator. The operator then implements these settings upon inspection thus performing optimization in an open-loop advisory mode. Closed-loop control is also an option of the software. Early results at Baldwin have shown positive results in efficiency and nitrogen oxide reductions. The use of the optimization system has also proved to provide a more consistent operation from shift to shift as the advisory data is utilized [McVay, 1998].

In a research paper [Radl], the use of artificial intelligence software systems is discussed for generating units. The software system primarily addressed is NeuSIGHT® by Pegasus Technologies and its application to Ameren's Labadie Station, Ontario Power's Lambton Station, and Houston Power and Light's Parish Station. Discussion of implementation and process data flow is given after these three station studies.

The Labadie Station boiler is a 600MW tangentially fired unit with PRB coal as the primary fuel. Prior to software optimization, the unit was fitted with ABB-CE's Low nitrogen oxide Concentric Firing System or LNCFS Level 3 nitrogen oxide control technology including two levels of closed-coupled overfire air and five levels of separated overfire air. The software optimization is interfaced directly to the distributed control system to allow both advisory mode and closed-loop mode for automatically introducing biases. Labadie has been able to achieve a 30% reduction in nitrogen oxide beyond the existing reduction obtained by the LNCFS Level 3 hardware and switch to PRB coal. Heat rate is calculated in real-time by the NeuSIGHT software and work is continuing to evaluate the impact on heat rate and furnace gas exit temperature. The optimization influences 24 controllables continuously over the 1/3 to full load range, including overfire damper settings, excess oxygen, wind box to furnace differential pressure, and mill feeder speeds [Radl].

Lambton Station units 3 and 4 were selected as a trial of the NeuSIGHT optimization software as part of the company's strategy to reduce heat rate by 2% and nitrogen oxide by 10% from the 1996 levels by the year 2000. Details of this project are presented in the research paper [Henrikson]. Units 3 and 4 are tangentially fired 510MW boilers controlled by a Bailey INFI-90 distributed control system. Nitrogen oxide reductions of 10% to 25% were obtained with a 0.5% improvement in heat rate [Radl].

Parish Station unit 8 is a base-loaded tangentially fired 600MW CE boiler with PRB coal as the primary fuel. Unit 8 did not have a distributed control system at the time of optimization and most process data was collected by a Honeywell data acquisition system. Originally, the project was not scoped to provide closed-loop control due to this limitation. However, this capability was realized with the addition of an Allen-Bradley PLC. The PLC was able to collect remaining data that was not in the Honeywell system such as excess oxygen, overfire air setpoints, etc. Optimized setpoints from the NeuSIGHT system were sent to the existing boiler controls via this PLC. Nitrogen oxide reductions of 15% were obtained with the system and an additional constraint on CO emission below 50ppm was also met. Work is progressing to fit the NeuSIGHT system to the other Parish units including a proposal to improve furnace cleanliness with soot blower and water lance optimization [Radl].

In an optimization at Ontario Hydro's Lambton Generating Station [Henrikson], software optimization at units 3 and 4 are first discussed and then optimization at units 1 and 2 are discussed in additional detail. The goal of optimization for all units was both a reduction in nitrogen oxide and an improvement in heat rate.

Lambton units 3 and 4 are 510MW tangentially fired 510MW CE boilers with 48 burners. Each unit has 6 horizontal ball mills with two primary air fans, two forced draft fans, two induced draft fans, and a precipitator. The distributed control system is a Bailey INFI-90 with NeuSIGHT by Pegasus Technologies serving as the optimization system. A total of 162 and 175 process variables are used as inputs to the NeuSIGHT model which biases 26 and 38 outputs as controllables for units 3 and 4 respectively. The main controllables for unit 3 are: 7 levels of auxiliary air dampers; excess oxygen; mill outlet temperatures; mill feeder speeds; and primary air dampers for 6 mills. Since unit 4 is fitted with low nitrogen oxide burners and separated overfire air ports (SOFA), the SOFA dampers and burner tilts are also included as controllable parameters. Unit 3 has shown a 15% to 25% reduction in nitrogen oxide with a 0.5% improvement in heat rate. Since unit 4 was fitted with low nitrogen oxide burners and SOFA, the baseline nitrogen oxide level was 60% of that for unit 3. Still, a 10% to 15% reduction in nitrogen oxide was obtainable for unit 4 [Henrikson].

Given the success of Units 3 and 4 of the Lambton station, optimization of units 1 and 2 were begun. During the optimization process of units 3 and 4, plant personnel gained sufficient experience with the NeuSIGHT software to perform the optimization in house. The first step was to upgrade the existing control systems of units 1 and 2 to the Bailey INFI-90 similar to units 3 and 4. A more thorough optimization plan was to be implemented for units 1 and 2 including advanced control schemes for various systems in addition to the NeuSIGHT optimization. The control schemes were: [Henrikson]

- Pulverizer Optimization – This is both reactive and proactive to changing plant conditions. Reactive optimization will allow the system to alter operating parameters based on fuel changes, equipment wear and drifting sensor. The proactive approach will incorporate a new technique called Visual Episoidal Associative Memory (VEAM) along with typical pattern recognition and clustering methods to monitor automatic settings in the software model to obtain more knowledge from the model's response to changing conditions. This would allow real-time and on-line condition monitoring and prediction to provide cost effective maintenance.
- Sootblowing Optimization – Optimal cleaning of the boiler is required to maintain efficiency and provide good control of steam and tube temperatures and exit gas temperature. Proper use of soot blowers can prevent excessive tube wear and reduce unplanned outages. Software optimization employs algorithms to detect the buildup of soot on heat transfer surfaces and to blow soot as needed while avoiding excessive blowing of regions. Individual soot blowers can be actuated for cleaning or utilized to reduce tube metal temperatures.
- Advanced Calibration Monitoring – Like most modern distributed control systems, the INFI-90 at Lambton has about 10,000 data points per unit. Of which, 60% are digitals, 10% are calculated or analog outputs, and 30% are analog inputs from sensors. The 30% or 3000 analog inputs from sensors include thermocouples and RTDs, oxygen, nitrogen oxide, pressures, flows, levels, etc which drift over time and require recalibration or some other maintenance. Periodic maintenance of these can be labor intensive and expensive. Advanced Calibration Monitoring

(ACM) is intended to monitor these sensors over time and detect when they require maintenance by comparing their readings with other data values. Errors that would be too small to detect by individual preliminary inspection are quickly detected with a neural network model and flagged in an automated fashion for easy maintenance. This can lower O&M costs by calibrating sensors only when they need it while helping efficiency by controlling with accurate data. For example, every +1F error in main steam temperature contributes a fuel cost increase of $75,000 per year. This will also improve optimization performance by ensuring that the data is of the best quality it can be.

- Feed water Heater Level Optimization – Feed water heaters use extraction steam to heat feed water, improving the unit's thermal efficiency. Levels too high can flood tubes, causing inefficiency, and levels too low can uncover the drain nozzle and cause vibration and premature damage. The optimum level changes with load and typical level controls are inadequate to maintain this level. As a result, heaters can fail in as few as 7 years (when life expectancy should be greater than 20 years) and peak efficiency is not obtained. Optimization is to control the levels with the distributed control system using a load-based setpoint derived from the differential of the inlet and drain outlet temperatures. This is referred to as the Drain Cooler Approach (DCA) and the level/DCA test is performed automatically by a patented software system known as Mdc2000.
- Turbine "Free Pressure" Mode Control – "Free Pressure Mode" is a term Bailey uses to describe what has also been called Valve Point Control, Floating or Sliding Pressure, Multiple Hybrid Variable Pressure, etc. The concept is that

operating at valve point increases turbine efficiency and therefore the turbine valves should be at valve point for a given load and throttle pressure allowed to vary within determined limits. Valve point is defined with sequential turbine valves as when the current valve is 100% open and the next valve is just about to open. Typically this has been difficult to control and reduced the responsiveness of the unit to load changes and, though reducing turbine wear, may increase boiler wear. Free Pressure Mode solves these problems by allowing the valves to participate in load changes and then return to valve point at stable load. The limits on varying throttle pressure and this participation provide tuning to alleviate these problems.

Results of these optimizations were not available at the time of this publication but are expected to produce excellent emissions and heat rate reductions along with valuable insight in unit operation [Henrikson].

Sample Cost Comparisons for Hardware vs. Software Nitrogen Oxide Reduction Efforts

As a final justification of software optimization techniques versus hardware techniques in power generating plants, the case of a nitrogen oxide reduction effort is examined. The reduction of nitrogen oxide, various oxides of nitrogen, is a key pollution parameter and greenhouse gas of current notoriety. There exist several technologies for the reduction of nitrogen oxide emissions in coal-fired plants. Systems such as Selective Catalytic and Non-Catalytic Reduction, or SCR and SNCR respectively, attempt to chemically alter the post-combustion flue gas such that nitrogen oxide is reduced to nitrogen and water. These systems have excellent nitrogen oxide reduction capabilities but are very expensive to install and operate. Rotating over-fire air systems and low nitrogen oxide burners attempt to improve the combustion process to reduce nitrogen oxide formation and are almost as effective as SNCRs and SCRs. Rotating Overfire Air systems have the highest costs but are relatively cheap to operate. Low nitrogen oxide burners have both reduced installation and operating costs. Software optimization also seeks to reduce the formation of nitrogen oxide but does so via dynamic tuning of the combustion controls. Software optimization often has widely varying reductions depending on the characteristics of the particular boiler and is usually the least effective in quantity reduction. However, the greatly reduced costs of installation and operation are proving to give software optimization the best cost-to-performance ratio in the industry.

A cost and benefit comparison of these systems is given in figures 2.2, 2.3, 2.4, and 2.5. The first two figures compare installation and yearly operating costs for a typical configuration. The third chart is a relative comparison of nitrogen oxide reduction by solution. The fourth chart attempts to compare a cost / reduction benefit by taking installation costs plus a 10year operating cost estimate divided by the expected nitrogen oxide reduction. Therefore, lower numbers indicate a comparative quantity of nitrogen oxide emissions was reduced for lower costs. Several factors should be considered with these charts, as installation costs will vary depending on the plant configuration. The effectiveness or appropriateness of certain solutions may also be dictated by plant configuration. For example, SCRs / SNCRs / Rotating Overfire Air may not be implementable if space does not permit. Rotating Overfire Air typically performs better for wall-fired units versus tangentially fired units. The values for these figures come from the analysis of a generating unit at the Duke Energy Gallagher Generating Station in New Albany, Indiana. This is a coal-wall-fired 18 burner steam-generating boiler with a gross generating capacity of approximately 150MW.

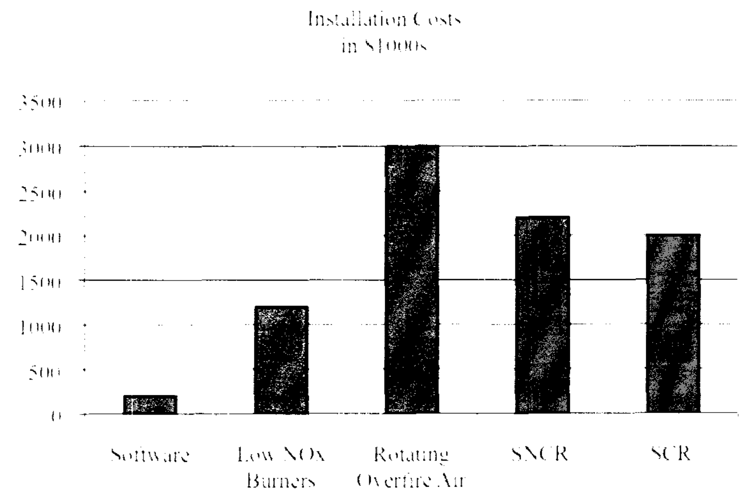


Figure 2.2. Initial installation costs of compared technologies.

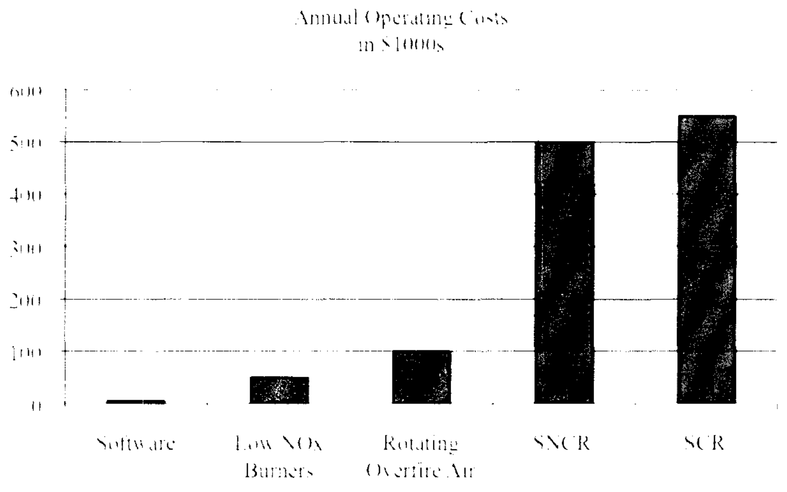


Figure 2.3. Annual operating costs of compared technologies.

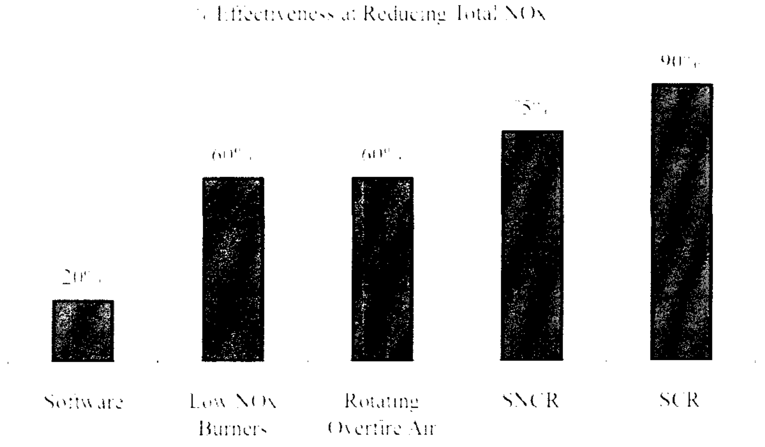


Figure 2.4. Effectiveness of compared technologies at reducing nitrogen oxide.

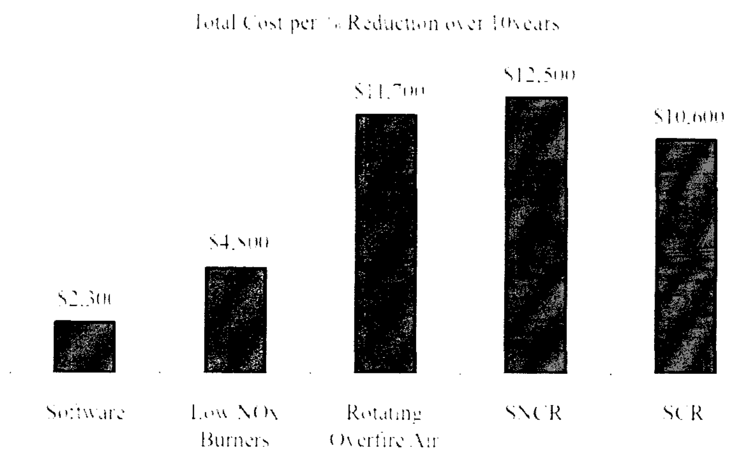


Figure 2.5. Relative costs for the equivalent nitrogen oxide reduction.

Research in Hydro Power Generating Plants in Particular

Due to the low negative environmental impact and relatively free fuel in the form of water flow available to hydro-generating units, hydropower remains one of the most practical forms of green power production. Incremental increases in hydropower production directly offset carbon dioxide, nitrogen oxides, and other emissions typical of fossil-based generation in addition to the monetary returns of increased power production.

There is some work on hydro research but because hydro-generating units are already environmentally friendly, they do not always get the level of research and optimization afforded to fossil fuel generating units. Several approaches have utilized a combination of artificial neural networks and fuzzy logic either replacing or enhancing legacy PID control. Zhang and Yuan [Zhang-1, 2006] achieve good performance by replacing conventional control with a fuzzy neural network controller on a single unit.

They claim hydro-generating units are non-linear and high-order systems that cannot be controlled optimally with classical control. They use a rule-based fuzzy neural network for unit characterization and a fuzzy neural network controller for affecting control. Djukanovic et al [Djukanovic, 1997] also utilize a neuro-fuzzy controller with self-learning capabilities to handle hydro-generator transients. They use a back-propagation-type gradient descent method, *temporal back propagation*, to propagate the error signal through different time stages. Precup et al [Precup, 2005] developed a Takagi-Sugeno based fuzzy controller dedicated to turbine speed control. They provide a thorough mathematical analysis incorporating their controller in the PID algorithm. Zhang and Zhang [Zhang-2, 2006] place an adaptive fuzzy controller between the existing PID control and the turbine governor for static and dynamic improvements to the governing system. Ramond et al [Ramond, 2001] examine direct adaptive predictive control and its application to improve the performance of existing PID control for a hydro plant. While these approaches have produced good results, the use of fuzzy control and predictive models are not as modular and simple as a multiple software agent structure and do not scale well when applying to multiple units and multiple plants [Huang, 2001].

Software agents are a new technology being explored in hydro generation. In contrast to the neuro-fuzzy and other approaches above, Huang explores using an ant colony system implemented by multiple software agents to determine optimal dispatching of hydro-generating units, although this work groups multiple units at a single plant together [Huang, 2001]. The author's paper [Foreman, 2008], develops software agents that optimize individual hydro-generating units. These agents are rule-based based and locally influence the turbine blade angle and wicket gate positions, also defined in [Paul, 1996], controlled by the existing control system. These agents incorporate a rule-based expert system and can autonomously negotiate with each other in order to achieve the additional benefits of total plant optimization. This also enables the system to scale well to other plants and provides a mechanism for outside business entities to influence the control as well, thus achieving an enterprise-level solution.

A commercial application, WaterView®, is discussed by March and Wolff [March, 2003] as applied to the Tennessee Valley Authority's fleet of hydro plants. This is described as an "optimization-based hydro performance indicator" and explores individual unit optimization as well as coordination with the hydro fleet.

Research in Enterprise-level and Business Solutions

Recently, power-generating companies have strived to be more competitive and as information technology continues to advance, enterprise-level solutions have grown in demand. Kulhavy et al [Kulhavy, 2001] discusses three types of enterprise optimizing technologies. The First is model predictive control (MPC), as discussed above. MPC already has much history in industrial control systems so it is natural to research this option. There are still limitations [Hugo, 2000] as MPC is best suited for local optimizations and does not handle changing goals and multiple users well. The next technology explored is data-centric forecasting and optimization, which is similar to data mining as mentioned above. Since power generating plant data has high dimensionality and multiple plants result in a large quantity of data, the data-centric approach focuses on the asset that is most plentiful, being the process data. This approach has yielded several interesting relations and with sufficient history, performs well in market forecasting. The data-centric approach also scales and interfaces well with corporate databases that are a more natural way for business entities to deal with information rather than scientific process relations. However, this approach is passive and lacks intelligence and autonomy. The final technology is based on software agents and this seems to dominate successful research.

Software agents are well suited to control optimization and, as the name implies, also provides an agency relationship between business entity users and the processes being optimized. The Electric Power Research Institute (EPRI) has developed a tool, SEPIA (Simulator for Electric Power Industry Agents), that simulates the integration of the power generating process with corporate business entities [Wildberger, 1999]. Another survey paper [Amin, 2002] focuses specifically on agent-based systems and how the evolution of such enterprise-level solutions is necessary in our global market for competitiveness. SEPIA is also discussed in more detail in this paper as well as application of agent technology in general. AspenTech is one company that has defined a strategic model of applying such enterprise-level optimizations across diverse business entities (operations, transmission, marketing, power trading, management, etc) and for the multiple goals (emissions, efficiency, reliability, profit, etc) [Aspen, 2002]. Specific applications include the JAVA-based MASPOWER [Vishwanathan, 2001], which is designed to provide an infrastructure for a multi-agent system that elicits coordinated and negotiated decisions from the decision makers of the enterprise. This system builds a negotiation framework for the power systems environment. Tolbert et al [Tolbert, 2001] developed a scalable multi-agent system for real-time management of multiple generating plants. This system attempts to manage power delivery from various generating assets for maximum efficiency while incorporating the ability to stabilize the power delivery grid during transient conditions for enhanced reliability in power delivery.

The architecture in this dissertation includes software device objects and software agents as necessary for handling individual components of the power system. These objects and agents are coordinated by an expert system that also provides influence with the business enterprise. Therefore, the architecture builds on the above efforts to both manage the finest details of individual components all the way up to the various business entities in the enterprise-level solution. Details of how the enterprise-level solution is handled in the architecture is discussed in *Chapter III*, section *Business Entities and the Enterprise-level Solution*.

Research in Vehicular Systems for Power Management

Vehicular systems are small and mobile. They incorporate the power generation and load components together in one power system. While there is an optimizing element, the efforts in vehicular systems are typically referred to as power management, since the multiple components of the power system are all available to be managed. The optimization and management of vehicular power systems are becoming more important and their application increasingly demanding and complex [Vahidi, 2007]. In automotive systems, operating range, cost, and longevity are key factors needed to gain adoption as viable consumer products. Minimization of fossil fuel use is also a defining reason for such automotive systems and with the many vehicles in use today, even marginal improvements produce large results. In spacecraft and other specialized systems, size, mass, and available power have always been limiting factors that correlate directly with cost and feasibility. Software management provides a theoretically zero footprint technology that can aid in reducing the size and mass expenditures while improving the availability of power.

Power systems management software began with the classical programming approach whereby power system devices were inter-connected on a power bus and then "managed" by simple logic, either enabling or disabling select devices. However, the increasing demands and complexity of such power systems has quickly ruled out the classical approach and an intelligent scheme has become necessary to realize true management. Lin et al [Lin, 2003] explores a dynamic programming approach in the application of a hybrid truck. The truck has two power sources for propulsion, a diesel engine and an electric motor. The power management system uses the dynamic programming approach to determine the power needs of the truck and how to split this need between two sources.

In Vahidi et al [Vahidi, 2006], a centralized approach for model predictive control is explored in a mild fuel cell hybrid vehicle that incorporates an ultra-capacitor [Schindall, 2007] for handling transients. The benefits of centralized control in general are also summarized. In Vahidi and Greenwell [Vahidi, 2007], the alternative approach of decentralized model predictive control is explored for a similar vehicle. These papers compare and contrast the benefits between the centralized and decentralized approaches. Centralized approaches manage the whole power system as one entity, thus automatically achieving a system-wide optimal solution. However, this approach is very, if not prohibitively, complex and changes to any part of the power system require updating the whole optimization [Vahidi, 2007]. The decentralized approach is simple, modular, and transportable [Vahidi, 2007]. When coordinated, it can still provide solutions near the centralized approach in performance [Camponogara, 2002]. In Vahidi et al [Vahidi, 2006], work is done to model power devices yet the coordination of these devices is alluded to and left for a future paper. Also in Bauman and Kazerani [Bauman, 2007], detailed mathematical models of power devices were explored from a hardware comparison aspect. Their power management system was still of a centralized approach but they demonstrated that device management is a key layer to power systems management.

Software agents have also been used in vehicular power management. For example, Luk and Rosario [Luk, 2005] explore a negotiation-based multiple agent system for power management in electric vehicles. In this work, the agents act intelligently and autonomously on behalf of the vehicle's various load devices to negotiate for power. However, this work does not involve the power generating devices to realize a total power management system. In *Chapter III*, developing a layered power management system whereby software objects perform local management functions while a higher layer performs intelligent coordination of these enhances this concept. The architecture defined in this chapter achieves the modularity, simplicity, and portability characteristic of decentralized approaches while obtaining the benefits of centralized approaches through the use of a coordinating layer. The specific problem of suboptimal performance resulting from a lack of a classical model or *a priori* knowledge [Schupback, 2003] is addressed by having the proposed architecture handle both the current and next operating states together. This is only possible with the coordinating layer since some other centralized entity would be required to determine future operating states.

Figure 2.6 illustrates how the best characteristics are included in the architecture. The architecture incorporates a rule-based expert system for the autonomous decision process with a small neural network to get some pre-classification benefits from this approach. These are then interfaced with a software device object or agent that achieves the benefits of model predictive control at the lowest level.

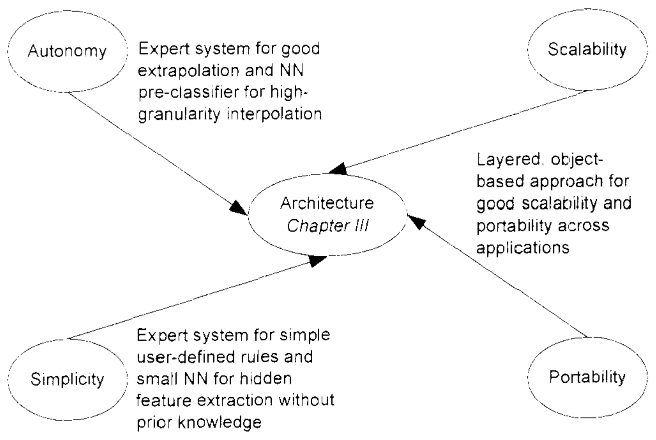

Autonomy
Expert system for good extrapolation and NN pre-classifier for high-granularity interpolation
Scalability
Layered, object-based approach for good scalability and portability across applications
Architecture
Chapter III
Expert system for simple user-defined rules and small NN for hidden feature extraction without prior knowledge
Simplicity
Portability

Figure 2.6. Selection of best characteristics.

Autonomy, simplicity, portability, and scalability influence the definition of the architecture so that the best of these can be formed into a solution without the penalties of the worst of these. This is discussed in more detail in *Chapter III.*

CHAPTER III

ARCHITECTURE

The architecture defines a framework for realizing an intelligent power management system in software. The architecture is designed to reside within the existing control system of the process to provide a zero-footprint solution. The architecture has the general structure of being modular software objects or agents, which are designed to handle individual components and devices of the process while being coordinated with a rule-based expert system to achieve a whole system optimization. The application of this architecture is for power systems management and includes the goals of: reducing power consumption; increasing power generation; increasing power storage efficiency; and reducing environmental impacts. The user interacts with the architecture similar to a model-view-controller approach. Depending on the user type, e.g. manager, engineer, or an intelligent software application, different user interfaces are utilized. These interfaces may include a custom database, SCADA type (Supervisory Control And Data Acquisition), or may be transparent, i.e. the user may interact directly with the whole system while the architecture autonomously handles management and optimization functions, e.g. automotive applications. The use of SCADA and similar interfaces also affords the ability to implement security into the architecture. Although security is not the focus of this dissertation, there is some discussion towards the end of this chapter in the section *Business Entities and the Enterprise-level layer.*

Evolution of the Architecture

Power management and optimization have been an integral part of power systems and the processes supported by these for some time. Starting in the 1990's, computers became powerful enough to start performing real-time management and optimization functions. In the beginning, this was limited to simple data collection and reporting and in some cases, the results of these reports would be sent back to the process control system to take some action based on the output. However, this has now evolved into intelligent approaches utilizing more advanced tools such as pattern classification, data mining, and sophisticated software structures. While the benefits of these advancements are obvious, as discussed in *Chapter II*, the varying approaches have complicated the process of building new implementations. In many cases, the architecture is redefined each time. This dissertation seeks to define a scalable and portable architecture that can be utilized across varying system devices, processes, and missions. This will minimize duplication in the design process and simplify implementation allowing a quick and standardized solution to be obtained.

In the power generation industry, there is much demand for management and optimization of power. In addition to the obvious benefit of increased power production, there are significant gains to be obtained in optimizing the process to reduce emissions and improve reliability of power delivery. There are environmental factors, especially when dealing with fossil-fuel combustion processes, which are of increasing importance. There have been attempts to standardize approaches for achieving these goals, such as EPRI [Stallings-1, 1998] [Stallings-2, 1998], but such approaches are both difficult to implement and very specific in application and therefore not portable or adaptable. Beyond these considerations, effective power management is beginning to mature and there is a desire for an enterprise-level solution to management. Such a solution allows other entities in the business enterprise to become an intimate part of the power generation process so that the whole company can make better strategic decisions. Therefore, scalability is becoming a key feature so the application can grow with the business' needs. Details on how differing business entities utilize the architecture are discussed further in the section, *Business Entities and the Enterprise-level Solution*, later in this chapter.

In other areas, power management and optimization have become key components of vehicular systems in recent years. Vehicular systems include hybrid automobiles but also more exotic applications such as spacecraft and remotely operated vehicles or ROVers. The mobility of these vehicles requires them to carry their power systems with them and occasionally be without any power generating resources. Size and cost become factors in consumer vehicles and environmental benefits can be achieved where power management reduces consumption of fossil fuels. This often results in power management and optimization becoming a mission-enabling technology for such vehicles.

Detailed considerations of these application areas are given at the beginning of *Chapter IV* where the implementation is discussed. The architecture is developed here, in *Chapter III*, independent of implementation environment.

Quantifying the Criteria of the Architecture

In order to effectively compare different approaches and determine the benefits of this architecture, quantitative metrics need to be derived for our criteria of portability, scalability, simplicity, and autonomy. In this section, methods for quantifying these criteria are defined. These are quantified in the implementation cases of *Chapter IV*.

When computing metrics, the power management software is broken down into its fundamental modules. A fundamental module is the smallest component of the application that can be considered independently of the other modules, i.e. it contains its dependencies, at least with respect to quantifying the metrics. For example, an artificial neural network cannot be further divided without destroying its functionality due to the interdependency of the neurons and thus becomes one module. Sequential logic can be divided into functional groups, such as battery control, solar cell control, etc. These groups would be code modules that pertain to a common controllable parameter. A rule-based expert system can be subdivided into interdependent rule sets based on their inputs and outputs. Below, a sample set of rules is segregated into fundamental modules.

```
Rule 1:     IF f(a,b,c) THEN g(x)  ; independent rule
Rule 2:     IF f(x) THEN g(y)  ; depends on output of rule 1
Rule 3:     IF f(a) THEN g(z)  ; independent of rule
```

Therefore...

```
Module 1:
        Rule 1
        Rule 2
Module 2:
        Rule 3
```

This modular break down results in a more granular metric calculation.

Quantifying Portability

Portability is defined in this dissertation to provide a measurement of the development effort required for a given architecture to move from one application to another in order to compare architecture. This is a measure of how easily the architecture can be moved horizontally, i.e. moving the application from one power system to another power system but with similar functional scope. This is in contrast to scalability (discussed later), which seeks to give a measurement of the development effort required to add new scope and functionality to an existing application. Functional scope in this case would refer to the intended goals, or responsibility, of the power management system, which would not change in a portable, horizontal case. This is normalized onto a scale of 0% to 100%. A portability metric of 80% would imply that 20% of the effort to initially build the application would have to be duplicated when porting to the new application, i.e. 80% of the application is portable.

Once the application is broken down into its fundamental modules, the modules that can be ported to the new application without modification contribute towards the portability metric. The portability of individual modules would be either 1 if portable or 0 if not portable without modification, or a fraction thereof. Since these modules should be divided into as small a functional unit as practical, i.e. fundamental modules, any modification will involve reviewing the whole module and thus the choice of 0 as the metric in this case. Equation 3.1 quantifies this metric.

$$P = \frac{\sum_{i}^{N} w_i p_i}{\sum_{i}^{N} w_i} \tag{3.1}$$

Where P is the portability metric, N is the number of fundamental modules, w is a weighting factor representing the effort for the specific module (since modules may have unequal different development efforts), and p is the portability (0...1) of the specific module i. The rule sets below illustrate this along with figure 3.1 for a sample application composed of 10 rules.

```
Module 1: portable p=1
       IF f(a) THEN g(x)    ; f and g do not change when porting
                            ; one rule or 10% of application
Module 2: not portable without modification p=0
       IF f(b) THEN g(y)
       IF h(b) THEN i(z)
                            ; two rules or 20% of application
Module 2':
       IF f'(b) THEN g(y)   ; needed to change conditional
       IF h(b) THEN i'(z)   ; needed to change action
```

And so on...

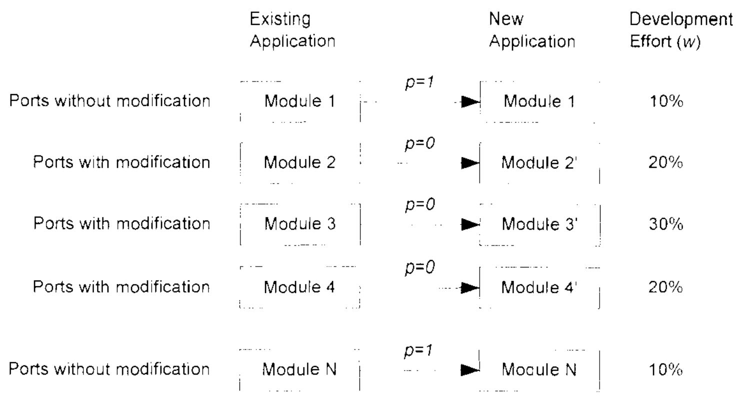


Figure 3.1. The portability metric.

In figure 3.1, the portability metric of the five modules is 1*10% + 0*20% + 0*30% + 0*20% + 1*10%, or 20%. Therefore, 20% of the initial effort would be retainable and 80% would require modification or rework.

Quantifying Scalability

Scalability is defined in this dissertation to afford comparison between architectural approaches to power management systems when enhancing the scope of the application. Scalability has some similarity with portability in that it also quantifies the architecture's ability to handle changes in the application. In contrast to portability, however, scalability predicts the effort required for the architecture to enhance the scope of the application. This enhanced scope would represent additional goals, mission environments, interaction with new users, or higher level coordination with other systems. This is normalized onto a scale of 0% to 100%. A scalability metric of 80% would imply that 20% of initial application would have to be modified to scale to the new scope requirement, i.e. 80% of the architecture is scalable. When evaluating the scalability of the architecture, it is broken down into its fundamental modules. Equation (3.2) quantifies the computation of the scalability metric and figure 3.2 demonstrates this.

$$S = \frac{\sum_{i}^{N} w_i s_i}{3\sum_{i}^{N} w_i} \tag{3.2}$$

Where S is the scalability metric, N is the number of fundamental modules, w is a weighting factor representing the effort for the specific module (since modules may have unequal different development efforts), and s is the scalability factor of the specific module i. Table 3.1 determines this scalability factor and an example of scaling rules is presented following the table.

Scalability factor s_i	**Degree of change**
3	No change
2	Parameter-level changes
1	Code-level changes
0	Not scalable

Table 3.1. Scalability factors.

For example, a rule that performs some action based on the current number of users might look like this.

```
b = 3 ; number of users
IF b = 1 or b = 2 THEN ; take action based on number of users
        g(x)
ELSE IF b = 3 THEN
        g(y)
ENDIF
```

If a new user is integrated into the application, *b* can be simply changed from 3 to 4 and thus this is a parameter-level change. If the new user is of a different type then a new coordinating rule is required to handle this type, which becomes a code-level change.

```
b = 3 ; number of users
IF b = 1 or b = 2 THEN ; take action based on number of users
       g(x)
ELSE IF b = 3 THEN
       g(y)
ENDIF
IF b = 4 THEN ; do special case to handle this new user
       h(x)
ENDIF
```

When the module cannot handle a new user without a complete redesign, this becomes a non-scalable module.

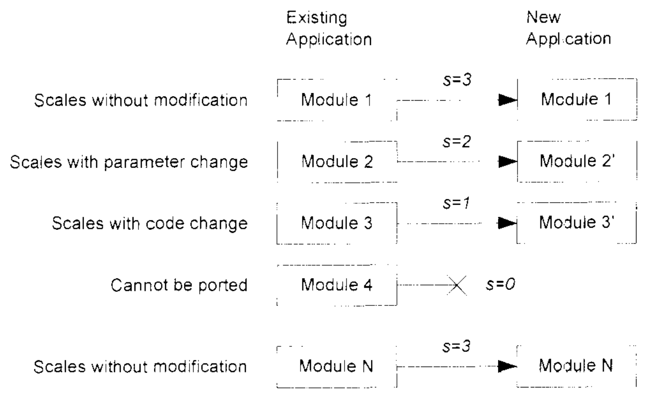


Figure 3.2. The scalability metric.

In figure 3.2, the scalability factor would be (3 + 2 + 1 + 0 + 3) / 15 * 100%, or 60%. Therefore, 60% of the initial development effort would be retainable and 40% modification would be required to incorporate the new scope. In this example, all the development effort weights were considered equal.

When the comparison of scalability needs to include multiple enhancements that are different in type, e.g. handling a new user and handling a new goal, it may be beneficial to compute the scalability of each enhancement separately and then average the individual scalability metrics to achieve a scalability metric for the total application enhancement. This is the case for the coal-fired boiler implementation in *Chapter IV*. Additional applications of this metric are also in *Chapter IV*.

Quantifying Simplicity

In this dissertation, simplicity implies the characteristics of being easily understood and maintainable from a maintainer's perspective while also being of minimal structural complexity from a software design perspective. Simplicity is difficult to measure directly so it is inferred by minimizing difficulty within the architecture. This is accomplished with a unit-less measure for relative comparisons among applications. We determine the difficulty by defining a metric that quantifies the characteristics above. Being easily understood and maintainable is synonymous with having easily readable and interpretable code. Structural complexity is well defined in software science. The difficulty metric is defined in the following equations for a fundamental module. The difficulty for a whole application would be the sum of the module complexities.

$$D_M = C_R C_S \qquad (3.3)$$

Where D_M is the difficulty of a fundamental module and C_R is the readability complexity defined in table 3.2 that quantifies difficulty in interpretation. C_S is the structure complexity defined by (3.4) from Henry and Selig's work [Henry, 1990] based on the information-flow metric of Henry and Kafura's work [Henry, 1981].

Readability C_R	**Difficulty**
1	Natural language (simplest, straightforward reading of meaning)
2	Computable equation (requires computing equations to determine meaning)
3	Procedural computation (requires following a difficult procedure to determine meaning)

Table 3.2. Readability complexity.

$$C_S = C_C(fanin \times fanout)^2 \qquad (3.4)$$

Where C_C is McCabe's cyclomatic complexity defined by the number of decision points plus one, *fanin* is the number of inputs to the module and *fanout* is the number of outputs from the module. The power of two used in this weighting is the same as Brooks' law of programmer interaction [Brooks, 1975] and Belady's formula for system partitioning [Belady, 1979]. Thus, an established method of measuring complexity is modified to include human readability as a characteristic. This is demonstrated in the pseudo code here and in the implementations of *Chapter IV*.

```
Module 1:
        IF a = TRUE THEN x = TRUE
                ; readability $C_R$ = 1
                ; $C_C$ = 1 conditional + 1 = 2
                ; fanin = 1, fanout = 1
                ; $D_C$ = 1 * 2 * (1 * 1)$^2$ = 2
Module 2:
        IF (b2 + 2b - 2sin(c) > 0) THEN y = TRUE
                ; readability $C_R$ = 2
                ; $C_C$ = 1 conditional + 1 = 2
                ; fanin = 2, fanout = 1
```

```
          ; D2 = 2 * 2 * (2 * 1)^2 = 16
Module 3:
     NeuralNetwork(input=(a,b,c,d), output=(w,z) {
          Foreach layer {
               Foreach neuron {
                    Foreach input { f(input) }
               }
          }
     }
          ; readability C3 = 3
          ; C3 = 3 conditional + 1 = 4
          ; fanin = 4, fanout = 2
          ; D3 = 3 * 4 * (4 * 2)^2 = 768
```

The sum of D_1, D_2, and D_3 is 786 in the above example and this becomes the difficulty metric for the application.

Once difficulty is calculated, the comparative simplicity can be inferred from a lower ratio of the difficulty metrics between compared applications. If application A has a difficulty metric of 2000 and application B has a difficulty metric of 3000, then the ratio A:B, or 2000/3000, indicates that application A is 67% of the difficulty of application B. A difficulty of 1, this would imply readability of 1, no conditional statements, and a *fanin* and *fanout* of 1; for example the statement `a = 4`.

Quantifying Autonomy

Autonomy is a measure of the architecture's ability to make decisions and perform the mission at hand with minimal human intervention. Achieving autonomy frees the operator from control tasks, handles trouble conditions automatically, allows strategic decisions to be automated, and finally enables cooperation with peer systems within the environment for a coordinated solution. Unlike the previous metrics, autonomy is measured on the whole application as opposed to fundamental modules.

There has been some work on quantifying autonomy [Clough, 2002], and meaningful application of this metric depends largely on the mission being evaluated. In power management systems, the key parameters chosen in this dissertation are:

- Operator independence – requiring minimal user interaction, having automation.
- Self-preservation – the ability to handle trouble conditions (alarms) automatically, recover and continue the mission, and fail in a safe manner.
- Strategy – the ability to enhance the control of the power system and thus add to its capabilities.
- Coordination – the ability to cooperate with other users and power management systems.

These parameters are quantified in the tables below, with examples following, to form the autonomy metric, A, in (3.5).

Independence A_I	**Level**
3	>90% of previously manual tasks automated
2	67% of previously manual tasks automated
1	33% of previously manual tasks automated
0	<5% of previously manual tasks automated

Table 3.3. Operator independence.

For example, a power process that requires the regular entry of 10 operator-entered parameters could have 9 of these parameters automated by the application, thus saving the operator from 70% of his normal workload resulting in $A_I = 3$.

Preservation A_P	**Level**
3	>90% trouble conditions handled
2	67% trouble conditions handled
1	33% trouble conditions handled
0	<5% trouble conditions handled

Table 3.4. Self-preservation.

For example, a power process that has 10 pre-defined alarm conditions could have 7 of alarm conditions handled by the application, thus saving the operator 70% of his alarm-handling workload resulting in $A_P = 2$.

Strategy A_S	**Level**
3	Many new goals, or strategies, applied to enhance the system capabilities
2	Some new goals, or strategies, applied to enhance the system capabilities, multi-goal optimization
1	One new goal, or strategy, applied to enhance the system capabilities, single-goal optimization
0	No enhancement

Table 3.5. Strategy.

For example, a power process that currently depends on the operator to make all its strategic decisions is enhanced with the application autonomously seeking solutions for a few goals, e.g. reducing a particular pollutant from power production, minimizing equipment wear on a certain actuator, etc. These few goals enhance the power management solution and result in $A_S = 2$.

When assessing coordination, the architecture is evaluated by the ability of the application to coordinate its actions with other power management systems and software applications, and other users. At level 0, the application behaves as a typical piece of control logic. At level 1, information of other systems can be input to perform fixed calculations only. More than one user may direct control parameters. At level 2, the application begins to balance the control influence of multiple users and perform limited bidirectional communications with other applications. At level 3, full cooperation with all other entities (human and application) is achieved with at least some intuition. Table 3.6 quantifies this metric.

Coordination A_C	**Level**
3	Full cooperation with all entities, intuitive.
2	Limited coordination with other applications and coordination of the influence of multiple users.
1	Aware of other applications but little or no coordination. Ability to handle multiple user types.
0	Unaware of other applications. Only operator-level control by users.

Table 3.6. Coordination.

For example, a power process is currently operated as a standalone application, e.g. a single generating unit in a multi-unit power plant. When the power management application is applied to each of these units, they can be linked together to share some information about each other to influence their control. This simple awareness results in $A_C = 1$. Higher levels of coordination would achieve a higher A_C.

The autonomy metric, as defined, becomes a four-dimensional quantity. When comparing simple magnitudes between applications, a vector distance measure provides the best measurement. This is the distance from the origin in four dimensions where the origin represents no autonomy, i.e. all autonomy metrics equal zero.

$$A = \sqrt{A_I^{\,2} + A_P^{\,2} + A_S^{\,2} + A_C^{\,2}} \tag{3.5}$$

Often, a more granular measure of the autonomy metric is required to qualitatively assess the differences between applications. In this case, it may be preferable to view the autonomy metric on a four-dimensional radar graph as in figure 3.3.

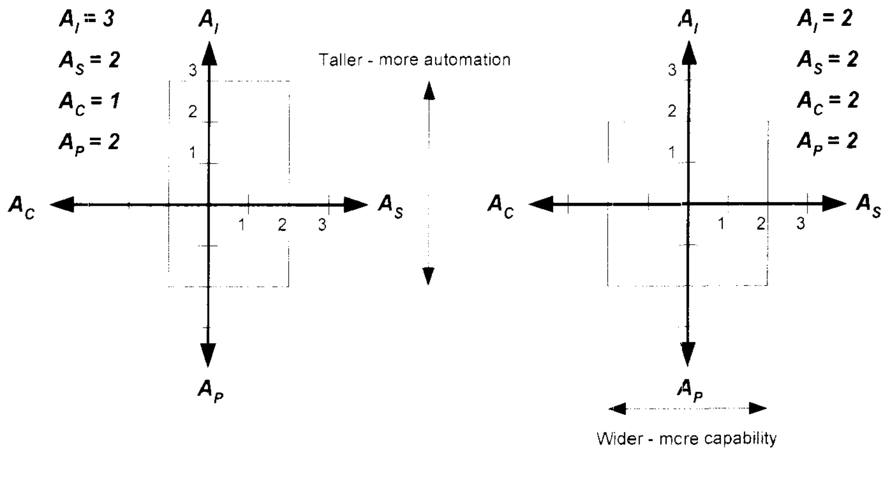


Figure 3.3. The autonomy metric.

In figure 3.3, the left-hand graph illustrates the above examples following the tables with a total autonomy metric of $\sqrt{18}$. The right-hand graph illustrates another application with a total autonomy metric of $\sqrt{16}$. Using the four-dimensional radar graph, however, allows us to see how the applications differ in each metric as a simple rectangle for a more qualitative analysis. Since total rectangular height is the sum of operator independence and self-preservation, this can represent a measure of simple automation, i.e. taller = more automated. Since total rectangular width is the sum of strategy and coordination, this can represent a measure of capability enhancement, i.e. wider = more capability.

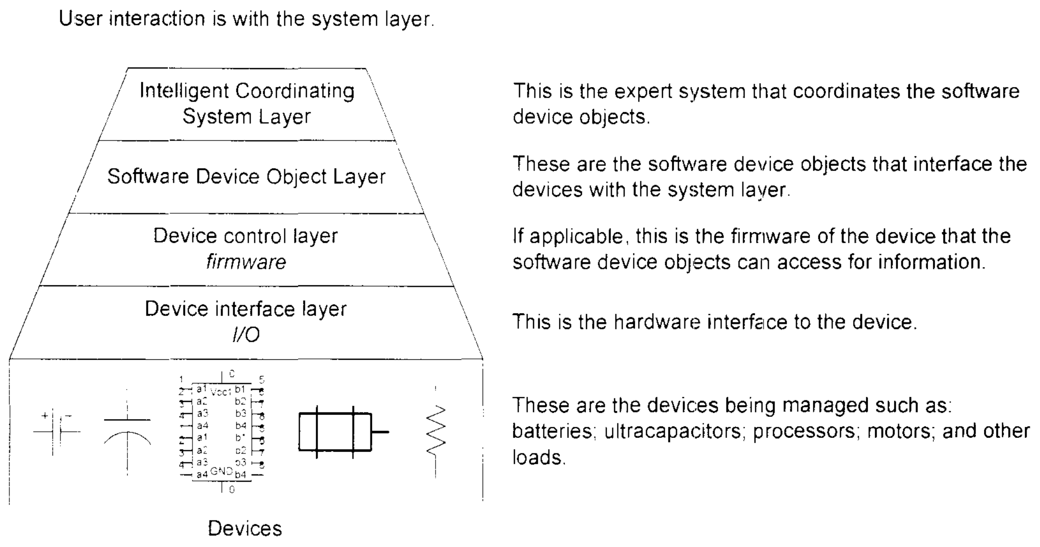
These metrics are applied in the implementation cases in *Chapter IV*. The first implementation case will evaluate an existing approach of an artificial neural network optimization of a coal-fired power plant. The second case begins preliminary development of the architecture with the application of software agents to a hydro-generating plant. The third implementation case is a power management system for the hydro-generating plant coupled to a personal hybrid vehicle, both utilizing the architecture presented in this chapter. In *Chapter V*, these metrics are discussed with relation to the architecture.

A Layered Approach

The architecture is designed as a layered approach, illustrated in figure 3.4. Individual devices in the power system are associated with software device objects and this constitutes the device layer. In this layer, the software device objects individually optimize the operation of the devices. In some cases, these objects may be able to act on their own or autonomously negotiate with other software device objects. In these cases, the software device objects act as agents [Foreman, 2008]. In other cases, these software device objects may only perform a few simple functions or even be limited to providing an interface to the next layer, which is the system layer.

In the system layer, the software device objects are coordinated to achieve a whole power system management scheme. This system layer incorporates an expert system to determine a management strategy based on the goals of the power management system and the statuses of the devices being managed. The use of an expert system allows an intelligent strategy to be produced based on deductive reasoning. The expert system is typically implemented with a set of rules that most closely resembles the way human experts understand the process, thus resulting in a more direct method of programming. In many cases, a classifier such as an artificial neural network can be used to reduce the number of inputs to the expert system and/or perform online feature extraction of the input data.

The layered approach is illustrated below in figure 3.4. This shows how the architecture is built from individual components up to a coordinated and intelligent power management solution. On the bottom are the devices to be managed. The next two layers are typically an existing part of the device provided by the device vendor. They provide an interface to the device from which the software device object layer can be constructed. Simple devices may not even have these layers. A smart battery, for example, may simply have methods for measuring the cell voltages and current draw and that is all. A solar cell may not have anything. A combustion engine may have the whole engine control system implemented here. In this case, the proposed power management system would sit on top of the existing control system. The final two layers are those explicitly defined by the architecture and are discussed next.



User interaction is with the system layer.
Intelligent Coordinating System Layer
Software Device Object Layer
Device control layer *firmware*
Device interface layer *I/O*
Devices
This is the expert system that coordinates the software device objects.
These are the software device objects that interface the devices with the system layer.
If applicable, this is the firmware of the device that the software device objects can access for information.
This is the hardware interface to the device.
These are the devices being managed such as: batteries; ultracapacitors; processors; motors; and other loads.

Figure 3.4. Layered approach.

The Device Layer

Power systems consist of various hardware devices of three types. The first type is a storage device, such as batteries and ultra-capacitors, which collect power through charging for later use. The second type is a source device, such as fuel cells, solar photovoltaic cells, and combustion engines, which generate power for both charging and operation. The third type is an electrical load device, such as motors, processors, and lighting, which consume the power to perform their mission. These basic types span missions that range from picosatellites to power generating stations. In picosatellites, for example, batteries and solar photovoltaic cells are relatively simple by design and therefore may have simple software device object definitions. In contrast, for power generating stations it may be more efficient to have a device definition that is a group of smaller components. For example, the whole turbine generator of a hydro-generating station may be more appropriately described as a single power source device, even though it consists of many components. The device definitions in this case may even include the classical control system software as one of its parts, similar to the firmware layer in figure 3.4. In this architecture, the software device objects represent the smallest component in which the power management system should be subdivided. The granularity of this breakdown would normally not go below the basic three device types of storage, source, or load as described above.

A loosely coupled architecture for the software device object is defined to characterize these devices so that the system layer can coordinate them as peers. While the internal methods vary according to the respective device being characterized, the same inputs and outputs for the software object are defined to achieve encapsulation. Figure 3.5 illustrates the basic architecture and the inputs and outputs of the software device object.

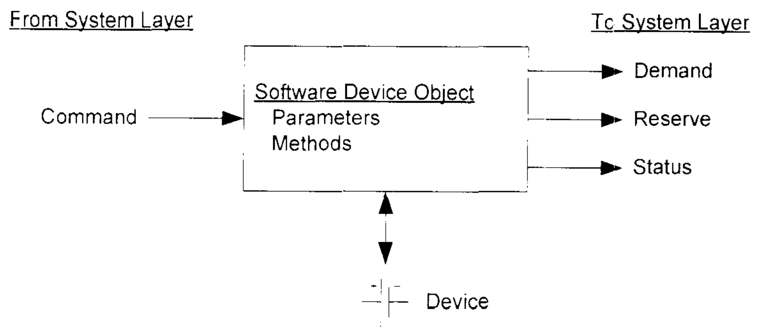


Figure 3.5. Software device object architecture.

The device object contains parameters that characterize the device, for example: dis/charging rates; operating limits; specifications; etc. The device object also contains methods that define how to calculate the outputs and utilize the command input for coordination with the system layer. The outputs of demand and reserve are normalized on a 0-100% scale and can be determined by the below pseudo code.

```
Params = { list }    ; parameters that specify device characteristics
Device.Type = (storageDevice, sourceDevice, or loadDevice)  ; choose type
       IF Device.Type = storageDevice THEN
              Device.Demand = f(Params)  ; calc power demand from device
              Device.Reserve = g(Params) ; calc reserve capacity of device
              IF Device.Reserve < Params.BatteryLow THEN
                     Device.Status = { currentStatus, BatteryLow }
              ENDIF ; add the BatteryLow status to the device status list
              IF Device.Command = chargeBattery THEN
                     setMode(chargeBattery) ; allow battery to charge
              ENDIF
       ELSE IF Device.Type = sourceDevice THEN
              Device.Demand = f(Params)  ; calc power demand from device
              Device.Reserve = g(Params) ; calc power available from device
       ELSE IF Device.Type = loadDevice THEN
              Device.Demand = f(Params)  ; calc power utilized by device
              Device.Reserve = g(Params) ; calc power requested by device
```

```
                                        ; for next mode of operation
          Device.Status = { currentStatus, ChangeMode to nextMode }
          IF Device.Command = permitModeChange THEN
               setMode(nextMode) ; put the device into the next mode
          ENDIF
     ENDIF
```

The status output provides the ability to report errors, trouble condition, or other general status messages, e.g. the status *BatteryLow* in the above pseudo code, to the system layer in order to assist the decision-making process of the expert system. The command input is the management response from the system layer that controls the device's power strategy to achieve coordination among all the devices, such as in the above pseudo code for *chargeBattery* or *permitModeChange* as a load device permissive.

The software device object utilizes methods and user-defined parameters to calculate the outputs and handle the command input. The method can be a simple equation, such as in (3.6) the demand for a battery, or a lookup table cross-referencing a set of operating modes versus power consumption for a complex load device.

$$D_{battery} = kVI \qquad \text{where } k \text{ is a constant, } V \text{ is voltage, and } I \text{ is current} \qquad (3.6)$$

More advanced methods are used to generate status messages based on device error or alarm conditions, or to handle the command input and change the operating mode of the device. Better methods enhance the information sent to the system layer and therefore improve the capability of the power management system. For example, a solar photovoltaic cell method may simply report the power generation available as a function of incident light, or it may report this value and additionally send a status message that more power generation is available if the cell's orientation towards the sun is changed.

Methods can also be used for local device power optimization for an enhanced power management solution. This is particularly applicable for complex devices such as loads that can manage their own power usage but still need to be coordinated with the system layer to achieve power management for the whole power system. For example, a communications system may employ a sleep mode or a burst transmission mode to achieve local power optimization, and the software device object will enhance this by interfacing with the system layer so that cooperation among other devices is achieved.

The software device objects are typically resident in the same computer-processing level as the system layer, although smart devices with their own firmware environment may implement their software device objects at their local device level. Figure 3.6 illustrates an expanded software device object highlighting this.

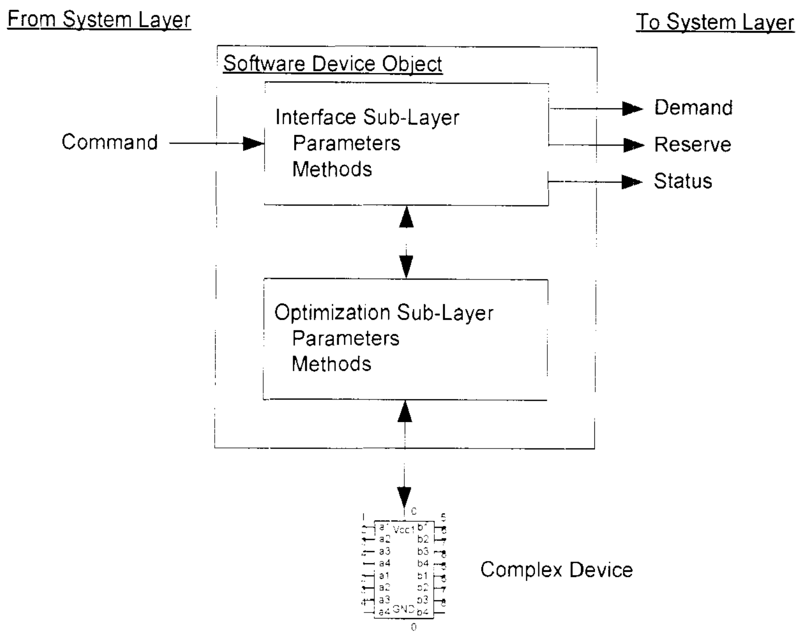


Figure 3.6. Software device object architecture expanded with optimization.

In figure 3.6, the software device object is expanded to include optimization for a more complex device object. The interface sub-layer provides the same input and outputs that are utilized by the system layer for coordination as discussed for figure 3.5. The optimization sub-layer, however, acts between the interface sub-layer and the device so that more advanced methods can be included in the software device architecture. The optimization has its own parameters that define the boundaries of the optimization and its own methods that implement the optimization. These methods are designed to handle devices with multiple power modes or where the same device operation could be obtained in multiple ways, thus requiring an optimization method to determine the approach of minimum cost with respect to the optimization parameters. The device and whole power system benefits from this software device object enhancement.

In power generating stations for example, the software device objects may be sophisticated enough to be classified as software agents. Software agents are software objects that function as autonomous entities, which act with an agency-type relationship for users or other software objects. That is, software agents can automatically make decisions and take actions on behalf of users or other software objects to achieve the goals of the power management system. Because the devices being managed (typically power source devices) are combinations of many subsystems and have an existing control system for their general operation, the system layer relies on the device layer to negotiate with the device's existing control system. This is illustrated in figure 3.7 as another expansion of the software device architecture in figure 3.5.

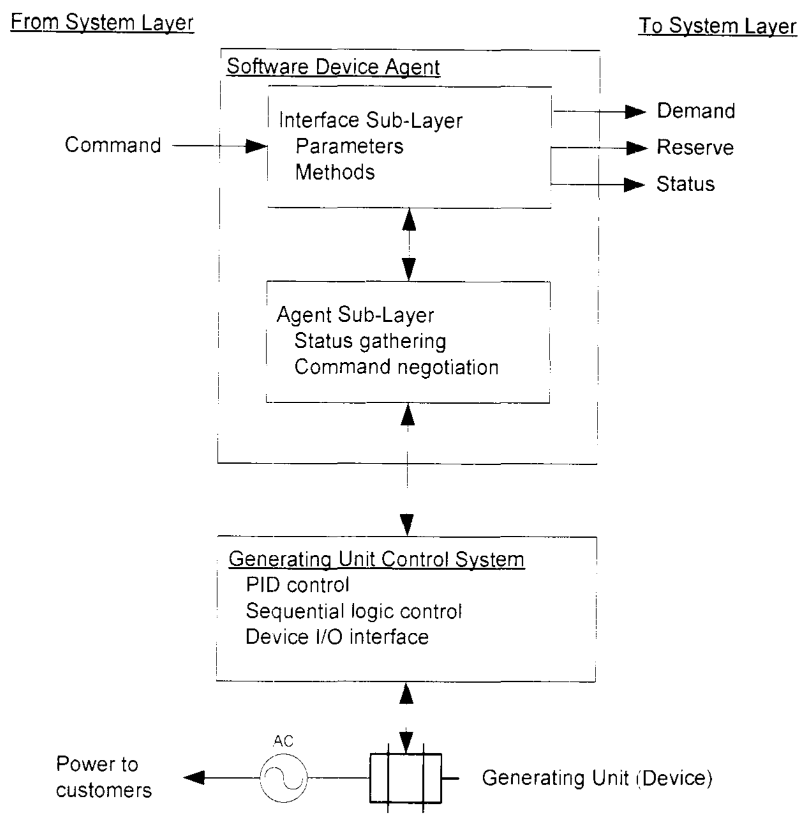


Figure 3.7. Software device object as a software agent.

In figure 3.7, the software device object is implemented as a software device agent for a power-generating unit. The interface sub-layer again provides the same input and outputs that are utilized by the system layer for coordination. The generating unit has its own existing control system for general operation incorporating PID control, sequential logic control, and a device I/O interface. These elements of the generating unit's control system are discussed further in *Chapter IV*. The distinction for a software device agent, however, is that an agent sub-layer exists between the interface sub-layer and the generating unit's control system and acts with an agency relationship on behalf of these autonomously. Therefore, the agent sub-layer gathers status information from the generating unit's control system based on what the software device agent thinks is necessary to provide the output information to the system layer. Furthermore, the agent sub-layer performs command negotiation taking the command input from the system layer and merging this into the generating unit's existing control scheme.

Details of the implementation of software device objects and agents, including example case studies, are discussed in *Chapter IV*. Use cases of the architecture are presented in a following section of this chapter.

The System Layer

The system layer coordinates the software device objects, and subsequently the power system devices, to achieve an intelligent power management system. The system layer utilizes outputs of the software device objects and coordinates them by sending a command input back to them. All communication in the architecture is in a *star network* configuration whereby each software device object communicates individually with the system layer. Thus, all software device objects or agents both with and without internal optimization appear the same to the system layer. The core component of the system layer is a rule-based expert system. An expert system allows an intelligent solution to be deduced logically. Employing a rule-based approach simplifies coding in that rules are a natural way for human experts to think about processes. Rules are modular, so they can be added and removed easily. Rules are also a white-box approach so that their probable actions can be determined by observation of the rule syntax.

In the device layer section, software device objects were defined with the outputs of demand, reserve, and status. Utilizing the demand and status outputs enables the system layer to determine the current operating state of each device. By utilizing the reserve output, the system layer can also determine the next operating state of each device since this variable includes reserve capacities for power storage and power source devices, as well as the reserve power requested by load devices for their next operating state. Therefore, the operating state sent to the system layer includes both existing and future information, providing a faster than real-time classification. This helps address the limitations of optimizations that do not know the process *a priori* and therefore result in suboptimal results [Schupback, 2003].

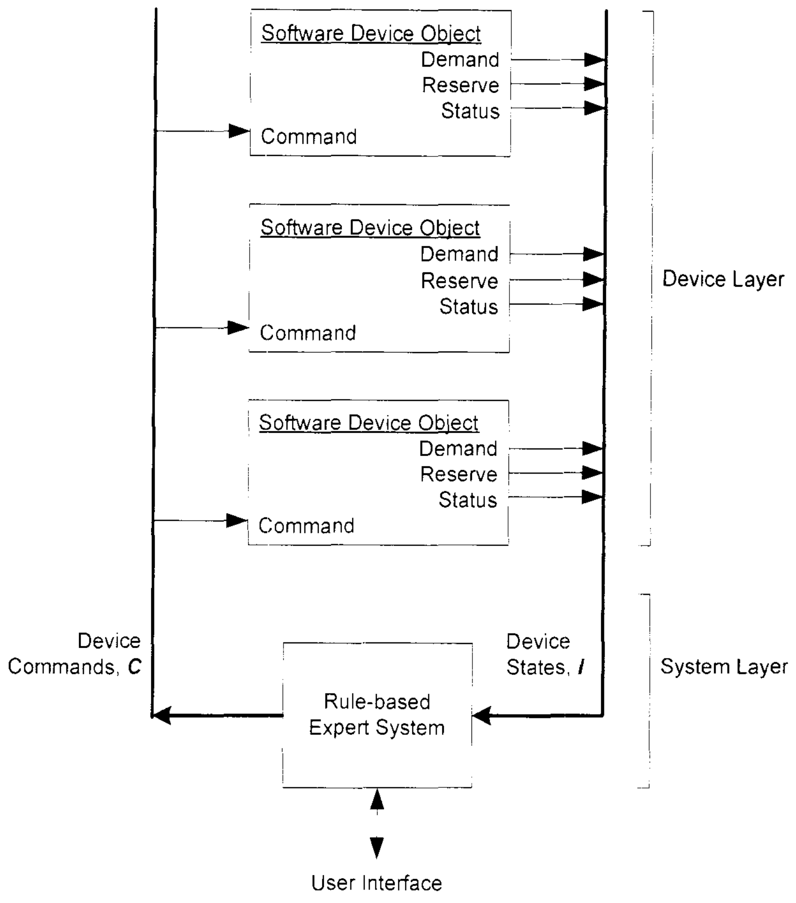


Figure 3.8. System layer coordinating multiple software device objects.

In figure 3.8, each software device object sends its outputs to the expert system. This results in a set of variables or a vector denoted by ***I***, device states, that are inputs to the expert system. These variables are used in building the rules, which then assemble the device command as an output from the expert system. The device command, ***C***, is defined so that when input to the software device objects, the desired action is taken by the software device object. The device command may be a single command sent to a single software device object or multiple commands sent to multiple software device objects. Details of how the device commands are formed and addressed are discussed in *Chapter IV.* A user interface is also shown as the interaction point for the user utilizing the power management system. Use cases of the architecture are discussed in the so-named following section in this chapter.

With a large number of power system devices, widely varying device types, or complex rule sets, it may be desirable to have some classification performed on the devices states, ***I***, prior to processing with the expert system. This simplifies applications when there are a large number of operating states by reducing dimensionality and/or when preprocessing for data feature extraction is beneficial for rule definition. In these cases, artificial neural networks can be included to perform this classification for the expert system. Since neural networks accept analog data and provide analog output, they result in fuzzy classification and do well interpolating over the operating state space. The enhancement of the system layer by a neural network classifier is illustrated in figure 3.9. This enhancement becomes the typical architecture for implementation as all but the simplest of power systems benefit from this additional functionality.

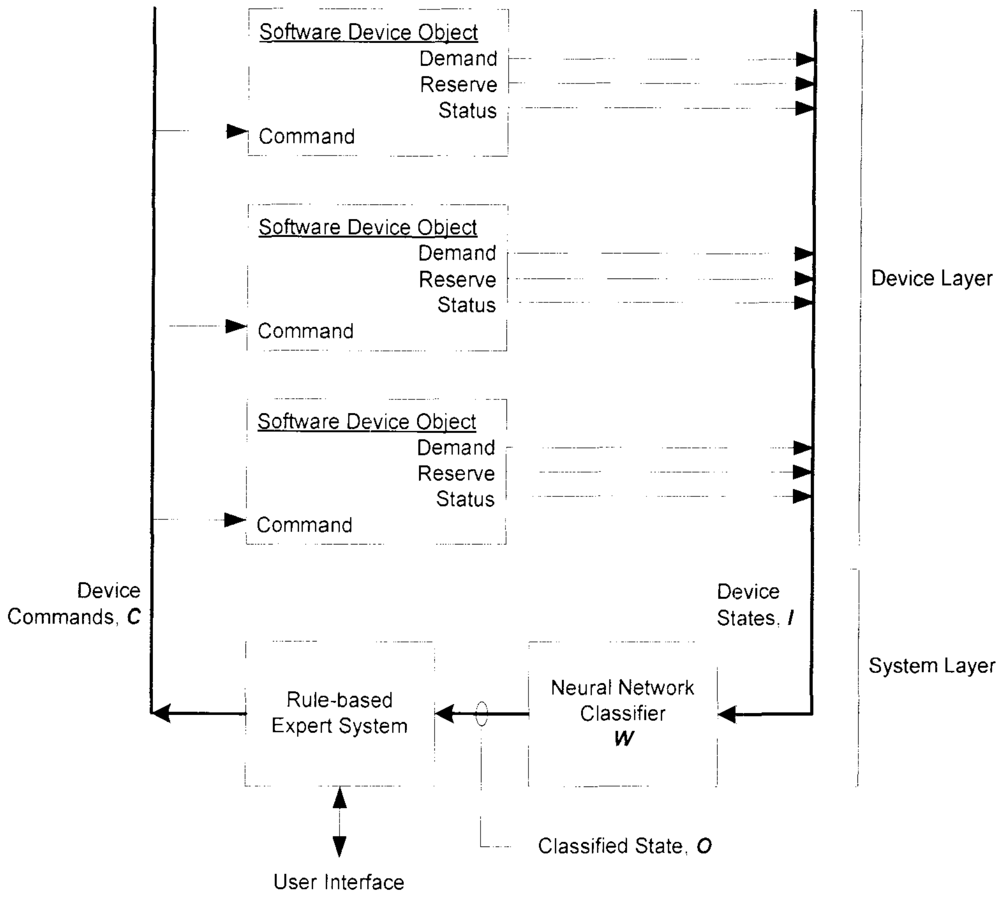


Figure 3.9. System layer enhanced with neural network classification.

The neural network classifier is added inline between the software device objects' device state vector, ***I***, and the expert system. The output of the neural network is a classified state vector, ***O***, that is a superset of the current and next power system operating states. This vector, ***O***, becomes the new input to the expert system and provides the variable set that is utilized by the rule set to form the device commands vector, ***C***. The neural network is as typically defined by (3.7).

$$\dot{O}_{m\times 1} = N\left[\overline{W}_{m\times n}\dot{I}_{n\times 1}\right] \tag{3.7}$$

The neural network is characterized by weight matrix, $\boldsymbol{W}$, of dimension m x n determined by n, the dimension of the device state vector, $\boldsymbol{I}$, and m, the arbitrary dimension of the classified state vector, $\boldsymbol{O}$, that is the result of the neuron activation function, $\boldsymbol{N}$. The neural network may be implemented in multiple layers by a nested application of (3.7) although two layers are typical. The neural network can be developed, or trained, offline and before deployment so that once in place, the power management system needs to perform only the function in (3.7), thus minimizing the processing footprint.

Also included in the classified state vector, $\boldsymbol{O}$, would be any data feature extraction with respect to the device operating states. For example, a quantized measure of the power system's stability might be too difficult to code directly, but a neural network can learn to recognize this quantity from the device state vector, $\boldsymbol{I}$, similar to pattern recognition. Details on neural network classification and rule development are discussed later in this chapter.

Integrating the Device and System Layers Together

The device layer and system layer together form the power management system, which manages the power system. This power management system provides for individual management and optimization capabilities through the custom methods in the software device objects. The power management system also integrates the power devices through their software device objects to achieve coordination of the whole power system. This addresses the respective limitations of the de/centralized designs while still providing the benefits of such designs [Vahidi, 2007].

The pseudo-format of variables used for data communication between the device and system layers is proposed as the following, where *n* is the number of software device objects:

- Inputs and outputs for software device objects (SDO):
    - command = <*device.id*, *device.commandcode*>
    - status = <*device.id*, *status.statuscode*>
    - {demands, reserves} = analog value of 0..1, (0-100%)
- Input and output for neural network classifier (NNC):
    - Input vector, ***I*** = [[SDO1.{demand, reserve, status}
      SDO2.{demand, reserve, status}
      SDO*n*.{demand, reserve, status}]]
    - Classified vector, ***O*** = [[class characteristic1
      class characteristic2
      class characteristic*n*]]
- Input and output for rule-based expert system (RBES):
    - Classified vector, ***O*** as above for NNC
    - Command vector, ***C*** = [[SDO1.command
      SDO2.command
      SDO*n*.command]]

For the software device objects, the *device.id*, *device.commandcode*, and *status.statuscode* are user defined for the implementation. *Demand* and *reserve* values are normalized on a 0-100% scale of capability as previously defined. The neural network classifier takes a vector of all software device object outputs as its input, ***I***, and supplies the classification as, ***O***. Class characteristics are determined by training the neural network to recognize patterns in ***I*** and would include user-defined feature extraction such as: measure of transient demand; measure of steady-state demand; measure of power storage; health of power generation; urgency of next requested state; and any additional characterizing quantities. The rule-based expert system then takes ***O*** and uses the rule set to deduce the command vector, ***C***, which is a vector of all the commands to be sent to the software device objects on the current calculation cycle. Further details such as those of the neural network classification characteristics or of calculation cycle timing are left in *Chapter IV* as they are application specific.

The software implementing these layers can reside on a single or multiple processor system and is typically coded in an embedded, object-oriented environment designed for real-time process control. Communications would utilize the existing network and device I/O infrastructures that are typically a part of such control systems. In *Chapter IV*, it is described how the architecture is coded in various control schemes from micro-controllers to plant-scale distributed control systems (DCS). Such systems include special data structures commonly referred to as process points, data points, tags, etc, that natively utilize the communications infrastructure of their control system to enable real-time control.

Use Cases of the Architecture

In most applications, the power system is a critical yet secondary sub-system. In other words, while the power management system is necessary for completing the mission, the user uses the whole machine to affect completion of the mission and typically relies on the power management system to autonomously work in the background. For those cases when a user needs to interact with the power management system, the following use case in figure 3.10 demonstrates how this user can utilize the architecture.

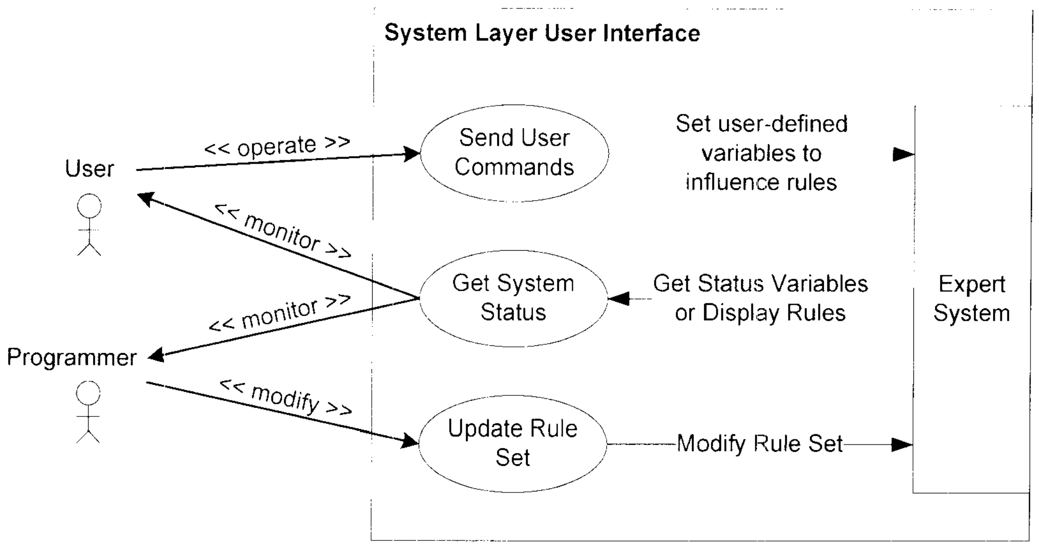


Figure 3.10. Use case for human users of the architecture.

In figure 3.10, two user types are presented. The *user* will monitor the power management system and enter commands to guide the power management system's optimization of the power system. In the case where the software device objects were replaced with software agents, as in figure 3.7, the user would interact further through these commands to influence operation of the individual devices when desired. The *programmer* may also monitor the power management system but would additionally make modifications when necessary to handle changes in the power management system's mission objectives.

In figure 3.11, the use case of the power system devices utilizing the architecture is presented to further illustrate how these devices interact with the architecture.

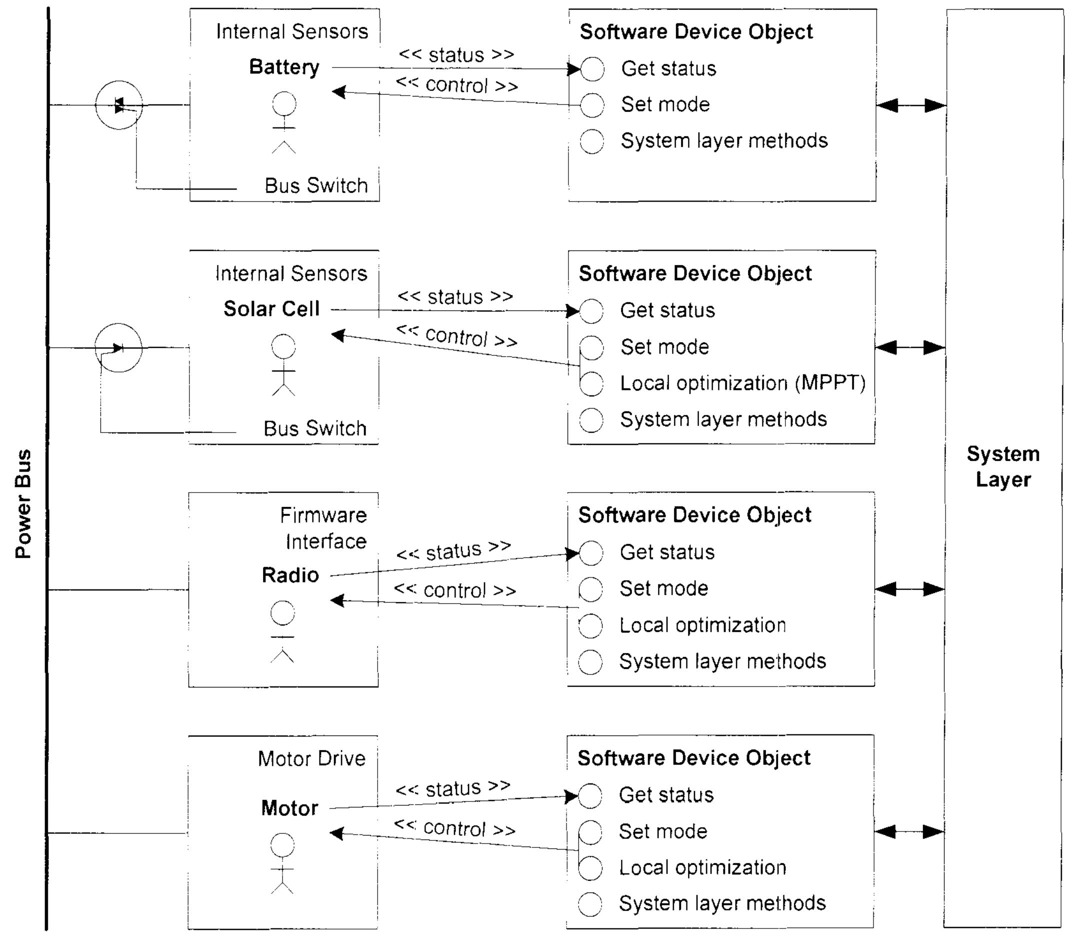


Figure 3.11. Use case for power devices of the architecture.

In figure 3.11, the devices, depicted as users, interact with the architecture, specifically with the software device objects in the device layer, in two generic paths regardless of device type. First, the devices supply their statuses to the software device objects through the *get status* method via the *status* path. Second, the devices receive control inputs from the software device objects through the *set mode* method via the *command* path. Some software device objects may include a *local optimization* method, for example maximum power point tracking (MPPT) for solar cells, to provide additional optimization of the devices, although interaction with the device is still along the control path. Lastly, the software device objects implement the previously discussed *system layer* methods for interaction with the system layer, although the devices do not typically use these methods directly.

Also in figure 3.11, the devices have included components illustrating how they would provide the statuses and utilize the commands received from the software device objects. Internal sensors, such as for voltage and current, measure and provide these quantities to the software device objects. A *bus switch* may be available that connects the devices to the power bus. This can provide either a dis/connect functionality or perform voltage matching via DC-to-DC converters. More complex devices have *firmware* that can provide a library of methods that a software device object can utilize. Other devices may have special functionality such as a *motor drive* for motors that software device objects can query for status and tune for performance. Further details are application specific and are given in *Chapter IV*.

Additional Layers of Enhancement

In the preceding sections, the architecture has been developed for power management and optimization systems residing within the existing control system and in direct application with the process at its fundamental level, i.e. control of the process through direct influence of the physical devices in contact with the process. This represents the core application of the architecture. In larger implementations, additional layers may be necessary as an expansion of the core application to build a complete power systems solution. Two expansions that are investigated here are a *data-mining layer* and an *enterprise-level layer*. An expanded version of figure 3.4 is given as figure 3.13 illustrating these additional layers after their discussion.

Data Mining Layer

The layered approach of the architecture data-mining layer is expanded to include the data-mining layer, which resides alongside the device layer and system layers. This layer performs two functions that may be essential for some applications. The first function of this layer is to collect and store data of the process. This data contains periodic real-time data from the power system and the overall process and itself. Since the data is real-time and may come from multiple sources, it is important to ensure that the data is time-synchronized such that variables from different sources can be correlated. This first function, therefore, forms a data warehouse providing historical operating data that serves as a resource for the device layer and system layer to aid in their optimization and management efforts. The second function of the data-mining layer is to perform analysis of the data, as needed, in the form of data mining. This analysis can uncover previously unknown relations in the data and enhance the capabilities of the device and system layers.

Business Entities and the Enterprise-level Layer

The layered approach of the architecture is expanded to include the enterprise-level layer, which resides atop the system layer. The enterprise-level layer handles all the outside users of the architecture providing them with status information and accepting control influence from them. In small or mobile applications, such as a vehicle, this may be anywhere from zero to a few users and in these cases, a simple human machine interface (HMI) would suffice. For larger implementations such as power generating plants, the enterprise-level layer handles several business entities. In this case, many user types will have differing goals and need differing levels of access. These user types are business entities beyond just operators and engineers to include marketing, power trading, corporate management, environmental compliance, etc. Figure 3.12 demonstrates the business entities for a typical power generating enterprise.

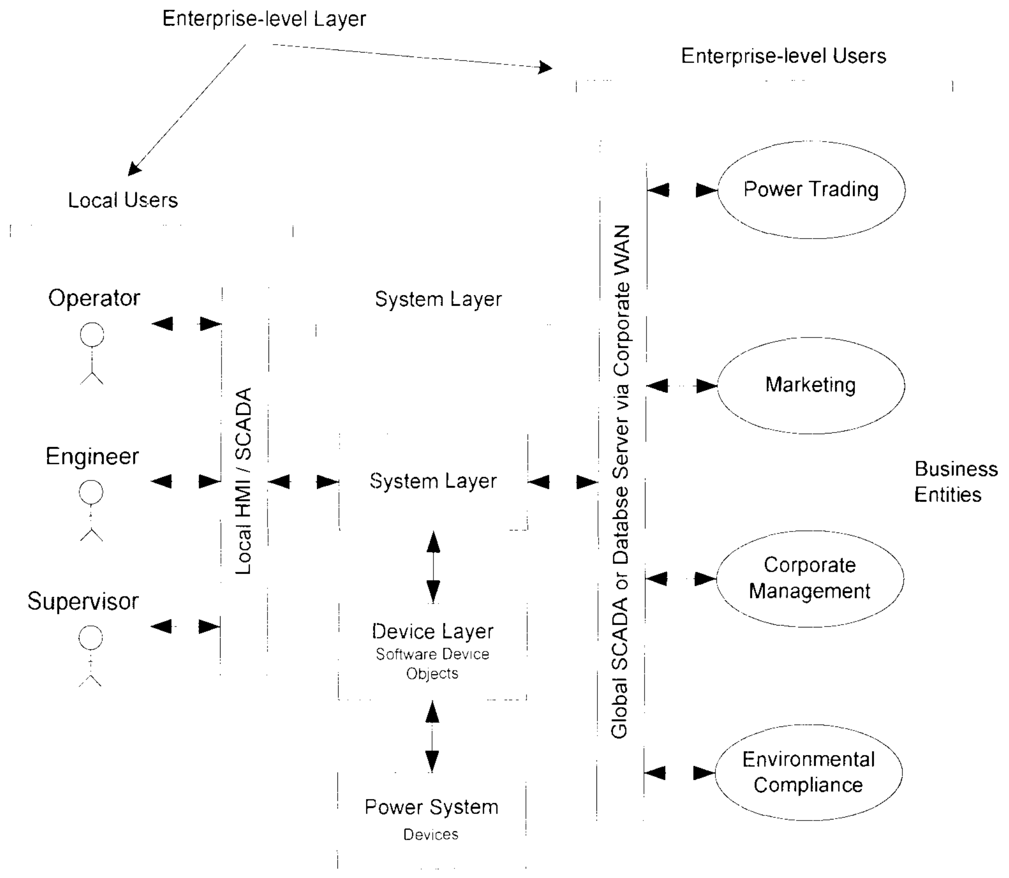


Figure 3.12. Power generating enterprise example.

The enterprise-level layer is where the application of the architecture includes diverse business entities intended to implement an enterprise-level solution. Business entities have differing needs of the architecture and therefore attempt a local optimization from their perspective, i.e. the environmental compliance entity attempts to minimize the emission of pollutants. A global SCADA or database server controls security access to the system layer. The system layer then prioritizes and incorporates these business entity directives into the solution using the previously mentioned rule set for power management.

Power scalability, i.e. the size of power resources handled, has an affect on the architecture. Large amounts of power typical of generating plants for profit call for special structures and hence the enterprise-level solution. Alternatively, power storage is a limitation in mobile systems that manage small power resources. The architecture handles power scalability by continuing the layered approach to achieve a complete and balanced solution at all hierarchical levels.

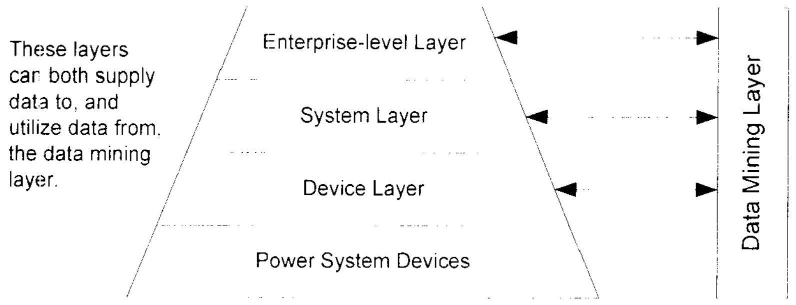

These layers can both supply data to, and utilize data from, the data mining layer.
Enterprise-level Layer
System Layer
Device Layer
Power System Devices
Data Mining Layer

Figure 3.13. Data mining and enterprise-level layers.

Conflict Resolution in the Architecture

To understand how conflict resolution is achieved by the architecture, the sources of conflict are first determined. Conflict is a disagreement between entities regarding a common point. The inability of a slave entity to follow a command from the master, and multiple entities trying to utilize a limited resource are examples. Sources of conflict arise between multiple software device objects in the device layer, between a software device object and the system layer, and between the system layer and external users. The rule-based approach provides natural resolution ability in the architecture.

Software device object conflicts

The architecture is defined such that the software device objects and software agents in the device layer only provide local control for their respective device and rely on the system layer for coordination. The system layer thus acts as a centralized governor based on its rule sets to resolve device layer conflicts. An example of software agents resolving their conflict is given in the hydro-generation case in the implementations in *Chapter IV*. In this case, agents compete when the limited resource of available river flow is increased. The most efficient agent has first priority over taking additional flow. Since the agents each know their efficiency and the efficiency of the others, the agents resolve this conflict using their rule sets. These are presented in the implementation case.

Software device objects and the system layer

Conflicts between the software device objects and the system layer are the result of commands sent by the system layer not being able to be performed by the software device object. For example, the system layer commands a generating unit to increase its power output, however, the generating unit cannot provide this increased output due to some problem. The software device object would respond to the system layer through the *reserve* output what power was available. The software device object, through its status output, would also report any trouble conditions. The system layer would then take this information and adjust its management strategy to cope with the limitation. This strategy may be to seek the power resource elsewhere in the system or to reduce the requirements of the process until such resources are available.

The system layer and external users

Conflicts between the system layer and external users arise can arise when external users make demands the power management system cannot satisfy or resolve. This is handled the same as the above case between the software device objects and the system layer. When multiple users are attempting to influence the system layer simultaneously, the system layer will need to prioritize these requests to resolve them. The hydro-generating case in the implementations of *Chapter IV*, as well as the enterprise-level discussion above, addresses this scenario by providing security through the control system SCADA interface. External users are assigned pre-defined process points to communicate with the system layer. The system layer can then internally prioritize the users' needs and deliver an optimized solution. This also prevents external users from accessing control areas that are restricted to them.

State Transitions of the Power System Utilizing this Architecture

The operating state is classified by the neural network as a combination of the current and next requested operating states from the supplied outputs of the software devices objects. This allows the architecture to have a faster than real-time performance to anticipate future power demands. Figure 3.14 illustrates a sample operating state transition cycle.

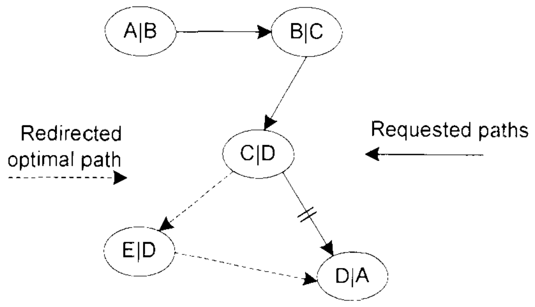


Figure 3.14. Sample power system state transition scenario.

Each operating state is summarized as *currentState* || *requestedState* as determined from the demands (current power demand), reserves (reserve capacity or requested load state), and statuses (device condition) provided by the software device objects of the respective power system devices. In this example, this classification is sent to the expert system, which either sends the command to the load devices permitting their desired transition path, or computes an alternate path for optimal power management. In figure 3.14, the states are defined as:

A. System is idle

B. Radio is receiving message data

C. Message data processing to calculate response

D. Transmit the response

E. Direct solar cell charging of ultra-capacitor

For this scenario, the power system starts in state ***A*** and must transit to state ***B***, since radio reception is an outside influence and cannot be scheduled. The power system transits to state ***C*** and attempts to subsequently transit to state ***D***. However, this radio transmits in a short high-power burst requiring the ultra-capacitor, which is currently not charged. Therefore, the expert system allows ***A→B→C*** and denies ***C→D***, instead forcing

C→E→D as an optimal power management strategy. Thus, the neural network determines the state classifications, and the expert system determines the paths between states.

Communications Timing in the Architecture

The power system and overall process produce and process real-time data from multiple components of the system. When coding the software components of the device and system layers, it is important to consider the paths that the process data takes within the control system and their respective delays. Vehicular systems are tightly integrated and often incorporate a high-speed network. Industrial control systems are often less intimately connected as a result of being composed of components from various vendors and implemented at various times. Therefore, communications timing is a larger design factor for these systems and it is easier to illustrate the complexity of paths in this environment. Figure 3.15 illustrates some process data paths with typical delay times between various components for a distributed industrial control system.

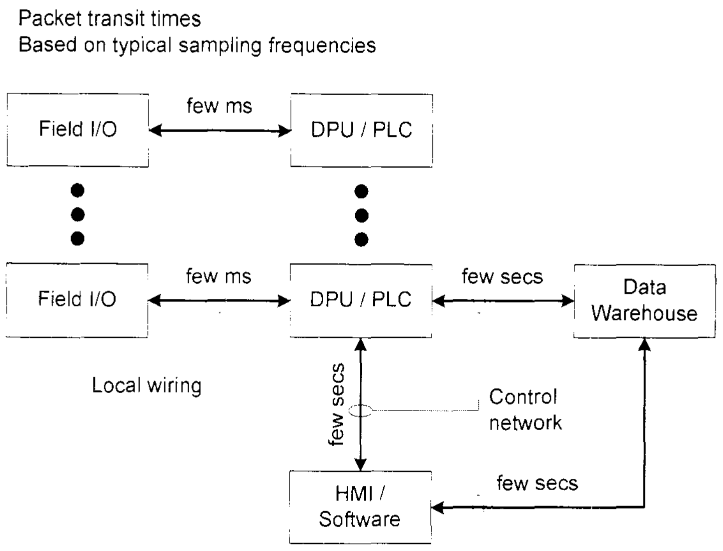


Figure 3.15. Data pathway timing for industrial control system.

In figure 3.15, it is seen that while the Distributed Processing Unit's (DPU) communications with the field I/O is on the order of a few milliseconds, access to that process data by a human operator or even another computer system is on the order of a few seconds. These time delays will influence the configuration of expert system rules, device layer methods and even location of these software components in the overall control system.

CHAPTER IV

IMPLEMENTATION

In the first section of this chapter, considerations for the unique environments are presented for power generating plants and vehicular systems. The special topics of proprietary systems, safe and reliable operation, and the PID algorithm are discussed briefly as well as a precursor to the implementations. The implementations presented here will first be a coal-fired generating unit with optimization to reduce emissions. This implementation demonstrates the limitations of monolithic neural network optimizations and serves as a motivation for a better architecture. The second implementation is a hydro-generating plant to optimize efficiency. In this implementation, some aspects of the architecture are introduced to address the limitations discovered in the previous coal-fired implementation. The third implementation is of a power management system for the hydro-generating plant coupled to a personal hybrid vehicle. This case demonstrates the industrial-scale application in cooperation with a small and mobile application. This encompasses power generation, storage, and utilization in a mission dependent on an autonomous management solution. Together, these implementations demonstrate the inspiration, growth, and development of the architecture and its ability to be applied across multiple applications.

Power Generating Plants vs. Vehicular Systems

While there are significant differences between the large industrial power generation plants and the small mobile vehicular systems, the architecture applies well to both platforms. Power generation and power utilization are key elements to optimize and manage in both cases. In power generation, the benefits of generating power in an environmentally clean and cost efficient manner scale with the quantity of power being produced. The generating company can manage its generating assets but often is unable to affect meaningful control over the load of the many individual customers. For this reason, industrial-scale optimization efforts only focus on one part of the solution, that of generation. It is important to note, however, that since the architecture is co-developed to manage vehicular systems as well, the components needed to enhance power generation optimization with the many customers representing the load is also present. Developing power management and optimization for vehicular systems therefore enables a more comprehensive power generating plant scheme. Similarly, vehicular systems represent a microcosm of the industrial power-generating platform. In their case, the generation and load components are more intimately joined and are both available to be managed by the software. This provides an opportunity to demonstrate the full potential of the architecture.

Considerations for Power Generating Plants

The typical power generating plant employs a Distributed Control System (DCS) for its primary process control. The DCS is comprised of multiple distributed processing units (DPUs), each with their own memory, control logic, and field I/O. The DPU is capable of handling thousands of points of I/O for controlling multiple sub-processes in parallel. The DPUs are networked together to provide a total process control solution. Often programmable logic controllers (PLCs) and/or other devices are used to provide ancillary or balance-of-plant type process control, such as when new stand-alone systems are added or when incorporation into the existing DCS is not feasible for some reason. In the last several years, the functional division line between the DCS and the PLC has become blurred with the advancement of PLC technology and PLCs are taking on a larger process control role. Therefore, both DPUs and PLCs are similar as controllers, typically varying in size more than other aspects. There are human-machine interfaces (HMIs) that allow operator interaction with the control system and subsequently the process. Finally, a data acquisition system (DAS) for archiving of process data is present in most modern control systems to serve as a baseline for plant operation and a diagnostic tool for fault analysis. These components may be interconnected with an Ethernet or similar network infrastructure. Figure 4.1 illustrates this layout. Figure 4.2 illustrates the functional diagram of the DPU and PLC controllers.

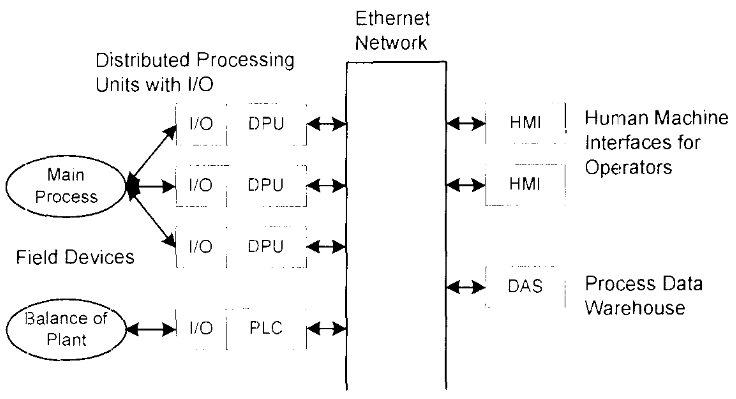


Figure 4.1. Industrial control system overview.

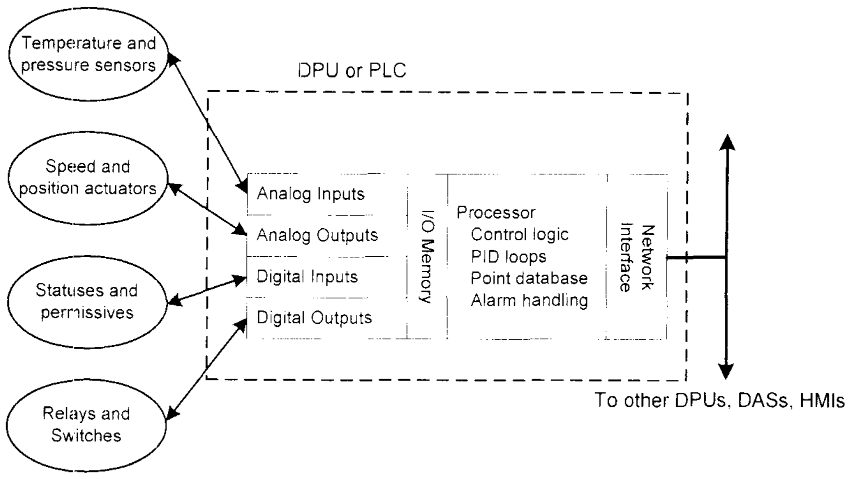


Figure 4.2. Functional architecture of DPU and PLC.

The I/O section consists of multiple input and output interface cards with termination blocks for field device wiring. These interface to the I/O memory via analog-digital converters or relays as appropriate to communicate analog and digital data to points mapped in the point database. These data points can then be manipulated as variable registers in the program logic allowing field sensory data to be utilized as inputs and field actuation devices to be controlled as outputs, thus effecting control of the process. A network component is included for communications to other controllers, HMIs, and other devices as needed.

Proprietary Systems in Power Generation Control

Control systems are not typically developed with the standard programming languages used in other fields. This is particularly true of legacy systems although some newer systems are beginning to incorporate interfaces to popular languages. Process control logic can be developed in formats of: structured text; functional block or

SAMMA diagrams; ladder logic; etc. Structured text is often similar to BASIC and other standard sequential languages with special functions for process control. Functional block diagrams are used to graphically connect blocks of algorithm code to produce a program. Figure 4.3 illustrates a sample function block diagram. Ladder logic is derived from relay logic used before the advent of computer-based control. It is designed to be easily readable and perform digital logic well. Figure 4.4 illustrates a sample ladder logic diagram.

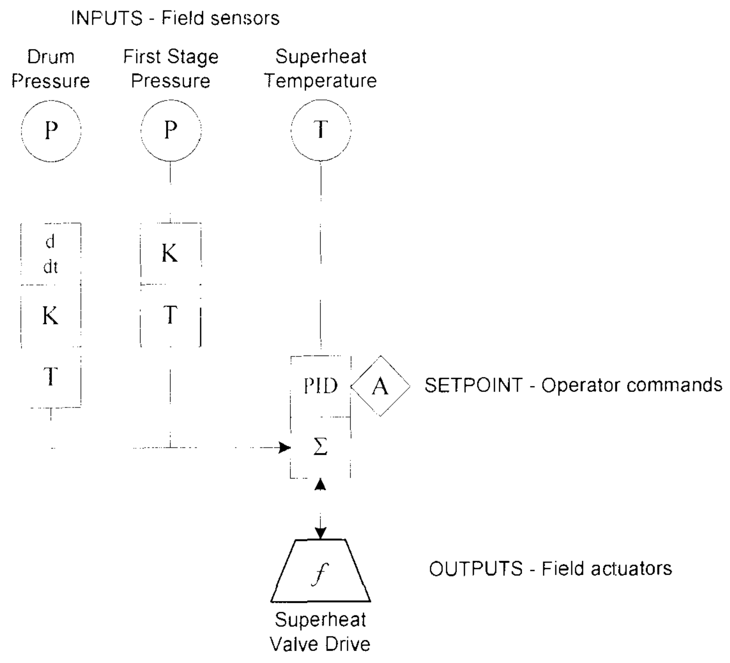


Figure 4.3. Sample of function block diagram or SAMMA.

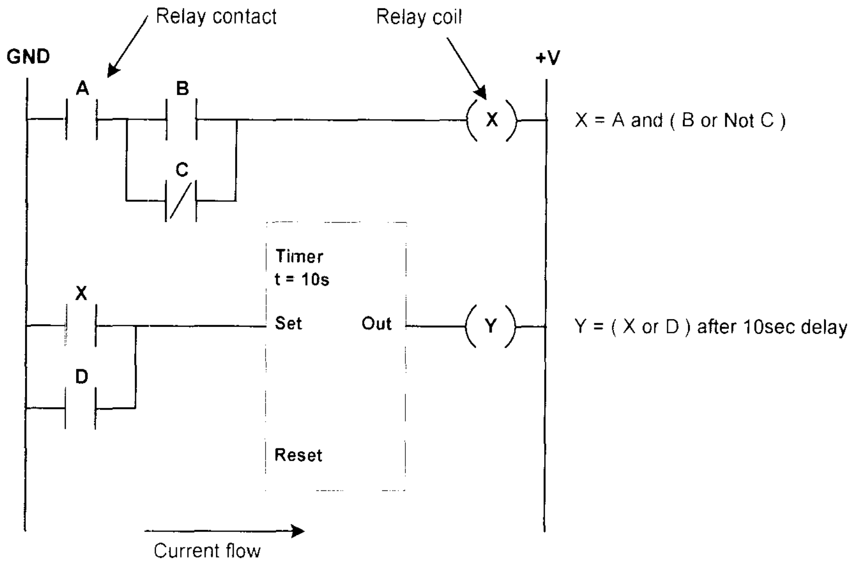


Figure 4.4. Sample of ladder logic diagram.

Proportional Integral Differential Control - PID

The cornerstone of industrial process control remains the proportional, integral, and differential algorithm or PID. PID control is used in more than 90% of control loops and predates digital control systems [Knospe, 2006]. A simple PID algorithm is given in (4.1). The constants k_p, T_i, and T_d refer to tuning parameters for the proportional, integral, and differential aspects of the algorithm, respectively. The attribute *AO* refers to the analog output of the PID, which drives the control element. *PV* refers to the real-time process value to be controlled. *SP* refers to the process setpoint for control. The exact mathematical implementation may vary among manufacturers but the general definition is maintained. The error is represented in (4.2) as ε.

$$PID.AO = k_p\left(\varepsilon + \frac{1}{T_i}\int(\varepsilon)dt + T_d\frac{d}{dt}(\varepsilon)\right) \quad (4.1)$$

$$\varepsilon = \pm(PID.PV - PID.SP) \quad (4.2)$$

Therefore the control action of the PID output includes: a linear gain component proportional to the error; an integral component that accumulates as the error persists in time; and a derivative component that accounts for the rate of change in the error signal. The PID algorithm provides an excellent source already existing in the control platform for real-time error control of an analog process. For a more in depth study of PID theory and control principles, refer to *PID Controllers: Theory, Design, and Tuning* by K. Astrom and T. Hagglund [Astrom, 1995].

While PID loops are the foundation upon which industrial control systems are built, it has been estimated that 50% of PID loops display undesirable characteristics, 37% need retuning once per year or more, and only 22% of those retuned show improvement [Morrison, 2005]. It is also estimated that PID loops are operated manually or in a suboptimal mode 65% of the time [Astrom, 1995]. This indicates the need for a power management system designed to fit within the existing control framework. The architecture of *Chapter III* fits into this framework.

Considerations for Safe and Reliable Operation

Since the program code or logic is used for controlling a physical process, safe and reliable operation becomes important. Interlocks, or permissives, are often used to provide a checklist before permitting certain actions to be taken. For example, before starting a motor, ensure the area is free of personnel and that the load is ready to be driven. It may also be necessary to check that sensory input data is valid. For example, if a pressure sensor fails by ceasing to give valid data, control logic needs to alert the operator to this condition and either handle the event or fail in a safe mode. Likewise, calculated outputs to field actuation may need other limit or sanity checks to ensure proper process control. Process control systems operate in real time. Therefore data can become obsolete and commands need to be executed on a strict schedule. The process also incorporates time constants. When commands are given, the process requires a settling or response time to react. Analog commands may need to be gradually incorporated or ramped in to avoid process instability.

Considerations for Vehicular Systems

Vehicular systems represent a microcosm of the industrial-sized implementations in that power generation, storage, and utilization are all incorporated into a small single mobile system. This provides an excellent demonstration of scalability in the architecture and allows the power management features to be explored from the perspectives of all the device types in one application.

Motivations for Vehicular Power Management Systems

There are three main motivations for vehicular power management and optimization. First is the reduction of emissions and fossil-fuel dependency. Automobiles represent the majority of vehicular implementations and given the large number of them in use, automobiles become a significant consumer of fossil fuels and a significant producer of greenhouse-gas emissions. Hybrid and all-electric automobiles are becoming popular and yet have much development ahead. An improvement in efficiency for these vehicles directly benefits the environment and reduces foreign dependence on resources.

The second motivation is the optimal use of power given limited storage and generating options. Vehicular systems are mobile by definition and therefore have size and mass limitations in addition to limitations in field maintenance for some applications, e.g. spacecraft. The third motivation is handling the complexity of many diverse power devices and integrating mission parameters into an intelligent management solution. When new devices, systems, and missions are developed, new management solutions must be developed as well. The architecture is designed to grow with these developments and minimize redesign costs. The intelligent power management system also integrates the power system with the overall mission and user in a way that new benefits through superior use are achieved. Autonomy in the architecture simplifies operation from the user perspective by freeing the user of continuous supervision.

Classifications of Vehicles and Architectural Considerations

Vehicles refer to a broad range of systems and can be classified in several ways. In figure 4.5, vehicles are classified by type within user areas and a few examples of each are given. Classification in this manner allows the architecture to consider the mission of the vehicle as well as its design.

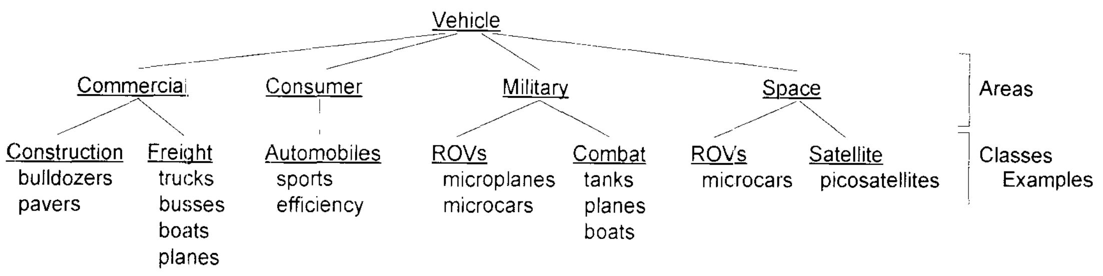


Figure 4.5. Vehicle types and examples.

The user area determines the mission parameters by which goals are defined. Commercial vehicles are typically operated by businesses to perform formal functions. For example, bulldozers for construction and busses for mass transit. Consumer vehicles are operated by individuals in the general public for personal transportation or sporting / recreational use. Military vehicles are also operated professionally similar to commercial vehicles but used for reconnaissance or combat missions instead. Space applications are operated with limited access and utilized for exploration or other technological support missions. In all of these missions, reliability, efficiency, autonomy, and flexibility are important but have different meanings. These are listed below.

- Reliability is a measure of how dependable a vehicle is at performing its mission.
    - Commercial reliability allows the business to utilize the vehicle for profit over a long life span for good return on investment.
    - Consumer reliability allows the consumer to utilize the vehicle at minimal cost since this is a major consumer motivation.
    - Military reliability allows the vehicle to perform its mission accurately in diverse and hostile environments and tolerate failure since the mission is critical to human life and freedom.

    - Space reliability allows the vehicle to function autonomously and tolerate failure since communications is limited and repairs are difficult or impossible in space.

- Efficiency is a measure of how much work can be done versus the resources consumed to perform that work. In all cases, this intends to reduce the consumption of power for a given task.
    - Commercial and consumer efficiency reduces operating costs and can offset fossil-fuel reliance in many cases.
    - Military efficiency extends the time of operation so that recharging / refueling is minimized since these resources may be limited in the combat theater.
    - Space efficiency extends, and often enables, the ability of mission tasks to be performed given the small size constraints and limited power available from solar photovoltaic cells and batteries.
- Autonomy is the ability of the power management system to operate itself and make decisions in the absence of human interaction.
    - Commercial and consumer autonomy frees the operator up from performing the more mundane tasks of power management and allows them to focus on their direct mission.
    - Military and space autonomy allows the vehicle to continue performing its mission when communications are lost or make real-time decisions faster than human operators can respond in critical situations.
- Flexibility applies equally to all vehicular classes and refers to the ease with which the architecture is created or modified to handle changing mission parameters. The

modularity of the architecture and rule-based approach allows this flexibility since this is a white-box method and can be changed incrementally or adapted to different missions by adding or removing the software device object components.

Safety in Vehicular Systems

Safety considerations for vehicular systems are of similar importance to those previously mentioned in industrial control. Vehicular systems are mobile which presents a special hazard to the environment around them in the form of collisions. Humans are also occupants of most vehicle classes and must be protected as well. Although these safety systems usually fall outside the domain of the power management system, there are some permissive-based actions for the power management system to handle. For example: shut down in a catastrophic event; warnings of impending failures or power depletions; and emergency backup power management.

Vehicular Control Systems Environments

Vehicular systems typically employ an embedded control model with specialized software libraries and a C compiler. As such, they have been limited in memory and processing speed compared with traditional computers; however, these limitations are quickly disappearing and complex software designs with large data structures are now possible. These embedded systems are real-time systems and employ real-time networks to ensure critical process data delivery. In automobiles, the CAN standard for a control area network is often utilized in one version or another [Yongqin, 2006]. There has also been much work in control software development environments for automobiles

[Beaumont, 1999] [Miller, 1998] [Smith, 1999]. Simulation of systems has been done in off-the-shelf applications such as Matlab's Simulink®. Therefore, software development is not outside the realm of typical development environments in the same way that industrial control systems traditionally have been.

Example Application of Nitrogen Oxide Reduction for a Coal-fired Boiler

Plant description

The author has developed and installed four applications based on the Pegasus NeuSIGHT® neural network optimization software at the Cinergy Gallagher Generating Station in New Albany, Indiana. The four applications were developed for four similar coal-fired steam-generating boilers for the purpose of reduced nitrogen oxide emissions. A Metso Automation Max1/Max1000++ distributed control system provides data acquisition and boiler control. Each unit is comprised of an Allis Chalmers steam turbine powered by a Riley Stoker wall-fired 18-burner boiler. Steam is delivered at 1,000,000lbs/hr at 1800psi at 1005F to produce 150MW by each generator at full load. Details of the Pegasus neural network application in general are discussed in *Chapter II* and process data flow is as illustrated in figure 2.1.

Application Architecture

The process variables to be controlled by the application, i.e. controllables or outputs, were chosen to fit within the existing control scheme with maximum nitrogen oxide influence and minimum operations impact. Inputs to the application included approximately 120 of the most significant of the existing field sensor control system inputs, since more inputs typically results in a more complete classification for neural networks. These were largely determinable by expert knowledge of the plant as well as some preliminary analysis of variable relationships with nitrogen oxide production. The controllables are biased by the application so that their influence is added to the current operator setpoint similar to (4.3). These controllables are listed in table 4.1. The basic architecture of the application is given in figure 4.6.

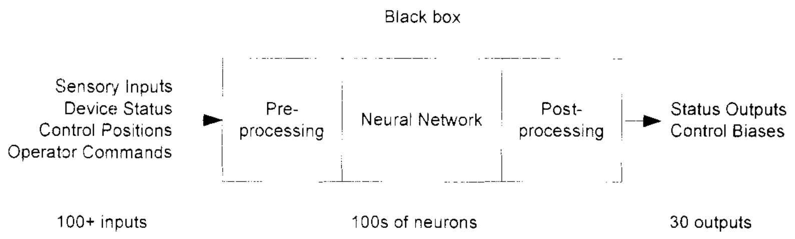


Figure 4.6. Neural network application architecture.

Process variable bias	Description
Excess air setpoint	Secondary air – 1 bias
Wind box–furnace differential pressure	Secondary air – 1 bias
Burner shrouds	Secondary air – 18 shroud biases
Overfire air dampers	Secondary air – 4 damper biases
Coal mill outlet temperatures	Primary air – 3 temp biases for 3 mills
Coal feeder speeds	Fuel – 3 speed biases for 3 feeders

Table 4.1. Application controllables.

$$Shroud_{SP} = Shroud_{Oper} + Shroud_{Bias} \quad (4.3)$$

Where the $Shroud_{SP}$ is the setpoint used by the control system to determine a burner shroud position, $Shroud_{Oper}$ is the operator-entered setpoint for the shroud, and $Shroud_{Bias}$ is the bias added by the neural network application.

The pre-processing and post-processing modules are coded directly in a conditional function block format. The neural network is trained by a typical back-propagation algorithm using a set of training data from the historical process data of the control system. With such a large number of inputs and outputs, a large quantity of training data is required. This data was manually validated to reduce noise and ensure that the domain of the training set properly spanned the operating state space. While some automation could be employed, e.g. Perl scripts to verify and filter large sets of process data patterns, this was still a time consuming task.

Applying the Metrics to the Application

The metrics of portability, scalability, simplicity, and autonomy developed in *Chapter III* are applied to the neural network application here. Comparison with other applications and qualitative discussion with respect to the architecture is presented in *Chapter V.*

Portability

When assessing portability, it is considered that the intention was to port the application to the remaining three generating units once developed for the first unit. Had this not been the case, portability would have trivially been zero since this was a custom application with specific inputs and outputs and a specifically trained neural network, resulting in a black-box approach. The three remaining were of identical design and even of similar age, however, it was made obvious that these units had aged unequally resulting in a plant with four similarly configured, yet individual, units. The pre-processing and post-processing modules could be moved with very little modification and these were assessed a portability metric of 1. The neural network required complete retraining including new training data collection and therefore this module was assessed a portability metric of 0. The neural network consumed approximately 80% of the effort. This resulted in a low application portability of 20% as demonstrated in table 4.2.

Module / Task	**Effort *w***	**Portability *p***
Pre-processing logic	10%	1
Post-processing logic	10%	1
Neural network Building training set Training and building neural network Testing	80%	0
Total Application from (3.1)	100%	20%

Table 4.2. Neural network application portability.

Scalability

When assessing scalability, it should be considered that the application was not designed to be scalable. Being dominated by a monolithic neural network implied that nearly any level of scope change would require retraining, which was demonstrated to be 80% of the effort. Specifically, the level of scope change desired from the application after initial deployment was to add the goal of opacity reduction to combat this new problem and provide an influence entry point for management. Evaluating scalability from the perspective of adding this new scope results in the following assessment in table 4.3.

Module	**Effort *w***	**Scalability *s***
Opacity goal		
Pre-processing logic	10%	3
Post-processing logic	10%	3
Neural network	80%	0
Scalability for opacity reduction from (3.2)	Opacity	20%
Management goal		
Pre-processing logic	10%	2
Post-processing logic	10%	3
Neural network	80%	0
Scalability for new management user from (3.2)	Management	16.7%
Scalability for both application enhancements	Total	18.3%

Table 4.3. Neural network application scalability.

As would be expected of a monolithic neural network application, the scalability is low similar to the portability. To achieve the enhancement of opacity, the same inputs and controllables were used which resulted in unchanged pre-processing and post-processing modules, thus the scalability factor of 3 for these modules. The neural network was retrained with the new goals of both nitrogen oxide and opacity reduction, thus the scalability factor of 0. Management's ability to observe the operation of the neural network application was already available through the existing control system. Since any direct control over the generating unit would be reserved for operations staff, the only significant influence accessible to management staff would be prioritization of the optimization goals. In this case, these goals were nitrogen oxide and opacity reductions. Prioritization of these goals was obtained by adding two additional inputs to the application, the nitrogen oxide priority and the opacity priority. These changed the balance in the neural network of achieving these goals. The resulting scalability factor for pre-processing was assessed at 2 for the modifications needed to handle the additional inputs. The post-processing scalability factor was assessed at 3 since the controllables were unchanged. The neural network again had to be completely retrained with a new data set to incorporate the new inputs, resulting in a scalability assessment of 0.

Simplicity

When assessing simplicity, difficulty is the measured metric and simplicity is inferred by comparison with other applications in *Chapter V*. Both the difficulty in maintaining the application and the software complexity of the application are considered. The pre-processing module consisted of logic to verify the ranges of the approximately 120 application inputs and the post-processing module included logic to calculate the 30 biases from the neural network outputs. Specifically, this involved a conditional statement for each input and output performed by an IF-THEN statement in addition to a bias calculation equation for each output. The neural network module consisted of a proprietary neural network engine with the configurable parameters: number of inputs; number of outputs; neuron activation function; and other training parameters. The analysis is summarized in table 4.4.

Module	**Readability** C_R	**Complexity** C_C	**fanin, fanout**	**Difficulty (3.3) (3.4)**
Pre-processing logic	1	121	1,1	121
Post-processing logic	2	31	1,1	62
Neural network	3	1	120,30	38880000
Application Difficulty				38880183

Table 4.4. Neural network application difficulty.

As expected of a monolithic neural network application, the neural network module dominated the difficulty. The readability factor for the pre-processing logic was assessed at 1 since these were simple conditional statements checking the limits of the inputs. The readability factor for the post-processing module was assessed at 2 since these included both conditional statements and an equation to calculate the bias as in (4.3). The readability factor for the neural network was assessed at 3 since it was virtually unreadable as a black box module. The complexity factor was assessed at 121 for the pre-processing module as 120 conditionals plus 1. The complexity factor for the post-processing module was assessed at 31 being the 30 conditionals plus 1. The complexity factor for the neural network module was assessed at 1 since the neural network algorithm was an external pre-defined function. The *fanin* and *fanout* of both the pre-processing and post-processing modules were assessed at 1,1 because each conditional statement was a separate component of the module with one input and one output. It did not make since to assess these based on the number of inputs and outputs since this would have resulted in an inaccurate representation of complexity. The *fanin* and *fanout* of the neural network was assessed at 120,30 since this was the number of inputs and outputs associated with this single function.

Autonomy

When assessing autonomy, the measures of automation, self-preservation, strategy, and coordination are calculated. With respect to automation, the scope of the neural network application a method for nitrogen oxide reduction. Therefore, some parts of the generating unit control system were intentionally not automated or directly influenced by the application and are thus not applicable (N/A) to an automation metric. Table 4.5 lists the subsystems in the control system and the nitrogen oxide reduction scope of the application.

Autonomy – Automation

Subsystem	**Controllable element**	**Automated**
Boiler combustion control		
Primary air	Mill suction dampers x 3 Mill exhaust dampers x 3 Mill barometric dampers x 3 Mill tempering air dampers x 3	Yes No No No
Fuel	Coal feeder speeds x 3	Yes
Secondary air	Burner shrouds x 18 Overfire air dampers x 4 Excess air (O_2) x 1 Furnace-furnace diff pressure x 1 Forced draft fans x 2 Induced draft fans x 2	Yes Yes Yes Yes No No
Water	No optimization	N/A
Turbine control	No optimization	N/A
Auxiliary systems control	No optimization	N/A
Applicable controllables	43	30

Table 4.5. Neural network application scope.

The controllables list in table 4.5 are what the operator would be required to manually tune if given the goal of reducing nitrogen oxide. Each system that is listed as automated has its setpoint biased by the application and therefore the operator is relieved from manually tuning that controllable. To determine an automation metric, we determine the percentage of controllables automated by the total number of controllables, in this case 30/43 or 70% resulting in $A_I = 2$. The induced and forced draft fans were not automated since these were already indirectly influenced by automation of furnace-furnace differential pressure and excess air respectively. The remaining mill dampers were not automated since these were either not expected to return enough benefit to justify the effort or safety considerations in mill operation.

Autonomy – Self-preservation

The application was designed to only minimize nitrogen oxide emissions, and later opacity. As such, the handling of trouble conditions was not built into the application. In fact, this application, as typical of most emissions-centric monolithic neural network applications, was designed to suspend its operation at the first sign of trouble from the control system, reverting primary control back to the operator. Therefore, self-preservation was assessed at $A_P = 0$.

Autonomy – Strategies

The strategies employed by the application were the minimization of nitrogen oxide and opacity emissions. While this represents two goals, these goals are inter-related in that minimizing nitrogen oxide emissions resulted increased opacity, thus requiring the minimization of opacity to be added as a complementary goal. When assessing strategy from table 3.5, it was decided that a value of 1 for a single was insufficient since two goals were achieved, however, a value of 2 was too high since these were complementary goals and not mutually independent. Therefore, the strategy metric was assessed at A_S = 1.5, the midpoint between these levels.

Autonomy – Coordination

When assessing coordination, it is considered that the application was not designed to be coordinated with other applications or multiple users. At the plant, there would be four peer applications for the four generating units, however, these units were still operated independently and were purposefully not linked together as part of the existing corporate strategy. In the future, linking these systems as a generating fleet would be a valuable consideration. This is demonstrated in the other implementations in this chapter and discussed further in *Chapter V*. For this application, the coordination metric was assessed at $A_C = 0$, since there was no coordination beyond that level.

A vector magnitude of these metrics by (3.5) given $A_I = 2$, $A_P = 0$, $A_S = 1.5$, and $A_C = 0$ results in an overall autonomy metric of $A = 2.5$. This is plotted in figure 4.7.

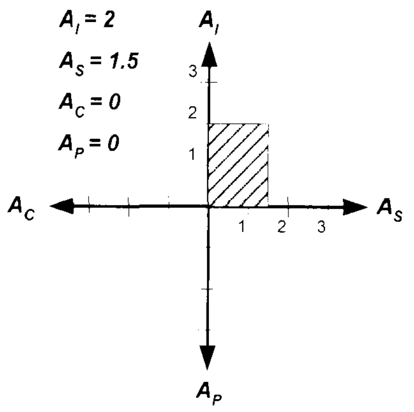


Figure 4.7. Neural network application autonomy.

Results and Closing Remarks for the Application

In the domain of power generating stations, the power management system must also encompass more than just efficient power production. Environmental emissions and impact are also key criteria for these applications. The application achieved good results of approximately 20% reduction in nitrogen oxide emissions by a purely software approach. Opacity was, however, increased and this needed to be remedied for local political reasons. This required significant re-training to correct. Furthermore, now that increased process data was available, management wanted an interface to the new system. This interface through the existing control system provided operational status and allowed management to prioritize the two goals of nitrogen oxide and opacity reduction. This, again, required significant re-training. Since the generating units had aged unequally, it was found that a new neural network was required for each unit. The fact that any horizontal movement or increase in scope required most of the development effort to be redone resulted in the low portability and scalability metrics quantified above.

As expected of a monolithic neural network, the application was difficult to understand and resulted in a black-box approach. This is apparent from the high difficulty, and thus low simplicity, metric. This made it difficult to gain acceptance since the behavior of the application could not be predicted during testing or operation. Plant personnel were apprehensive when the question of *"What is the application going to do next?"* could not be answered definitively. Some level of autonomy was achieved with a good measure of operator actions automated by the application and the complex strategy of emissions reduction achieved as well. There was, however, no coordination or self-preservation employed and this would have enhanced the application.

These limitations were mostly the result of the monolithic neural network. The author's experience in this implementation served as motivation for a better way. It was determined that the criteria for the software metrics presented in *Chapter III* would result in an architecture that was: portable, to reduce effort; scalable, to provide room for enhancement; simple, to gain acceptance and again reduce effort; and autonomous, for better decision-making and coordination. A rule-based software agent approach was selected based on various research efforts, discussed in *Chapter II*, to implement a white-box solution to these problems. This approach is studied in the following implementations.

Example Application for a Hydro-generating Plant

The idea of a software agent suggested in the previous neural network application is pursued here for a river-based hydro-generating station. Portions of this work have been accepted for publication in *IEEE Transactions on Control Systems Technology* [Foreman, 2008]. The software agent will utilize a rule-based system for ease of integration. An enhancement to facilitate the constraints of river level and flow by an external user as well as the goals of corporate dispatching is then developed through the existing SCADA system. Finally, expansion to coordinate multiple hydro units at a single location is presented.

Plant Description

The plant is the Markland Hydro Generation Facility owned by Duke Energy operating on the Ohio River near Markland, Indiana USA. The plant consists of three axial-flow Kaplan-turbine-generating units of approximately 25MW in size and similar configuration. The turbines run at a constant 64.3rpm when synchronized with the power grid. The turbines are controlled by a Woodward Governor 505H control system. The plant utilizes the General Electric Fanuc iFix© supervisory control and data acquisition (SCADA) system as the control system human machine interface (HMI) and data archive. The plant coexists with the Markland Dam operated by the United States Army Corps of Engineers to accommodate river traffic and maintain a set river level. A single hydro unit is illustrated in figure 4.8 adapted from [Paul, 1996].

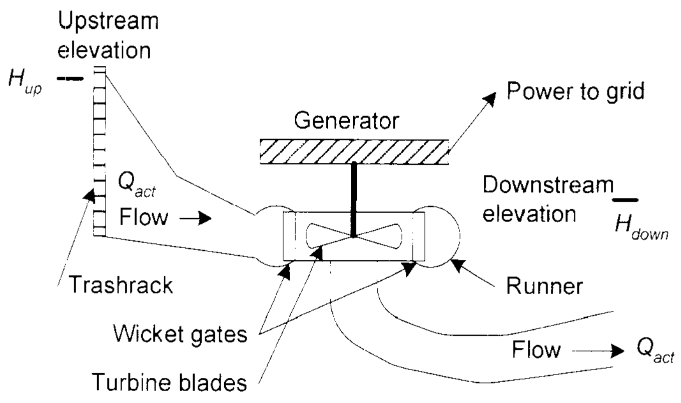


Figure 4.8. Typical hydro unit with primary variables.

Existing Control Scheme

The variables for control actuation are wicket gate position and turbine runner blade position. The wicket gate position refers to the aperture size for river water entry into the turbine and is the main control variable for the unit's flow rate. The turbine blade position refers to the pitch of the blades from horizontal. The blade position is used to extend the efficiency of the turbine at higher gate positions since power is developed by the reaction of water pressure against the turbine runner blades [Paul, 1996]. Control of the wicket gate position *GP* and the resulting unit flow rate is accomplished by a typical PID loop. Control of the turbine blade position *BP* is by a software cam. A software cam is modeled by a virtual 3-dimensional surface where independent variables *X* and *Y* are mapped to a dependent variable *Z*. In this case, *X* and *Y* refer to gate position and net head while *Z* refers to the blade position determined from these inputs. This control scheme is illustrated in figure 4.9.

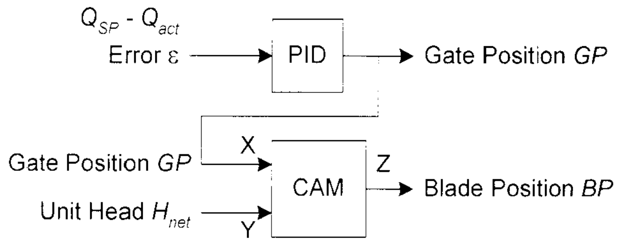


Figure 4.9. Existing control scheme.

Integration of the Software Agent

A single software agent is then added to the control scheme of figure 4.9 within the existing control software. This agent will influence the control action for the variables *GP* and *BP*. This incorporation is illustrated in figure 4.10.

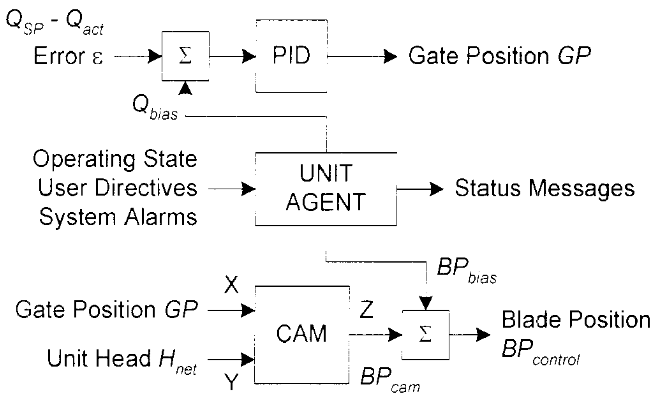


Figure 4.10. Individual unit integration.

The agent includes a rule-based expert system module for the optimization engine, to be discussed in the following section. A *bias calculator* module is defined to calculate the biases to be added to the control scheme as illustrated in figure 4.10. This module performs the same functions as described in the post-processing module of the previous neural network application. Biases are calculated as in (4.3) and conditionals are utilized to ensure that user-defined boundary conditions are not exceeded. A *message handler* receives directives from outside users or other unit agents and broadcasts the status of this agent. This is accomplished by using the built-in process points of the existing control system. The software agent writes to a process point for its status and other agents and users read this point. Likewise, other agents and users have their respective status process points they write to for this agent to read. This results in a trivial definition of the message handler and makes use of the existing process point data structure for secure and reliable communications. Figure 4.11 illustrates these modules of a single agent.

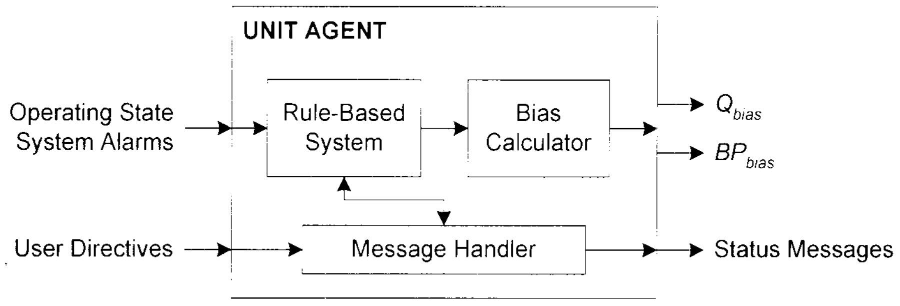


Figure 4.11. Modules in the software agent.

Use Cases of the Application

There are two outside users in addition to the local unit operator that need to influence the unit through the agent, the Army Corps of Engineers and the corporate dispatching office. The Corps is tasked with maintaining the upstream river elevation within a one-foot tolerance of 455ft above sea level and locking river traffic through the dam. Once the corps determines the river flow requirements and subtracts the locking requirements, the resulting value for available flow is manually reported to the hydro plant via a SCADA interface. The corporate dispatching office is tasked with dispatching generating units in the corporate fleet to meet customer demand and maintain stability of the power delivery grid. The corporate dispatching office occasionally needs to adjust power delivery for grid stability issues and would also benefit from the unit status updates the agent could provide. Figure 4.12 illustrates a use case diagram for the local operator, Corps, and dispatch users as they interact with the unit agent.

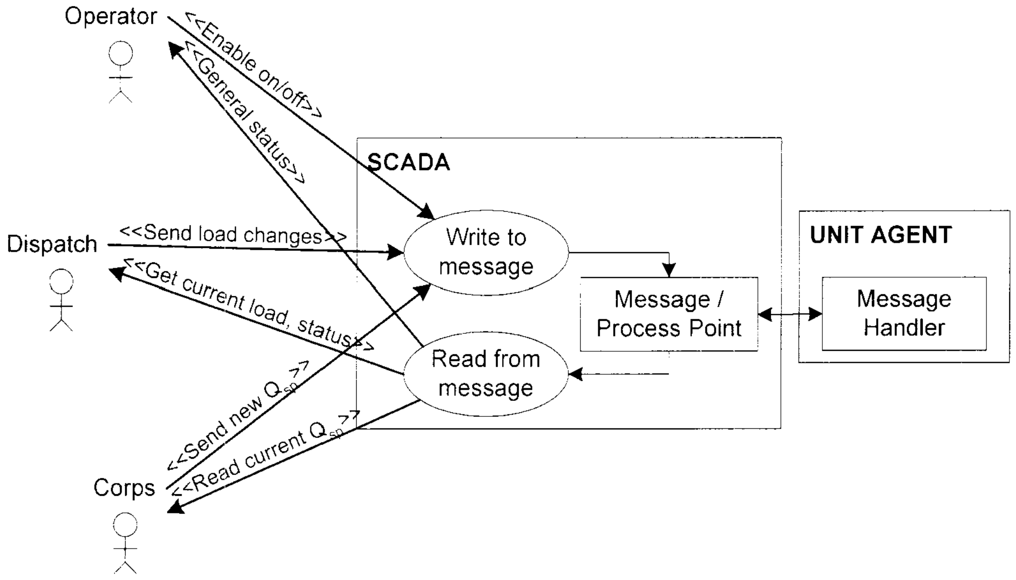


Figure 4.12. Use case diagram for a single agent.

The single agent, and therefore single unit, architecture is expanded to include multiple units to accommodate the three units at the plant. The expanded use case is illustrated in figure 4.13.

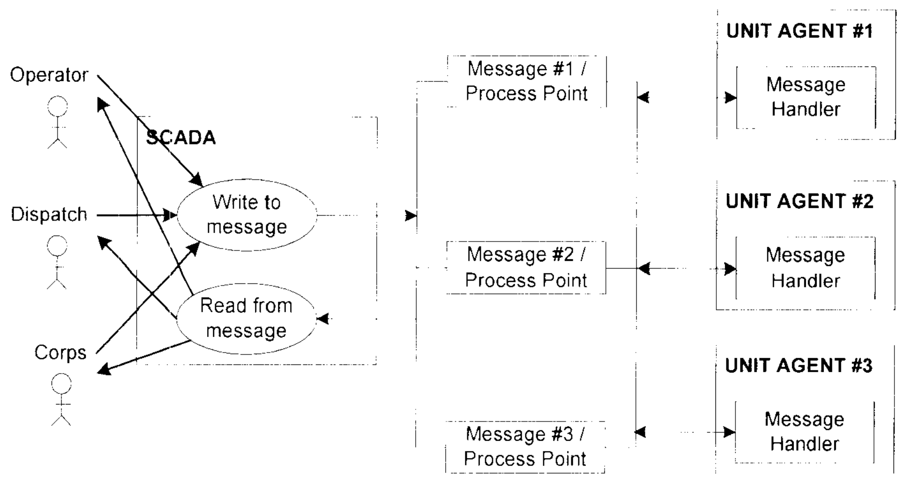


Figure 4.13. Multi-agent use case diagram.

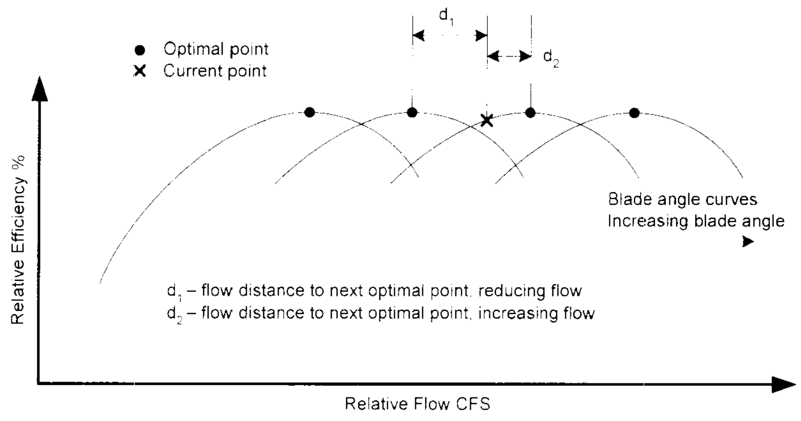
Each unit agent attempts the local optimal production of power for its unit while coordinating the directives of other users. With the addition of the other unit agents as users, the status of other generating units is now able to influence each unit agent.

Development of the Rule Sets

The rule-based expert system is built from user-defined rules governing the scope of the application. Rules were developed for biasing for optimal generating efficiency, handling of trouble conditions, and coordination with other users and agents. Specifically, the other users are the Army Corps of Engineers at the Markland Dam and the corporate dispatching office. The other agents are the other generating units at the plant.

Optimal point control

The hydro-generating unit is essentially a water pump operating in reverse such that river water flow turns the turbine blades and attached generator, thus producing electricity. As such, the hydro-generating unit is characterized by a pump efficiency curve that defines optimal operating points for efficiency. This is illustrated in figure 4.14.



● Optimal point
× Current point
d_1
d_2
Blade angle curves
Increasing blade angle
d_1 – flow distance to next optimal point, reducing flow
d_2 – flow distance to next optimal point, increasing flow
Relative Efficiency %
Relative Flow CFS

Figure 4.14. Optimal points of operation.

In figure 4.14, there are multiple curves because our hydro-generating unit has variable-angle turbine blades as described previously. This curve is based on a constant head. As the blade angle changes, the characteristic curve changes and extends the efficient operating region of the turbine. The optimal point for each blade angle is depicted on the curves. These points are typically determined by index testing and can be stored in a database in the format (head, flow, efficiency) for use by the application. The distance in flow from the current operating point to the nearest optimal points can be determine as depicted in figure 4.14. Rules in the expert system can detect this and increase or decrease flow as necessary to achieve optimal operation. The below pseudo code demonstrates this.

```
; given the current head find the optimal flow points below and above
; the current point and the distances in flow to these
```

```
Mode.single = TRUE ; for a single unit, no coordination
; PlantFlow.allocated is the flow allowed by the Corps
PlantFlow.available = PlantFlow.allocated - PlantFlow.setpoint
; if a single unit then just take the additional flow if you can
; more flow always equals more generation even if not optimal
IF Mode.single THEN
        IF PlantFlow.available THEN ;
                Qbias = PlantFlow.available ; see figure 4.10
        ENDIF
ENDIF
; if a multi unit situation then we need to distribute the flow properly
Mode.single = FALSE
IF NOT Mode.single THEN
; determine d1 and d2 from optimal point database as in figure 4.14
        d1 = flow.setpoint - flow.optimal.below
        d2 = flow.setpoint - flow.optimal.above
; others.efficiency list of other units efficiencies from message handler
        efficiency.below = database(head, flow.optimal.below)
        efficiency.above = database(head, flow.optimal.above)
; PlantFlow.minimum is user-defined minimal amount to change flow
; if this unit is most efficient then take flow
        IF PlantFlow.available > PlantFlow.minimum THEN
                IF efficiency.above > MAX(others.efficiency) THEN
                        IF PlantFlow.available > d2 THEN
                                Qbias = d2 ; go up to next highest optimal point
                        ELSE ; or at least as close as you can
                                Qbias = PlantFlow.available
                        ENDIF
                ENDIF
        ENDIF
; if this unit is least efficient then give up flow
; do this always to make flow available to more efficient units
        IF efficiency.below < MIN(others.efficiency) THEN
                Qbias = -d1 ; go down to next lowest optimal point
        ENDIF
ENDIF
```

```
; C. = 8
```

In the pseudo code above, plant flow that is available from the Corps is allocated to the most efficient unit to get that unit to its next optimal point. Flow is also taken from the least efficient unit and reallocated to the most efficient unit to increase overall plant efficiency. As this redistribution is continued, a steady state condition is reached (or a user-defined terminating condition) resulting in a more efficient flow distribution and therefore more plant generation. The *Results* section gives a simulated example of this. In the single unit case, this is trivial since you always want to use all the flow available. The *database* function looks up the missing parameter from the optimal point database on the given parameters, e.g. if head and flow are given then efficiency is returned and so forth.

Startup and shutdown

Another important application at plants with multiple units is the determination of how many units to run and the order of start up and shutdown of individual units. In general, we want to give priority to units that are more efficient and also those with less cumulative run time to result in uniform machine wear. The number of units to run is typically based on flow or on the product of flow and head since generator temperatures or cavitation usually bound the upper limit of a hydro-generating unit. The Hill curve in figure 4.15 illustrates this. The rule sets below would reside in each unit's software agent and demonstrate how many units to run, when to cycle on, when to cycle off, and the start up order.

```
; determine number of units to run based on flow
NumberUnits.online = < number of units online >
IF PlantFlow.available < 2000cfs THEN
```

```
        NumberUnits.desired = 0
ELSE IF PLantFlow.available < 7000cfs THEN
        NumberUnits.desired = 1
ELSE IF PLantFlow.available < 14000cfs THEN
        NumberUnits.desired = 2
ELSE
        NumberUnits.desired = 3
ENDIF
; if we want another unit online then pick the one with the lowest run time
IF NumberUnits.desired > NumberUnits.online THEN
        IF NOT Permitted_to_run THEN
                IF runtime < MIN(others.runtime) OR NumberUnits.desired = 3 THEN
                        Permitted_to_run = TRUE
                ENDIF
        ENDIF
ENDIF
; when to cycle on, once permitted to run
; this makes sure the new unit can make it to the first optimal point
IF Permitted_to_run and NOT Online THEN
        IF PlantFlow.available > OptimalPoints.flows.minimum THEN
                Go_online()
        ENDIF
ENDIF
; when to cycle off
; this takes the least efficient unit offline first
IF Online and NumberUnits.desired < NumberUnits.online THEN
        IF efficiency < MIN(others.efficiency) THEN
                Go_offline()
                Permitted_to_run = FALSE
        ENDIF
ENDIF
; C = 11
```

Trouble conditions

The handling of trouble conditions is another area of optimization where significant gains can be achieved. While there are many ancillary alarm points in a hydro-generating unit, the trouble conditions that can result in the biggest gains are stator temperature excursions, vibration conditions, and cavitation conditions. In figure 4.15 is depicted the Hill curves for a typical hydro-generating unit. These curves are defined by (head,flow) data points that are determined from index testing and operating history. The central region illustrates the normal operating region of the unit, shown by point X_1. In the top right corner is the generator limit line. This line is not typically expressed explicitly because increasing generation beyond the generator's capability results in excessive heating of the stator. Therefore, the generator limit is implied when stator temperatures go above their preset limit. The lines of maximum and minimum head and gate position are self-explanatory preset limits of the unit. Cavitation is the event of bubbles forming by transition to the vapor phase when water enters an area of low pressure and then subsequently collapsing when these bubbles reenter an area of higher pressure. Cavitation is thus a sonic and vibrational issue that damages the turbine blades. The areas of operation where cavitation has been determined to occur are depicted on figure 4.15. Vibration conditions can occur throughout the operating region. A Bently Nevada proximity probe system is used to measure the turbine vibration in the 1X, 1Y, 4X, and 4Y modes. Two proximity probes placed 90 degrees apart in the turbine shaft bearings represent the X and Y modes. The 1 and 4 notation refers to vibration measured at 1 times and 4 times the rotational speed of the turbine respectively. Cavitation will also result in vibration. Therefore, vibrations detected when the unit is operating near the cavitation areas, shown by X_2 and X_3, are judged to be cavitation.

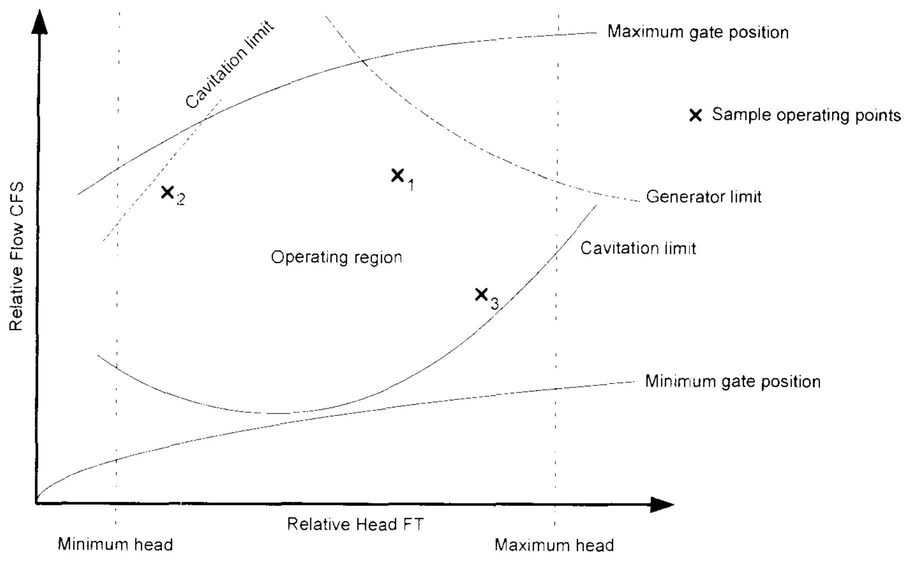


Figure 4.15. Hill curve of operating space and limits.

Adapted from [Paul, 1996].

The Hill curve helps classify the type of trouble condition currently being experienced and the probable corrective action to take. Head is always fixed for a given scenario since this is a disturbance variable and not controllable. Vibration at low head and high flow is likely to be cavitation and would be corrected by reducing flow to the unit. Vibration and high head and low flow would also be due to cavitation but in this case, flow to the unit should be increased. High stator temperatures would result from both high head and high flow and therefore flow should again be reduced. If flow needs to be increased or decreased for this unit, the rule sets above for distributing flow should move the other units to compensate. This is depicted in the rule sets below.

```
; is there cavitation
; distance is the distance between points
; min is a minimum distance to classify constant
IF vibration THEN
```

```
    IF distance(OperatingPoint.current,OperatingPoint.cavitation) < min THEN
        cavitation = TRUE ; vibration near cavitation curve
    ELSE
        cavitation = FALSE ; vibration not near cavitation curve
ENDIF
; now handle which type of cavitation
IF cavitation THEN
    IF head < max_head/2 AND flow > max_flow/2 THEN
        ; we are on the top curve and need less flow
        ; reduce our efficiency to get flow taken away and
        ; bias gate position down
        Qbias = Qbias - 500cfs ; do in 500cfs steps
        efficiency = -1 ; negative which should be below plant minimum
    ELSE
        ; we are on the bottom curve and need more flow
        ; we would bias gate position up and
        ; increase our efficiency to have a flow priority
        Qbias = Qbias + 500cfs ; do in 500cfs steps
        efficiency = 2; 200% which should be above the plant maximum
    ENDIF
ENDIF
; for vibration away from a cavitation point,
; it is probably best to just reduce flow and thus reduce machine load
IF vibration and NOT cavitation = HI THEN
    Qbias = Qbias - 500cfs ; do in 500cfs steps
    efficiency = -1 ; negative which should be below plant minimum
ENDIF
; for stator temps, we always need to reduce flow which reduces generator load
IF stator.temp = HI THEN
    Qbias = Qbias - 500cfs ; do in 500cfs steps
    efficiency = -1 ; negative which should be below plant minimum
ENDIF
; C = 7
```

Trouble conditions and flow allocations should be addressed by biasing the gate position or the unit flow setpoint directly. Biasing the blade position should be reserved for optimizing the generating efficiency at a specific steady-state operating point.

Rule scheduling

Given that the hydro-generating unit is a physical process, it is necessary to control the timing of rule execution. This is accomplished by enclosing the respective rule set within code that schedules when the rule set gets evaluated. Typically this *TimeDelay* might be 3-5 minutes for each evaluation step, or quicker for cavitation and vibration issues since their response time is quicker, e.g. 5-10sec. The below pseudo code demonstrates this.

```
; TimeDelay is a constant wait time between evaluating this rule set
; timer is the last time rule set was evaluated
IF (Clock - timer) > TimeDelay THEN
      timer = Clock ; reset timer to current system clock
      < insert rule set here >
ENDIF
; Ci = 2
```

River trash

A final point of optimizing would be load ejection to clear river trash. The unit has a trash rack that serves as a filter for river trash entering the turbine runner, see figure 4.8. When trash accumulates on this rack, it effectively reduces the net head available to that unit due to the restriction of water flow. This can only be cleared by performing a *load eject* which is a rapid load reduction or total shutdown of the unit. This rapid shutdown causes a backwash wave that clears trash from the trash rack. The unit is then immediately restarted and benefits from the reduced restriction in the form of a higher net head and thus higher generation capability and efficiency. A rule set that would determine when to perform a load eject is given below.

```
; cost of a load eject is the load lost during the ejection time
; loadeject.time is a preset constant
; load is proportional to head times flow, so just use head * flow
; drawdown is the level difference across the trash rack, field measured
loadeject.cost = head.current * flow.setpoint * loadeject.time
load.new = (head.current + drawdown) * flow.setpoint
loadeject.benefit = load.new - load.current
IF loadeject.benefit > loadeject.cost THEN
      Ejectload()
ENDIF
; you may want to delay load ejections during critical demand times
; C. = 2
```

Results

The opportunity for local optimization of an individual unit by its agent is now demonstrated. In figure 4.16, a set of steady-state operating points from historical process data for one of the units is plotted for a particular set of operating conditions. Showing multiple dependent values *Load* for each independent value *Flow* illustrates that there are operating states of varying efficiency. This occurs due to the sub-optimal control of the existing system.

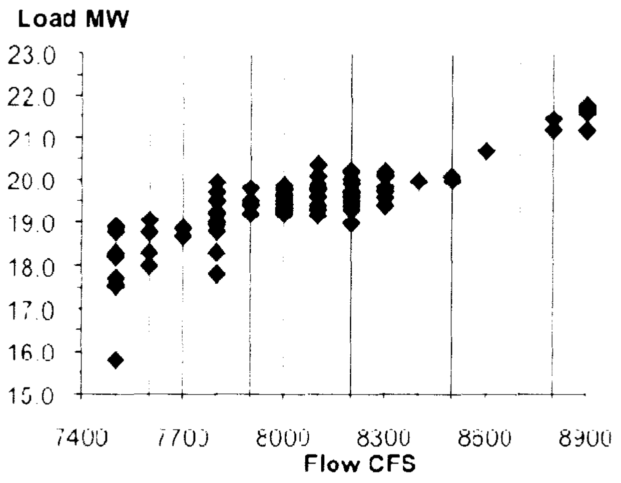


Figure 4.16. Load vs. flow.

Assuming the highest point for a given flow value represents an estimate of potential power generation while the median value represents a typical blade choice, a conservative estimate of at least 0.5MW of increased instantaneous power generation is achieved. Assuming this increase of 0.5MW for 50% of the time for one year, results in 2190MWhr of additional power generation. This offsets 912.5tons of less coal on an annual basis by (4.4). This also reduces annual carbon dioxide release approximately 1670tons by (4.5). This is compared with a typical coal fired generating unit operating with a heat rate (HR) of 10MBTU/MWhr. Fuel is assumed as bituminous coal with a higher heating value (HHV) of 24MBTU/ton and 75% carbon composition.

$$Coal\ Tons/yr = Power * HR / HHV \tag{4.4}$$

$$CO_2\ Tons/yr = Coal\ Tons/yr * 1.83C/CO_2 * 75\% \tag{4.5}$$

Flow redistribution is a second opportunity for efficiency improvement now possible with coordination among multiple units. The three generating units start at an equal flow. Based on their individual efficiencies from (4.6), flow is incrementally redistributed with priority given to the most efficient unit, shown by the solid line and the left-hand scale. Flow is inevitably taken from the least efficient unit until its efficiency drops, resulting in a net loss for the plant. Notice that the maximum load line peaks at a higher value before trailing off as expected from diminishing returns. Therefore, the agent needs to detect this and cease flow redistribution prior to this point. The gain depicted in figure 4.17 represents nearly 1MW of additional power generation for the whole plant, shown by the dotted line and the right-hand scale.

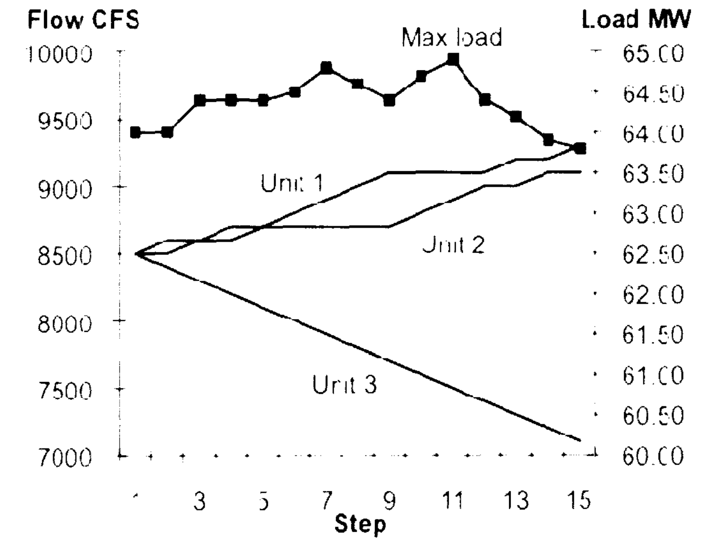
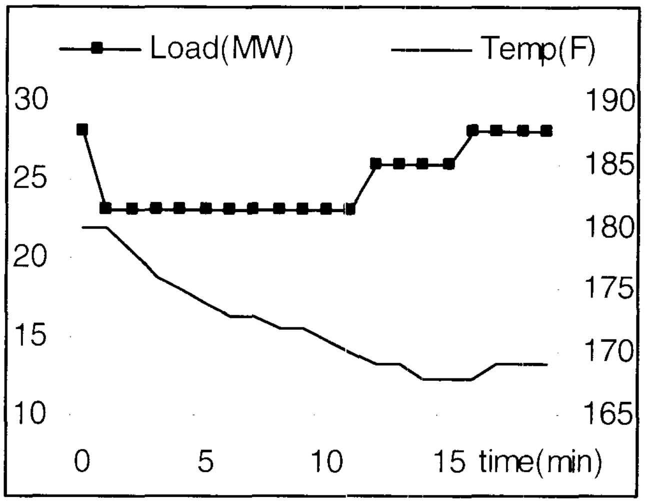


Figure 4.17. Load vs. flow redistribution.

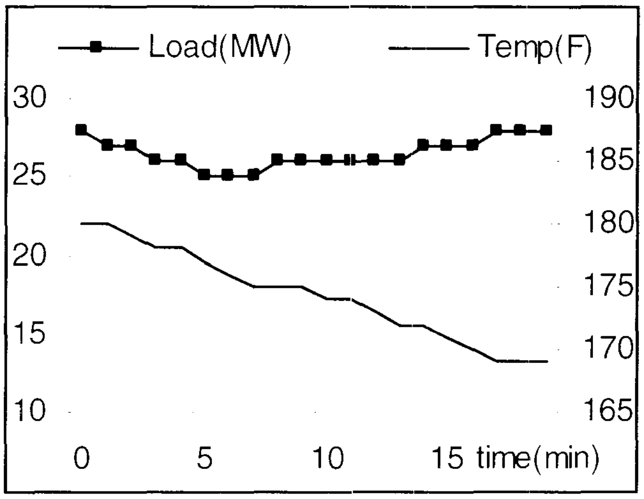
$$P_{avail} = k * \eta * Q_{act} * H_{net} \quad (4.6)$$

Where P_{avail} refers to potential power generation available from the water flow, η is the unit efficiency, Q_{act} is the water flow, H_{net} is the unit's net head, and k is a proportioning constant.

Another opportunity for improvement is available from handling trouble conditions in an automated manner, which relieves the operator from manually handling these conditions. This results in earlier implementation and a better-measured response. Figure 4.18 illustrates a simulated handling of a condition as compared to typical operator response for a single unit.



Operator response.



Optimized response.

Figure 4.18. Response to trouble condition.

In the 200 days of data, there were 71 high stator temperature events (>180degF and >10min) for unit 1, 24 such events for unit 2, and 199 such events for unit 3. Assuming 0.55MWhr gain per event as simulated in figure 4.18, this results in 161.7MWhrs of additional generation.

The similar result of figure 4.18 exists for the trouble conditions of vibration and cavitation utilizing BP_{bias} or Q_{bias}. During vibration and cavitation events, the blade position can be adjusted or the flow biased to alleviate the event. A measured response can be applied rather than a step change load reduction and any flow reductions can be added to the other units.

Applying the Metrics to the Application

The metrics of portability, scalability, simplicity, and autonomy are now applied to the application as in the previous neural network implementation. These are discussed comparatively with the other implementations in *Chapter V*.

Portability

The application is designed to be portable among hydro-generating units. Even units with different configurations should be able to utilize the application with only parameter level changes. The use of a rule-based expert system as the optimization engine affords the ability to look into the application, i.e. it is a white-box approach. As such, the rules are modular and can be modified individually. Table 4.6 evaluates the portability of each module of the application.

Module / Task	**Effort *w***	**Portability *p***
Message handler	10%	1
Bias calculator	10%	1
Expert system rule sets Optimal flow distribution Optimal unit cycling Optimal point operation Trouble conditions Load ejection	80%	1
Total Application from (3.1)	100%	100%

Table 4.6. Software agent application portability.

Table 4.6 assesses the portability metric at 100% for the application. While the expert system module can be further broken down, the rule sets are similarly defined and can therefore be evaluated as a group. In contrast with the previous neural network application, 100% portability is expected when porting the application to identical hydro-generating units since the parameters and rule sets would be the same. Porting the application to other hydro-generating units would require some modification but this would be limited due to the laws of similitude. The laws of similitude are a set of equations that define geometric, kinematic, and dynamic similarity between different hydro-generating units [Paul, 1996]. Using these equations the variables of flow, head, power, etc can be related between two different cases – different cases being different units or different conditions for the same unit. Therefore, portability to differently configured units would still be expected to be high. These equations are listed here for reference from [Paul, 1996].

$$\frac{Q_1}{N_1 D_1^3} = \frac{Q_2}{N_2 D_2^3} \qquad (4.7)$$

$$\frac{P_1}{N_1^3 D_1^5 \rho_1} = \frac{P_2}{N_2^3 D_2^5 \rho_2} \tag{4.8}$$

$$\frac{N_1 D_1}{\sqrt{(g_1 H_1)}} = \frac{N_2 D_2}{\sqrt{(g_2 H_2)}} \tag{4.9}$$

$$\frac{Q_1}{D_1^2 \sqrt{(g_1 H_1)}} = \frac{Q_2}{D_2^2 \sqrt{(g_2 H_2)}} \tag{4.10}$$

$$\frac{P_1}{D_1^2 \rho_1 (g_1 H_1)^{3/2}} = \frac{P_2}{D_2^2 \rho_2 (g_2 H_2)^{3/2}} \tag{4.11}$$

Where Q is the unit flow, H is the net head, P is the power available, D is the intake runner diameter, ρ is the density of water, and g is the local gravitational constant. The subscripts of 1 and 2 denote case 1 and case 2 respectively. Units can be metric or SAE.

Scalability

When assessing scalability of the application, the ability to add new features and scope to the application is measured. The Corps user already influences the application by dictating the allowable flow allocated to the plant. Adding the influence of the corporate dispatch user allows this user to circumvent normal optimal point operation in favor of specific generation output in order to stabilize the power grid when necessary. Also, we consider the addition of a safe fish passage goal, discussed in the closing remarks below. Research has been performed by [Fisher, 1997] and [Railsback, 2003] demonstrating that when fish migration is significant, the blade and gate positions can be biased away from the optimal point to a configuration that improves fish mortality when passing through the turbine blades. Table 4.7 assesses this scalability.

Module	**Effort *w***	**Scalability *s***
Message handler	10%	2
Bias calculator	10%	3
Expert system rule sets	80%	
Optimal flow distribution	10%	3
Optimal unit cycling	10%	3
Optimal point operation	15%	2
Trouble conditions	20%	3
Load ejection	5%	3
Safe fish passage (new)	10%	1
Corporate dispatch (new)	10%	1
Total Application from (3.2)	100%	78%

Table 4.7. Software agent application scalability.

Table 4.7 assesses the scalability metric with respect to the corporate dispatch and safe fish passage enhancements at 78%. In evaluating the modules and rule sets in the table, the benefit of a rule-based system over a monolithic neural network becomes obvious. The message handler needs moderate changes to facilitate the additional process points for corporate dispatch users to interact with the application. Since the same controllables are being used, the bias calculator does not change. Also the rule sets unaffected by the new scope will not change. The optimal point rule set would be modified because both safe fish passage and corporate dispatch would tune the unit away from the optimal point to achieve their goals. Also, new rule sets would need to be defined to handle the both safe fish passage and corporate dispatch, though these would simply add to the existing rule sets.

Simplicity

When assessing simplicity, we are measuring the difficulty and then comparing with other applications to determine relative simplicity, as in the previous neural network application. Table 4.8 determines the difficulty metric for the application for each hydro-generating unit.

Module	Readability C_R	Complexity C_C	fanin, fanout	Difficulty (3.3) (3.4)
Message handler	1	5	2, 1	20
Bias calculator	2	2	2, 2	64
Expert system	1.5	29	10, 2	17400
Application Difficulty				17484

Table 4.8. Software agent application difficulty.

Before implementation of the application, the existing control system included global process points of various parameters and alarm conditions for the units. Therefore, the message handler is implemented by adding a new point for each new agent and user of the system. Given 3 hydro-generating units, a Corps user, and a corporate dispatch user results in 5 new process points. These are simple existing structures and assessed a readability of 1. A complexity of 5 is assessed, one for each process point. The bias calculator is similar to the post-processing module of the previous neural network application and thus, a readability of 2 is assessed. The complexity is assessed at 2 since there are 2 biases, gate position and blade position. In assessing the expert system, it is considered that the pseudo code above is a partial representation of the actual code that would be deployed. The readability is assessed at 1.5 as a compromise between 1 and 2 since the rules are nearly directly readable with some equations. The complexity for each section of pseudo code above was assessed a local complexity at the end of the code section and these were summed to 29. The final result for the expert system is 17400 and each unit of the application is 17484.

Autonomy – Automation

When assessing autonomy, the metrics of automation, self-preservation, strategy, and coordination are measured as in the previous neural network application. The hydro-generating unit was close to fully automated before implementation. However, changing the automated control state to a more optimal state, determining the start up order and unit cycling, and redistributing flow would have been the responsibility of the operator. Biasing the flow setpoint, Q_{bias}, and the blade position would have been how the operator would accomplish these. Since the rule sets in the expert system are designed to completely handle these, automation is assessed at a level 3 indicating that at least 90% of the operator tasks are automated.

Autonomy – Self-preservation

Self-preservation is another feature intentionally designed into the application. With the exception of ancillary support systems and unforeseen problems, the standard trouble conditions of stator temperature excursions, cavitation, and vibration are addressed by the application. Again, since the rule sets in the expert system are designed to completely handle these, self-preservation is assessed at a level 3 indicating that at least 90% of the trouble conditions are handled.

Autonomy – Strategy

When assessing strategy, the application implements the main goal of optimizing efficiency in power generation through optimal point control, flow distribution, and trash load ejection for trash. The application also implements the goals of determining the start up and shut down order of the units and handling trouble conditions. Provisions for safe fish passage and corporate dispatch coordination were discussed but not included in the initial application, thus there are 3 goals but some room for additional goals to be achieved. A strategy metric of 2 is assessed demonstrating both the capability and room for growth.

Autonomy – Coordination

Coordination was another capability designed into the application. Most hydro plants incorporate multiple hydro-generating units that must work together if the most optimal strategies are to be achieved. It is also typical that outside users, such as the Corps, would determine some operating parameters such as river flow allocated to the plant. The ability to enable multiple users to affect operation of the hydro-generating unit and the level of coordination with the other units, through their agent's message handler, results in a coordination metric of 2. A coordination metric of 3 would be reserved if truly intuitive cooperation was achieved.

The autonomy metric for the software agent application is therefore $\sqrt{(3^2 + 3^2 + 2^2 + 2^2)}$ or $\sqrt{26}$. This is depicted in the four-dimensional radar chart of figure 4.19 and are discussed further in the closing remarks.

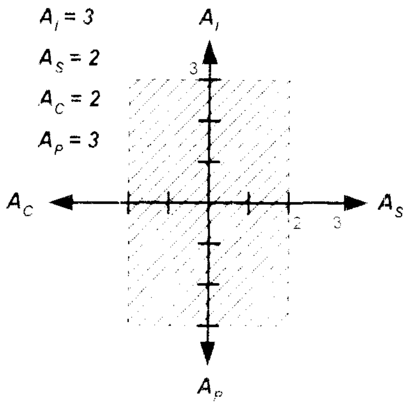


Figure 4.19. Software agent application autonomy metric.

Closing Remarks for the Application

The high metrics for portability and scalability illustrate the potential for software agents and the rule-based approach in adapting to change. The previous neural network application with many inputs and outputs learned the process too well. The particular nuances of each coal-fired boiler prevented that application from porting even to identically designed units. The ability to add new scope was also severely limited since the neural network required complete retraining for any change. The rule-based approach modularizes the optimization engine so that only what needed to scale was changed. The white-box characteristic designed into the application also improves readability, which reduces the difficulty resulting in a simpler design. When testing the application, it is possible to predict how the application behaves, thus reducing apprehension in use.

Autonomy was greatly enhanced though the units were well automated beforehand. Figure 4.19 demonstrates the height at the maximum level of automation and self-preservation. Width is good as moderate coordination and strategy are implemented; yet there is room for growth. These metrics demonstrate that the modular, rule-based approach has several qualitative benefits over the previous neural network design.

When dealing with multiple agents and multiple users, conflict resolution and security become issues to address. With respect to the external users such as the operator, the Corps, and corporate dispatch, it is seen in figure 4.12 how these users have different types of influence over the plant. Therefore, restrictions in the SCADA system control how these users access the control system. Allowing users to only access their process points, e.g. the Corps can only write to the *PlantFlow.available* process point, provides security and resolves conflicts at the user level. At the software agent level, the rule sets determine how conflict is resolved and user inputs prioritized. For example, in the pseudo code above, it was described how new plant river flow would be distributed among the unit agents. Each agent calculates its generating efficiency and reports this to the others. The agents, therefore, know if they are the most efficient and the rules dictate that they take the flow. Since the same rule set is in the other unit agents, they likewise know they are not the most efficient and thus do not take the flow.

The results clearly show that there is free power generation left on the table due to suboptimal gate and blade positions. Furthermore, there are significant qualitative gains in operating strategy by coordinating the enterprise users and multiple units with software agents. Some of the qualitative benefits and further considerations include:

- Through better management and response to trouble conditions, maintenance and downtime are both expected to be reduced. This translates into reduced operating costs and increased production and availability. There are also operating states that dramatically reduce cumulative machine wear under variable conditions, similar to those in this paper, studied by March [March, 2003].
- A major investment both in money and effort is in the Federal Energy Regulatory Commission (FREC) relicensing process. Demonstrating better use of the natural resource of river flow, as well as better plant management to increase generation thus reducing reliance on fossil fuels, provides solid evidence to both FERC and the general public for the case of continued operation.
- It would be mutually beneficial to the Corps and the Markland Hydro plant to automate control of the river level. This would allow the Corps direct control over river level rather than determining a plant flow target and thus indirect control. This simplifies the Corps' duties and provides more accurate calculation of Q_{sp}, which is a critical variable for plant operation. For multiple hydro plants along the same river, level control can be linked between plants.
- As wildlife and environmental concerns become increasingly important, the issue of fish mortality through the turbine blades must be addressed. Research has been done on models for safe fish passage by varying operating conditions that would integrate well into this model. Simple revisions to the rule sets could accomplish this [Fisher, 1997] [Railsback, 2003].

The Architecture Applied to a Power Plant with Hybrid Vehicle Coupling

The architecture of *Chapter III* is now applied to the previous hydro-generating application. This demonstrates how the architecture advances the state of the art by implementing a layered approach that incorporates the benefits of both decentralization and centralization in the same application. Once the hydro-generating application is developed, a similar application to a typical hybrid vehicle will then be developed. Finally, these two applications are linked to achieve a power plant to hybrid vehicle coupling, commonly known as vehicle to grid (V2G) technology. This will demonstrate the enterprise-level and advanced coordination capabilities of the portable, scalable, simple, and autonomous architecture.

Overview

Hybrid vehicles are beginning to replace the previous model of fossil-fuel-only vehicles in personal automotive applications. This new vehicle model includes components for both power generation and storage and a large enough scale to serve most of the needs of a single-family home. It is now becoming possible for these vehicles to be integrated into the power grid at the individual user's residence. Currently, this is for manually charging of the vehicle's power storage system. However, this affords a new opportunity as power utilities could now use the hybrid vehicle remotely to store and generate power on demand for supplemental needs. In many areas, the power grid is taxed near its maximum capability at peak demand times, yet at other times during the same day, the demand is only a fraction of the grid's capability. The repository of hybrid vehicles could smooth out this demand and bring stability to power delivery. The benefits of this would dramatically reduce spot power prices since demand would be managed by controlling supply in this new way. This also creates a strategy to handle the anticipated demand on the power grid by the adoption of such vehicles.

Figure 4.20 provides an overview of how the architecture is applied to the hydro-generating plant, the hybrid vehicle, and their coupling. Each hydro-generating unit will have a software device agent that provides local, decentralized management functions that is linked to the plant's system layer for centralized management functions. Similarly for the hybrid vehicle, each power system device will have its respective software device object that is coordinated by the hybrid vehicle system layer. The system layers of these two applications are then linked for negotiation through the consumer's power meter and via the IP over power lines protocol.

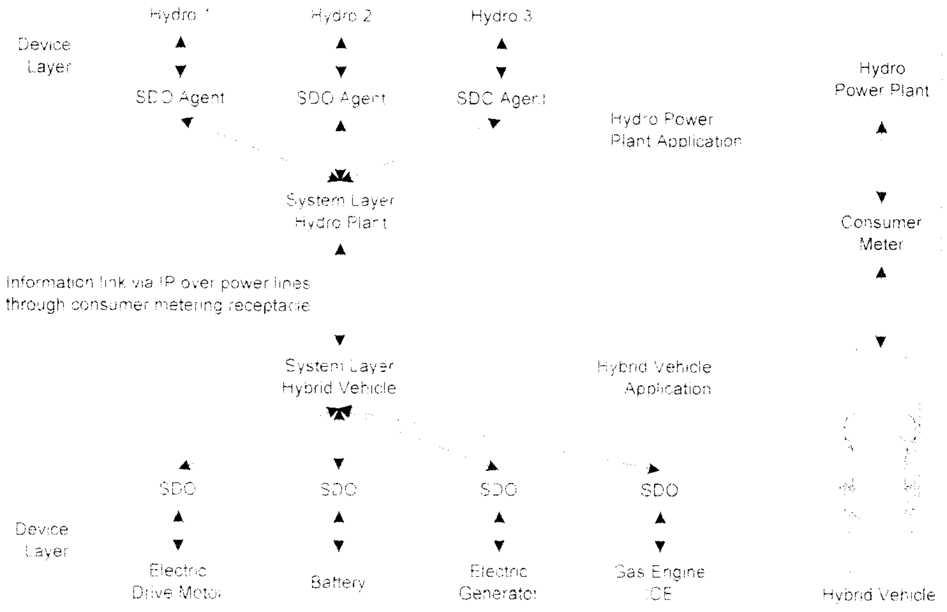


Figure 4.20. Overview of hydro plant to hybrid vehicle coupling.

Hydro-Generating Plant Application

Application to the hydro-generating units and plant requires rules for the individual units in their respective software device agents, methods for interfacing the device layer to the system layer, training sets for the neural network pre-classifier, and rules for the expert system. The rule sets developed in the previous hydro-generating application would be used similarly with the distinction of location in this architecture. Rules for local unit optimization would reside in the software device agent and rules for plant management would reside in the system layer. Table. 4.9 illustrates how these rules would be relocated to achieve the decentralized benefits of individual unit optimization and the centralized benefits of coordinated plant-wide management.

Layer	Rule set
Unit's SDO agent	Optimal point control
	Trouble condition handling
	River trash load ejection
	Safe fish passage rules
System layer rules	Number of units to run
	Startup and shutdown prioritization
	Flow redistribution
	Corporate dispatching rules

Table 4.9. Rule sets in the hydro-generating application.

The rule sets for optimal point control, trouble conditions, load ejection, and safe fish passage are applicable to a specific unit as defined in the previous hydro-generating application. These rules run in the device layer and are distributed among the individual generating units. The rules sets for how many units to run, the order of startup and shutdown, flow distribution assignments as a response to the unit's optimal point control, and rules for dealing with corporate dispatch are applicable to the whole plant and reside in the centralized control of the system layer.

In addition to the above rule sets for optimization and management, methods in the software device agents need to be defined to interface with the system layer. In *Chapter III*, these methods define the demand, reserve, status, and command inputs and outputs of the software device objects. The demand for a generating device refers to the current power generation. Flow is used to simplify power-based calculations since flow is proportional to power by (4.6). The reserve will indicate the power available defined by the flow to get to the next optimal point of operation, unless a trouble condition exists which would reduce power availability. These are defined in the pseudo code below.

```
; demand is current unit flow
Demand = flow.setpoint
; reserve is flow at next optimal point unless trouble
IF NOT Unit.trouble THEN
        Reserve = flow.optimal.above ; if everything OK try to get more flow
; If the trouble requires a flow reduction, subtract the needed reduction off
ELSE IF Unit.trouble = { reduce flow } THEN
        Reserve = flow.setpoint - flow.trouble.reduced
        ; This tells the system layer to reduce the flow setpoint to the unit
ENDIF
```

The status method would signal the software device agent's status conditions to the system layer. The system layer would consider the demand and reserve values above along with these status messages to decide how to redistribute flow, if necessary. Typical statuses are listed in table 4.10 along with the relation between demand, D, and reserve, R.

Status code	Meaning
Vibration HI	Vibration needing a flow increase to correct. D < R
Vibration LO	Vibration needing a flow decrease to correct. D > R
Cavitation HI	Cavitation needing a flow increase to correct. D < R
Cavitation LO	Cavitation needing a flow decrease to correct. D > R
Stator Temp HI	Reduce flow to cool stator. D < R
Load Eject	Unit needs to perform a trash ejection. D >> R. R=0
Sub Optimal OK	Unit OK but not at optimal flow point. D < R
Optimal OK	Unit OK and at optimal flow point. D=R

```
; Example status list to reduce flow for cavitation
Unit.status = { trouble, cavitation LO }
```

Table 4.10. Status method for the hydro-generating application.

The command input to the software device agent from the system layer also needs to be handled. For a hydro-generating unit, this command would include the flow setpoint and directives to go on or offline or perform a load eject. The pseudo code for the software device agent to handle the system layer command would be defined as in the following examples.

```
; is the command output for this unit? Unit1
IF Command = { Unit1, * } THEN
      Unit.command = { * } ; assign sub list to local command variable
ENDIF
; for a new flow setpoint
IF Unit.command = { set flow, newflow } THEN
      flow.setpoint = newflow
ENDIF
; to permit a load ejection
IF Unit.command = { load eject, TRUE } THEN
      loadeject() ; call load ejection function
ENDIF
```

The system layer would form commands based on the results of its rule sets as in the following pseudo code example.

```
; to decrease flow to unit 1 by 500CFS and
; increase flow to unit 2 by 500CFS and
; deny a requested load eject to unit 3
Command =      { Unit1, set flow, Unit1.demand-500 }
               { Unit2, set flow, Unit2.demand+500 }
               { Unit3, load eject, FALSE }
```

Prior to management decisions being applied by the system layer, it is beneficial to have the operating state classification performed by the artificial neural network for feature extraction. Neural networks apply well to control environments where a standard model is not clearly defined. This allows the neural network to find abstract relationships in the operating state pattern that would otherwise be undetectable by conventional rule conditions. This is known as a soft or virtual sensor approach where an immeasurable input condition is detected by pattern matching with available inputs. Neural networks also provide a continuous analog classification as opposed to the discrete classifications of expert systems. This allows a classification to be quantified in terms of confidence relative to other classifications.

Table 4.11 illustrates the inputs and outputs selected for the neural network. The demand and reserve for each unit is selected along with the actual load generated by each unit, the plant head, and the river temperature. The outputs are selected to provide an analog measure of: distance from optimal operation to improve optimal control; sensor validation to detect failures in measurement or changes in machine conditions; and seasonal classification to enable the expert system to execute seasonal strategies. This results in a relatively small and manageable neural network of 11 inputs and 7 outputs.

Inputs (11)	**Outputs (7)**
demand, unit 1	distance to optimal operation, unit 1
demand, unit 2	distance to optimal operation, unit 2
demand, unit 3	distance to optimal operation, unit 3
reserve, unit 1	sensor validation, unit 1
reserve, unit 2	sensor validation, unit 2
reserve, unit 3	sensor validation, unit 3
load, unit 1	river season
load, unit 2	
load, unit 3	
plant head	
river temperature	

Table 4.11. Neural network configuration of hydro-generating application.

Developing the neural network to classify the inputs in the above manner involves training with a user-defined data set. Coupling the desired outputs to the given inputs assembles the training set. For the first three outputs of distance from optimal operation, the previously discussed database of optimal points is used, as in figure 4.14. A sample operating point is chosen and compared to the optimal operating point. The desired output is the distance to this point. Additional sample operating points would be chosen to define the boundaries of the operating space. This operating space is defined by the Hill curve in figure 4.15. Neural networks provide good interpolation in their trained space and would subsequently classify any sample point relative to the closest optimal point. This determination of distance from optimal operation can be used together with the statuses reported by the device layer to provide validation for reports of sub-optimal unit operation and determine how far from optimal the plant is operating.

For the second set of three outputs, unit generation (load) is compared to the current operating states. The relation between the inputs of head and flow, and the output of load would be a training pattern. When the actual value for unit load differs from the trained value, the system layer can conclude that either there is some error in the field inputs or that the machine condition has changed in some way, as shown in the following pseudo code.

```
; train the neural network with user-defined patterns
; (neural network inputs) -> (neural network outputs)
(head,flow) -> load.expected ; for several sample points
; if actual and expected loads differ significantly
IF load.expected <<>> load.actual THEN
        Unit.trouble = TRUE
        ; there must be some problem
        ; if load is too high it must be a sensor problem
        IF load.expected << load.actual THEN
                Unit.trouble.condition = sensorfail
        ENDIF
        ; if load is too low it might be a machine failure
        IF load.expected >> load.actual THEN
                Unit.trouble.condition = machinefail
        ENDIF
ENDIF
```

The last output of river season would consider the head, flows, and river temperature to determine the season, either summer or winter, trained similar to the above. This allows the system layer to have different strategies for different times of the year. For example, fish migration may not be an issue in the winter season so this module could be deactivated during this time.

By using the neural network, more classification information can be generated to allow better rules to be constructed in the expert system. The neural network enhances the ability of the system layer to deal with continuous data and interpolate between known operating points, which can be a weak point for purely rule-based systems.

Hybrid Vehicle Application

So far, the architecture has been applied and discussed in terms of power generating plants. However, the flexibility of the architecture allows it to be applied to any system needing power management. Hybrid vehicles are an excellent example since they include power generation, power storage, and power utilization devices. Thus, they represent a microcosm of a complete power cycle. The hybrid vehicle exists to reduce fuel consumption and subsequently reduce environmental emissions. Power management is a critical technology for these goals that enables them to succeed while being easily accessible to consumers.

Figure 4.20 above includes the illustration of a hybrid vehicle power system. The main components are a gasoline engine and generator for power generation, a battery for power storage, and an electric motor as a load device that drives the vehicle. Each of these devices receives a software device object according to the architecture in *Chapter III*. The respective software device objects incorporate the methods for demand, reserve, status, and commands for interfacing with the system layer: and local optimization methods for the specific device. Pseudo code for these devices is as follows.

```
; battery demand, power being supplied (+) or charging power (-)
battery.demand = battery.voltage * battery.current
; battery reserve, battery capacity or state of charge
; battery starts with some charge
IF Initialize THEN  ; system startup
        battery.reserve = battery.initialcharge
ENDIF
; battery reserve changes as power is supplied or stored
; performed by stepwise numerical integration from initial value
WHILE Operating DO  ; during operation
        battery.reserve = battery.reserve - battery.demand * timestep
ENDWHILE
; C = 3

; electric drive motor
motor.demand = motor.voltage * motor.current
; motor reserve is simply its capability unless there is trouble
IF motor.trouble = FALSE THEN
        motor.reserve = motor.rating
ELSE
        motor.reserve = motor.rating - motor.trouble.derate
ENDIF
; C = 2

; internal combustion engine ICE
engine.torque = lookup.table(airflow, fuelflow, engine.rpm)
engine.demand = engine.torque * engine.rpm
; reserve is power available at optimal point
engine.reserve = engine.torque.optimal * engine.rpm.optimal
; C = 2

; generator is driven solely by engine
generator.demand = generator.voltage * generator.current
```

```
; the reserve is what the ICE can supply unless there is trouble
IF generator.trouble = FALSE THEN
        generator.reserve = engine.reserve
ELSE
        generator.reserve = engine.reserve - generator.trouble.derate
ENDIF
; C = 2
```

The system layer provides power management of the above devices for the hybrid vehicle. Figure 4.21 illustrates the power flow paths between devices. Table 4.12 demonstrates the different modes of operation of the vehicle and the general magnitude and sign of the power flow quantities noted in figure 4.21.

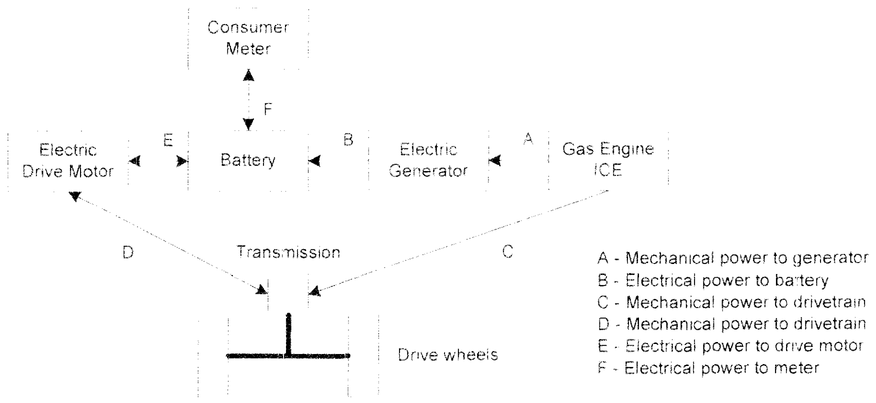

Consumer Meter
F
Electric Drive Motor
E
Battery
B
Electric Generator
A
Gas Engine ICE
D
Transmission
C
Drive wheels
A - Mechanical power to generator
B - Electrical power to battery
C - Mechanical power to drivetrain
D - Mechanical power to drivetrain
E - Electrical power to drive motor
F - Electrical power to meter

Figure 4.21. Hybrid vehicle power flow.

Operating mode	A	B	C	D	E	F
Idle	0	0	0	0	0	0
Cruising, low acceleration	0	0	0	+MED	+MED	0
Med acceleration	0	0	0	+HI	+HI	0
High acceleration	+MED	+MED	+MED	+HI	+HI	0
Medium regenerative braking	0	0	0	-HI	-HI	0
High combination braking	0	0	0	-HI	-HI	0
Mobile charging / battery low	+MED	+MED	+MED	0	0	0
Home charging / power storage	0	0	0	0	0	-HI
Home power discharge	0	0	0	0	0	+MED
Home backup power	+MED	+MED	0	0	0	+MED

Table 4.12. Power flow relative quantities.

In table 4.12, HI refers to nearly maximum capability and MED refers to a medium or mediated level determined by the power management system with the sign referring to the direction of net power flow. The idle mode is self-explanatory in which no power is moved within the system. During cruising at steady speeds or with light acceleration and deceleration, the electric motor is used exclusively. When high acceleration is needed, the ICE is added for extra drive power although this is not an efficient type of driving behavior. During moderate braking effort, regenerative braking is used through the drive motor and absorbed by the battery for recovery. When high braking effort is needed, the vehicles hydraulic brakes are used in combination. Mobile charging occurs when the battery's state of charge is low during transit and the ICE is then used for supplemental charging. The modes for when the vehicle is docked at home are discussed in greater detail in the *Coupling the Applications' System Layers* section below. In summary, the battery can be charged from the power utility grid, discharged to the grid for grid stabilization, or discharged to the home for backup power during blackouts. The user preferences for these are discussed here as well. Whenever the ICE is utilized, it is operated at its most efficient operating point with the mechanical drive from the electric motor making up the difference. The strategy is to use electrical motor for primary propulsion and the ICE only when supplemental power or battery charging is needed, thus minimizing fuel consumption and environmental pollution.

The neural network is now developed for pre-classification and summary data. In addition to the demand and reserve inputs from the device layer, the user inputs relevant to mobile operation of the vehicle of throttle and braking inputs, which are typically on a scale of 0-100% of vehicle capability, are included. These are listed in table 4.13.

Inputs (10)	**Outputs (5)**
Demand, electric motor	electrical demand summary
Demand, battery	mechanical demand summary
Demand, generator	predicted vehicle range
Demand, engine	predicted vehicle fuel economy
Reserve, electric motor	predicted vehicle emissions
Reserve, battery	
Reserve, generator	
Reserve, engine	
User throttle position	
User braking effort	

Table 4.13. Neural network configuration of hybrid vehicle application.

The outputs of the neural network would be analog values on a range of -1 to +1 with the expert system rules providing proper scaling to engineering units. These would provide an indication of the current requirements for the vehicle to aid in expert system rule construction. The relative quantities in table 4.12 would provide a guideline for generating training data. Although the expert system can perform classification, the neural network can also classify the operating state and with a continuous, analog quantification. This can aid the rule sets in determining quantitative command responses to the device layer. Considering the electrical and mechanical demand summary outputs for example, when the battery is too low to drive the vehicle effectively but the user needs propulsion, the ICE is used to both drive the vehicle and provide battery charging. Not only can the neural network classify this discrete state as mobile charging, it can also provide a continuous classification guiding the expert system in the appropriate balance of ICE mechanical power between the drivetrain and generator. This is illustrated in figure 4.22.

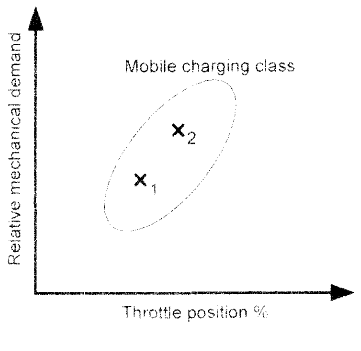


Figure 4.22. Neural network interpolative classification example.

In figure 4.22, both x_1 and x_2 are in the operating state of mobile battery while the vehicle is in operation. However, the balance of power mechanical power sent to the drivetrain versus the generator is different in each case. The output of the neural network aids the expert system in determining the power appropriate balance as in the following pseudo code. This pseudo code therefore handles any points within the mobile charging classification due to the neural network's natural interpolative capabilities.

```
IF vehicle.mode = MobileCharging THEN
        engine.power.drivetrain = constant * NN.mechanical_demand
        engine.power.generator = engine.power.total - engine.power.drivetrain
ENDIF
; C = 2
```

Raw data from vehicle simulations can be used to predict the real-time vehicle range and fuel economy under a given set of conditions [Bauman, 2007] [Gao, 2001]. This data is used directly to train the neural network in the prediction of these quantities for the expert system and user feedback, thus simplifying the generation of training data. The final output of environmental emissions is determined directly by the quantity of fuel burned.

Some sample rules for the system layer are given below that outline strategies for regenerative and combination braking, starting and stopping the motor and ICE, mobile charging mode, and optimal operation of the ICE. Additional rules would follow similarly and software device object commands would be formed and communicated as in the previous hydro-generating application.

```
; sample of braking balance rules
IF userinput.brake > 0 THEN
```

```
        IF userinput.brake > motor.regen.capability THEN
                vehicle.mode = CombinationBraking
                motor.regen.setpoint = 100%
                ; hydraulic brakes will make up the difference externally
        ELSE
                ; use brake to modulate regenerative braking
                vehicle.mode = RegenerativeBraking
                motor.regen.setpoint = constant * userinput.brake
        ENDIF
ELSE
        motor.regen.setpoint = 0 ; turn off regenerative braking mode
ENDIF
; C = 3

; sample rules for when to start up and shut down ICE and mobile charging
; if user wants more drive power than the motor can deliver
IF userinput.throttle > motor.power.capability THEN
        engine.start()
ENDIF
; if battery gets too low then use ICE to recharge
IF battery.reserve < battery.minimum THEN
        engine.start()
        ; stop the motor for quicker charging if ICE is sufficient
        IF userinput.throttle < engine.power.capability THEN
                motor.stop()
        ELSE IF motor.stopped THEN
                motor.start()
        ENDIF
ENDIF
; if ICE is running when to shut down
IF engine.running THEN
        IF batter.reserve > 80% THEN
                ; if battery ok and motor is sufficient then stop ICE
                IF userinput.throttle < motor.power.capability THEN
                                engine.stop()
                ENDIF
        ENDIF
ENDIF
; C = 8

; sample rules to operate ICE at optimal operating point with exception
IF engine.started THEN
        ; control engine throttle for optimum rpm by default
        ; use internal PID algorithm with below configuration
        engine.PID.setpoint = optimum_rpm
        engine.PID.analogvalue = engine.rpm.current
        engine.throttle = engine.PID.output
        ; if we need more power then go ahead but only when true
        IF userinput.throttle > (motor.power.capability + engine.power.current) THEN
              engine.throttle = user.throttle - motor.power.capability
        ENDIF
ENDIF
; C = 3
```

More rules to handle coupling with the power utility and home charging and back up are given in the following section.

Coupling the Applications' System Layers

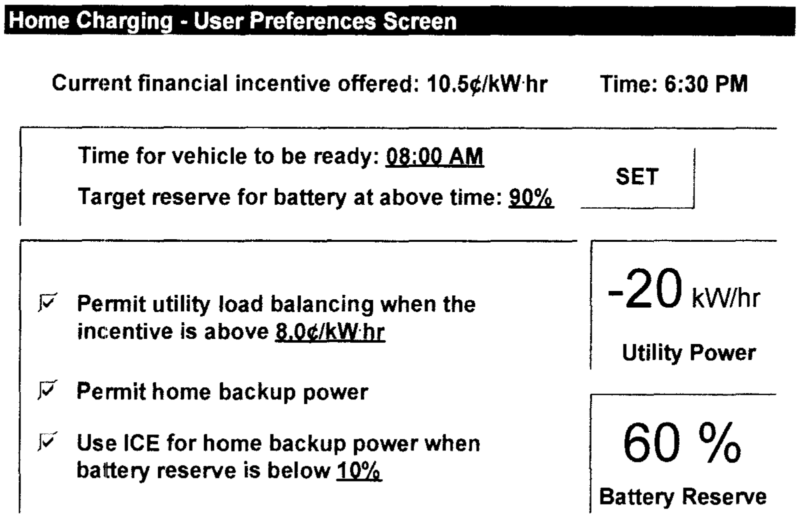
In the hydro-generating and hybrid vehicle applications above, the flexibility of the architecture has been demonstrated as a means of achieving power management. These two applications will now be coupled to form a coordinated solution for power management between the generating facility and the vehicle end user. The new capabilities achieved by coordination demonstrate how this whole is greater than the sum of its parts. Specifically, that simply joining these two applications preserves their individual capabilities while enhancing them both to a new level.

Because the power systems' devices do not change in this coupling, the software device objects and agents do not change. Also, the neural network configuration does not change since this performs local classification. Therefore, all of the coordinating effort is done through the system layer's rule-based expert system. The existing rules for power management of the respective application remain intact and an additional rule set is added to achieve coordination. The additional capabilities achieved through this coupling are listed in table 4.14.

Mode	**Coordination**
Battery recharging	Consumer demanded recharging of vehicle battery
Power storage	Storage of power in vehicle battery by power plant
Power depletion	Recovery of power in vehicle battery by power plant
Power backup	Backup power for consumer's home during power outages

Table 4.14. Modes of coordination.

The mode of battery recharging is the obvious plug-in recharging method from the power grid to the vehicle's battery. Power storage functions the same way with the exception of being initiated by the power plant. This is for the storage of grid power at low demand times, which is later recovered by the power plant at high demand times during the power depletion mode. Thus, these two modes achieve load balancing and power grid stabilization. The final mode of power backup is a feature that allows the vehicle's battery to supply power for the consumer's home during power outages or power cycling. The consumer chooses to participate in load balancing and receives a financial benefit for participation. A sample user interface is illustrated in figure 4.23 of the status and user preferences relevant to these modes.



Home Charging - User Preferences Screen

Current financial incentive offered: 10.5¢/kW·hr   Time: 6:30 PM

Time for vehicle to be ready: 08:00 AM

Target reserve for battery at above time: 90%

SET

☑ Permit utility load balancing when the incentive is above 8.0¢/kW·hr

☑ Permit home backup power

☑ Use ICE for home backup power when battery reserve is below 10%

-20 kW/hr

Utility Power

60 %

Battery Reserve

Figure 4.23. User preferences interface for coupling application.

The user preferences screen in figure 4.23 allows the consumer to set the desired time for utilizing the vehicle and the minimum reserve of the battery. The vehicle's system layer will attempt to meet these parameters by controlling the amount of power transfer between the vehicle and the power grid. The consumer may also participate in load balancing for a financial incentive or enable a power backup for their home with parameters defining how these modes are executed. The current battery reserve is given in percent and the current power flow between the vehicle and the power grid is shown with positive values indicating discharging to the grid and negative values indicating charging of the battery. The financial incentive offered to the consumer by (4.14) and to the power plant by (4.13) is based on the market pricing at different times by (4.12).

$$B = MP_R - MP_S \tag{4.12}$$

$$B_P = (1 - k_{share})B \tag{4.13}$$

$$B_C = k_{share} B_P \tag{4.14}$$

Where B is the benefit, MP is the market price, and k_{share} is the benefit-sharing fraction to the consumer. The subscripts of C and P denote consumer and plant respectively while the subscripts of R and S denote the times of recovery and storage respectively. During peak summer demand for example, the difference between the market price at midnight during storage and the market price at noon during recovery can be as high as an order of magnitude. This translates to both a good financial incentive for both parties as well as effective load balancing. The rules for realizing this negotiation in the hybrid vehicle's system layer are developed below using the values in figure 4.23.

```
; for simple home charging without load balancing
IF battery.reserve < 90% AND NOT permit.loadbalancing THEN
       Chargebattery(ON)
ELSE IF battery.reserve > 90% THEN
       Chargebattery(OFF)
ENDIF
; C. = 3

; if there is plenty of time to reach charge target allow load balancing
IF (time.ready - time.current) > time.tocharge(90%-battery.reserve) THEN
       IF permit.loadbalancing AND incentive.current > incentive.min THEN
              Loadbalancing(ON)
       ELSE
              Loadbalancing(OFF)
       ENDIF
ELSE ; just charge the battery for the consumer if needed
       Loadbalancing(OFF)
       IF battery.reserve < 90% THEN
```

```
            Chargebattery(ON)
        ELSE
            Chargebattery(OFF)
        ENDIF
ENDIF
; C  = 5

; send command to utility for load balancing
FUNCTION Loadbalancing(state)
        IF state = ON THEN
            Command.meter = { loadbalancing, allow }
        ELSE
            Command.meter = { loadbalancing, disallow }
        ENDIF
; C  = 2

; for home backup power during outages
IF permit.backup THEN
        IF utility.power = 0 THEN
            IF battery.reserve > 10% THEN
                Supplyhome(ON) ; send power to home
            ELSE IF permit.backup.useICE THEN
                Supplyhome(ON)
                Chargebattery(ON)
            ELSE
                Supplymeter(OFF) ; discontinue backup
            ENDIF
        ELSE
            Supplymeter(OFF) ; discontinue backup
        ENDIF
ELSE
            Supplymeter(OFF) ; discontinue backup
ENDIF
; C  = 5

; supply meter sends power to home during outages and
; power to grid when grid online
FUNCTION Supplymeter(state)
        Chargebattery(OFF)
        Dischargebattery(ON) ; connect battery to meter / grid
```

The rules for realizing this negotiation in the hydro-generating plant's system layer are developed below.

```
; compute and send incentive
; F is defined by corporate management as a function of given variables
; also by equations (4.12), (4.13), and (4.14)
incentive.consumer = F(price_forcasting, demand_forcasting)
Command.consumer = { incentive, incentive.consumer }

; if commanded to load balance then do so
IF Command.consumer = { loadbalancing, allow } THEN
        ; to store power charge the consumer's battery
        IF mode.powerstorage THEN
                Command.consumer = { chargebattery, ON }
                Track.powertoconsumer(Stored)
        ENDIF
        ; to recover power discharge the consumer's battery
        IF mode.powerrecovery THEN
                Command.consumer = { supplymeter, ON }
                Track.powerfromconsumer(Recovered)
        ENDIF
ENDIF
; C = 4

; when done load balancing measure the benefit and credit the consumer
IF Command.consumer = { loadbalancing, disallow } THEN
        Benefit.consumer = incentive.consumer * Recovered
ENDIF
; C = 2
```

Software Metrics of the Applications

The software metrics developed in *Chapter III* are now applied to the applications. In this case, the metrics are defined with respect to the coupling of the hydro-generating and hybrid vehicle applications.

Portability

Comparing the hydro-generating application to the hybrid vehicle application assesses portability. This demonstrates how the architecture can be moved between very different application domains. Table 4.15 compares the modules between the two applications and quantifies the portability of the architecture.

Hydro Plant	**Hybrid Vehicle**	**Effort *w***	**Portability *p***
SDO agent unit1 SDO agent unit2 SDO agent unit3	SDO battery SDO motor SDO generator SDO engine	25%	25%
Neural network	Neural network	25%	25%
Expert system rule sets	Expert system rule sets	50%	25%
Total from (3.1)		100%	25%

Table 4.15. Coupled applications portability.

While the portability of 25% appears low, it is considered that these are very different applications. As in the previous hydro-generating agent-based application, the portability was high moving from one hydro-generating unit to another, even when design parameters were different. Assessing the portability as above points out an important characteristic of the architecture. That characteristic is the preserving of the structure, if not the actual code and rule sets. Between these two applications, the same methods are designed for the software device objects, they are connected to the system layer in the same manner, the neural network is trained in the same way, and the rule sets in the expert system are formed similarly. A score of 25% is assessed signifying that when moving to a very different application domain, the architecture is preserved, thus eliminating the effort of designing a new control environment structure.

Scalability

The scalability of the applications is measured with respect to adding the scope of coordination. Thus, the hydro-generating and hybrid vehicle applications that are complete on their own are now enhanced with additional rules in the system layer to allow coordination to achieve additional goals. This is illustrated in table 4.16.

Hydro Plant	**Hybrid Vehicle**	**Effort *w***	**Scalability *s***
SDO agent unit1 SDO agent unit2 SDO agent unit3	SDO battery SDO motor SDO generator SDO engine	25%	100%
Neural network	Neural network	25%	100%
Expert system rule sets +extra rules for coordination	Expert system rule sets +extra rules for coordination	50%	90%
Total from (3.2)		100%	95%

Table 4.16. Coupled applications scalability.

In table 4.16, the capability of the architecture to scale is clearly demonstrated with a high factor of 95%. Adding the additional scope of coordination between the two applications significantly enhances their capability although only minor effort is required. The layered approach preserves the work in the device layer and neural network pre-classifier allowing these to scale unmodified. The expert system rule sets existing for each individual application are also largely unchanged. The only modifications to this module are the addition of rules to handle the coordination as demonstrated in the coupling discussion in the previous section.

Simplicity

Measuring difficulty assesses simplicity, as with previous applications. This is summarized in table 4.17.

Module	**Readability C_R**	**Complexity C_C**	**fanin, fanout**	**Difficulty (3.3) (3.4)**
Hydro plant				
SDO agent unit1	1.5	13	1, 3	176
SDO agent unit2	1.5	13	1, 3	176
SDO agent unit3	1.5	13	1, 3	176
Neural network	3	1	11, 7	17787
Expert system	1.5	25	16, 3	86400
Hydro application total				104715
Hybrid vehicle				
SDO battery	1.5	3	1, 3	41
SDO motor	1.5	2	1, 3	27
SDO generator	1.5	2	1, 3	27
SDO engine	1.5	2	1, 3	27
Neural network	3	1	10, 5	7500
Expert system	1.5	31	17, 4	215016
Vehicle application total				222638

Table 4.17. Coupled applications difficulty.

The readability metric is determined with rules being assessed a 1.5 and neural networks assessed at 3. The complexity is summarized at the end of each section of code. *Fanin* and *fanout* are determined to be relatively large due to the nature of the centralized approach of the system layer. However, due to the high readability of the rules throughout the applications and the relatively small and simply defined neural networks, this is respectable. Especially when compared to the difficulty of the monolithic neural network approach to centralization in the previous coal-fired application.

Autonomy – Automation

When assessing autonomy, the metrics of automation, self-preservation, strategy, and coordination are measured with respect to the coupling of the hydro-generating and hybrid vehicle applications. As before, the hydro-generating plant was already close to fully automated and the scope of the architecture in the new hydro-generating application resulted in at least 90% automation for an assessed value of 3, determined by table 3.3. Full automation of the user's operation of the hybrid vehicle is not desired due to driving preferences of the user base. However, the power management system should relieve the user from any power management functions created by the new scope and allow the user to easily set preferences for power management. In this case, the automation is assessed at 3 for the hybrid vehicle application since it is nearly transparent to the user while serving the user's needs. These result in the coupled applications being assessed an autonomy metric of 3.

Autonomy – Self-preservation

Self-preservation is another feature intentionally designed into the applications and their coupling. Previously, the hydro-generating application was assessed a self-preservation metric of 3 since 90% of the trouble conditions were handled, defined by table 3.4. The handling of trouble conditions in the hybrid vehicle related to power management is included by design within the rule sets. Therefore, the self-preservation metric for the hybrid vehicle and subsequently for the coupled applications is assessed at 3.

Autonomy – Strategy

The previous hydro-generating application was assessed a strategy metric of 2 indicating that there was some room for additional scope, defined by table 3.5. The hybrid vehicle application incorporates the typical operating strategies of efficient use of multiple power system components and adapts to user inputs during operation for additional tasks, e.g. providing high demand acceleration through the electric motor and ICE. The hybrid vehicle is assessed a strategy metric of 2 indicating a few power management goals are achieved. The additional strategies for load balancing and home backup power however increase the strategic scope to a new level and the strategy metric for the coupling is assessed at 3.

Autonomy – Coordination

Coordination was designed into the coupled applications by definition. The previous hydro-generating application was assessed a coordination metric of 2, defined by table 3.6, since it was able to cooperate with other hydro units and handle multiple users. The hybrid vehicle typically handles one instance of a single user type at a time and does not cooperate with other vehicles. However, the coupled applications do cooperate with each other using the unique capabilities of the architecture. This represents a relatively high level of cooperation from the standpoint of typical hybrid vehicle applications. Therefore, the coordination metric for the hybrid vehicle is assessed at 2 and for the coupled applications, is assessed at 3 reflecting the unique capabilities achieved by cooperation between the hybrid vehicle and power utility for both grid stabilization and financial benefit.

The autonomy metric for the coupled applications is therefore $\sqrt{(3^2 + 3^2 + 3^2 + 3^2)}$ or $\sqrt{36} = 6$. This is depicted in the four-dimensional radar chart of figure 4.24.

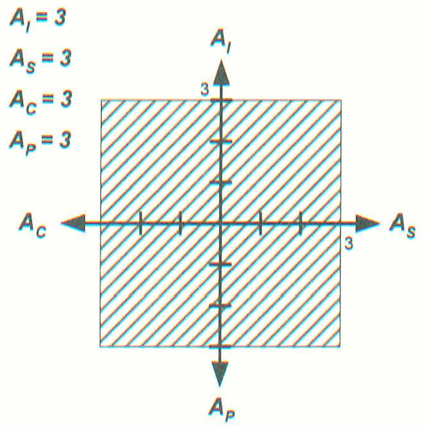


Figure 4.24. Autonomy metric of the coupled applications.

Figure 4.24 represents the largest square for autonomy possible. While this does not indicate there is no room for additional autonomous capability, this is intended to indicate that the two applications and their coupling achieve a complete solution that provides for their individual power management while working together for a power management solution that is greater than either of their individual scopes.

Closing Remarks for the Applications

The above coupling of the plant and vehicle applications demonstrates how the architecture achieves this cooperation. In the macrocosm of the power generating grid, the system layers of a power utility's generating fleet would be coordinated themselves and appear to the consumer as a single power-providing entity. Each consumer would then link their hybrid vehicle resulting in multiple instances of this coupling application.

In [Kempton-1, 2005] and [Kempton-2, 2005], the benefits to the power industry and consumer of the vehicle to grid application are derived in detail. In these papers, the generating capacity of the U.S. power utilities is estimated at 602GW, discounting unregulated generation. There are over 176 million vehicles in operation in the U.S. with an average power generation and consumption of 111kW each. Assuming these vehicles are used only 4% of the time still results in a vehicular power base of 19,500GW, eclipsing the generating capacity of the power industry. The other 96% of the time the vehicle is not in use, it can be made available for power storage. Thus, even a small level of participation would produce significant results. The financial benefits to each participating consumer would be approximately $2000-$4000 annually. This is on the same order as the average consumer's electric power costs, resulting in a direct offset of the consumer's power costs. Additionally, as wind and solar power become more prevalent, the storage capability of vehicle to grid technology becomes more important as an enabling technology to improve the availability of these power-generating options.

While these papers demonstrate the benefits of vehicle to grid technology, this architecture provides a software solution for achieving this technology. As consumers embrace the hybrid vehicle technology, they will look for new ways to harness its capabilities. The portable, scalable, simple, and autonomous architecture presented is a path to achieving these new benefits.

CHAPTER V

CONCLUSIONS AND FUTURE DIRECTIONS

The Architecture as a Solution

In *Chapter I*, the problems in using current power management systems were outlined. These systems typically had a simple model, e.g. a large neural network optimizer, yet these simple models became difficult to implement or understand once implemented. The author was the lead engineer on such a large neural network optimization at a coal-fired power generating plant. When a preliminary nitrogen oxide reduction was performed, it was observed that opacity had been increased significantly. Therefore, the optimization needed to account for this as well. This shift in scope required much rework of the neural network model and even when this model was completed, similar shifts in scope would have the same drawback. Additionally, local operations and management staff did not readily accept the neural network. The black-box approach could not answer questions such as, *"What would be the response to X stimulus?"* without first submitting the response. These unknowns caused the application to proceed with apprehension, and perhaps rightly so. A white-box approach was needed, but one that could be assembled modularly. This was achieved with the rule-based approach outlined in *Chapter III.*

As such advanced control techniques have matured; management has come to embrace their aspects of increased functionality and data analysis. This has revealed another limitation in existing models: the lack of scalability. The neural network model, for example, could not adapt to multiple corporate users now applying varying degrees of influence and rapidly shifting goals. Architecture with an enterprise-level solution was needed while still being able to manage the lowest level of details in the process. The layered approach outlined in *Chapter III* demonstrated how this problem would be addressed. By engaging multiple business entities in the decision-making process, more advanced multi-goal solutions are now possible. This keeps business entities informed for their own strategic benefit while letting them contribute their strategic influence over the process being optimized, through the architecture's layered approach, to realize an enterprise-level power management system. As competition increases, enterprise-level power management systems are becoming mission-critical solutions for the next generation of power systems.

As this architecture was being developed, the author noticed other areas, such as hybrid vehicles, which would benefit from power management. It seemed redundant that such areas would require a whole new architecture, thus this architecture was enhanced to include support for these. It was found that this enhancement brought returns to all application areas and the developed architecture could be ported and scaled as needed for almost any power management problem. This also results in a collaborative benefit whereby researchers in one application field of power management could utilize lessons learned from an unrelated field as a fresh perspective.

One of the primary concerns in applying for research grants is that solutions should be meaningful and provide real benefit. This architecture was developed to be accessible to those in industry who might apply them. While there have been other architecture with many of the same capabilities, none have been as flexible as the layered and modularized approach developed here. This is discussed in detail in the following sections of *Scalability*, *Portability*, *Simplicity*, and *Autonomy*. When it is considered that power generation and utilization consume our natural resources, impact our environment, and limit or exclude some missions on a large scale involving all individuals and companies alike, the benefits of architecture that is not only effective but also implementable by those in the field becomes obvious.

Scalability in the Applications

Scalability is the ability of the architecture to grow to meet new goals or an expanding mission. The scalability demonstrated in the applications of *Chapter IV* illustrates how different types of application modules are modified when they scale. Neural networks scale very poorly while well-written rules scale very well. Even rules that need to be changed are changed individually, minimizing rework. The architecture demonstrates good orthogonality since changes in software device objects are relatively independent of the system layer and vice versus. The device layer methods for *demand*, *reserve*, and *status* as well as the handling of *commands* are encapsulated in the software device objects. The system layer is reserved for whole process power management so that individual device or sub-system rule sets are not present to jeopardize the quality of loose coupling.

Security in Scalable Applications

In a scalable application where multiple users with different missions to accomplish are present, security must be a consideration. Between the device layer and the system layer, security is not significant as these layers reside internally to the local power management system. Security for local users is accomplished using the existing authentication protocols present in the HMI or SCADA user interfaces. As users outside the normal operation of the process are introduced, security needs to be customized for each user. This is accomplished through the database server or global SCADA system and again would utilize existing authentication protocols. This allows the architecture to incorporate access-level security by taking advantage of existing protocols. The rule sets developed by these business entities for their influence over the local system layer can be partitioned into special sections of the expert system to enforce limitations on their influence, or the rule sets can be reviewed manually by local engineering staff.

Portability in the Applications

Portability is the ability of the architecture to be applied to varying missions and power system configurations. By using the layered approach, modifications that would be necessary when moving from one implementation to another become modularized. Therefore, if a new implementation is to utilize the same hardware components, the device layer will port to the new implementation virtually unchanged. In the system layer, there are some rule changes but these rules are individually modifiable simplifying portability.

The applications in *Chapter IV* demonstrate the extremes of portability in power management systems. In the large monolithic neural network application, portability was virtually nonexistent; as even identically configured generating units could not utilize the same neural network due to variances in device conditions such as age and machine wear. The hydro-generating software agents with 100% portability among identical units and the architecture applied to the vehicle to grid technology however achieved much greater portability due to their rule-based approach and cohesive software object / agent definitions. This was clearly demonstrated in the portability metrics of these applications.

Simplicity in the Applications

Simplicity refers to the ease of implementing and maintaining the architecture in the power system environment, including ease of understanding by maintenance personnel and users of the architecture. The architecture achieves simplicity by using a layered approach to modularize the optimization and management problem, breaking the problem into simpler pieces, which can then be quickly coded or ported to/from other implementations. Once in place, the system becomes a white-box solution that is quick to learn, easy to maintain, and well accepted among its users. For enterprise-level solutions, each user interfaces only with their respective rule sets of the system layer.

While the resulting difficulty of the vehicle to grid application is demonstrated as greater than the previous hydro-generating application, it should be considered that the vehicle to grid application incorporates a much greater scope. Also, it can be seen that this architecture can move to other applications with a relatively small change in difficulty since the architecture is so cohesively defined. Each module is well defined and its interconnections pre-defined regardless of application type. This is in contrast to the first neural network application which is unreadable once in place and must be completely reconfigured for each application, as determined by its high difficulty measure and thus inferior simplicity to the architecture of *Chapter III*.

Autonomy in the Applications

Autonomy refers to the ability of the architecture to provide an intelligent decision-making capability to the application with minimal user interaction. This autonomy frees the operator from constant supervision allowing them to perform other actions and otherwise simplifying operation. Autonomy also becomes a mission-enabling feature for applications where user interaction is limited or unavailable, e.g. space and certain military environments. Figure 5.1 illustrates how autonomous and intelligent decisions are produced in the architecture.

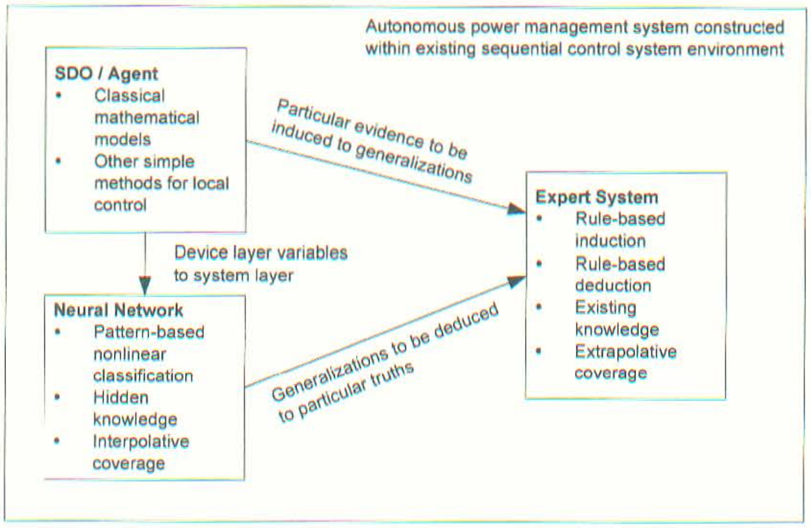

Autonomous power management system constructed within existing sequential control system environment
SDO / Agent
Classical mathematical models
Other simple methods for local control
Particular evidence to be induced to generalizations
Expert System
Rule-based induction
Rule-based deduction
Existing knowledge
Extrapolative coverage
Device layer variables to system layer
Neural Network
Pattern-based nonlinear classification
Hidden knowledge
Interpolative coverage
Generalizations to be deduced to particular truths

Figure 5.1. Autonomy in decision-making.

The monolithic neural network application did achieve autonomy, though hidden from the operator, but with limited ability to handle multiple goals and users. The software agents in the hydro-generating application added multiple goals and users with the agents having some interaction among them. A high level of automation and self-preservation through alarm handling were also achieved. The vehicle to grid application demonstrated the highest level with autonomy by taking the hydro-generating application further through cooperation with a hybrid vehicle's power management system. This allowed new the goals of power storage and load stabilization to be realized on the hydro-generating end while benefiting the vehicle user with financial incentives and scheduled charging. It is clear in this application that additional strategies and cooperative efforts are built in to the architecture and can be accessed by additional rules sets added modularly.

Future Directions

The following future directions are ideas for expanding the application of the architecture. These demonstrate the flexibility and growth potential inherent in the architecture and give some direction for research ideas that would be interesting, practical, and beneficial.

Closing the Loop in Consumer Power Generation

New capabilities are allowing bidirectional communications between power utilities and their customers over the existing power lines, e.g. Internet protocol over power lines (IPoP). This was discussed in the vehicle to grid application in *Chapter IV*. This architecture can take advantage of these new capabilities by modeling the customers as unique instances of a generic load device and interfacing them with software device objects or agents on the utility side. This would allow customers and utilities to directly negotiate their power needs for a more robust and responsive power delivery solution. Instead of trying to predict the chaotic behavior of thousands of customers, software agents would know this behavior in advance allowing power utilities to make more accurate decisions about power generation and delivery needs. A proposed design would follow similar to figure 5.2.

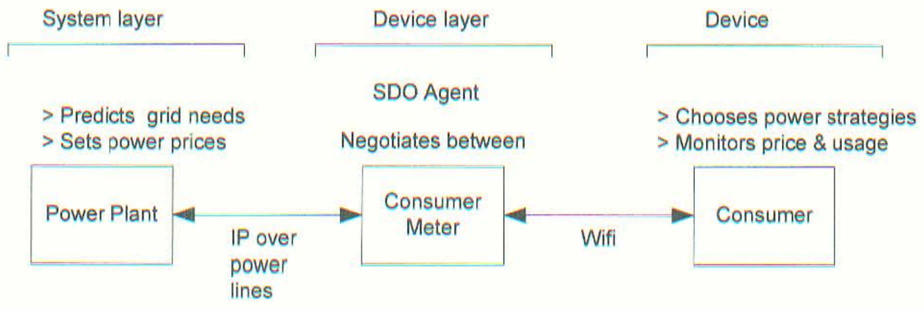
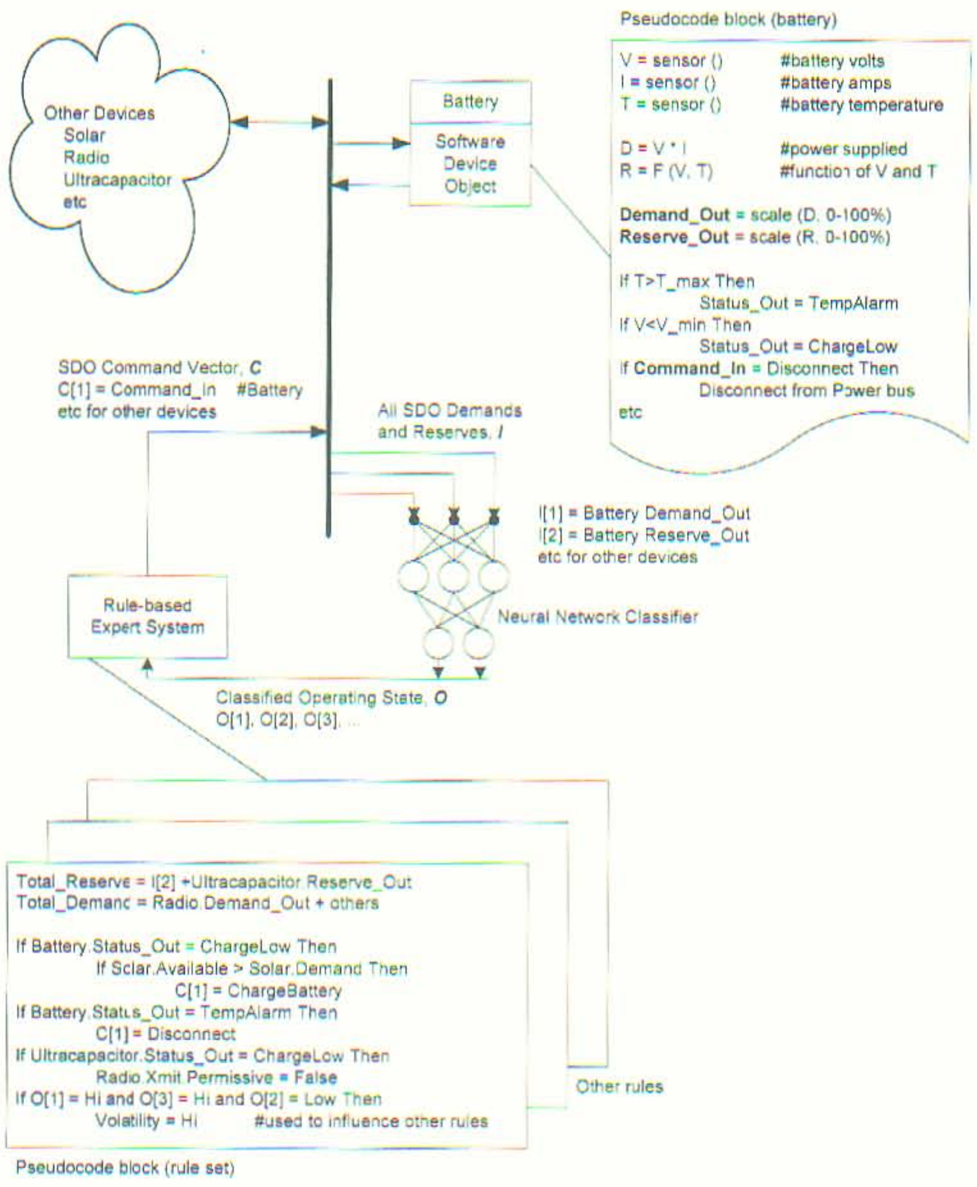


Figure 5.2. Closing the loop in consumer power generation.

Application of the Architecture to a Picosatellite

The picosatellite is also an excellent example of how the architecture can move from the industrial scale to the pico scale to handle new mission constraints such as mass, physical size, limited user direction, and power availability. A typical picosatellite conforms to the standards similar to the CubeSat [Heidt, 2000], being 10cm cubed with a mass of less than 1kg. The power management system needs to be robust and autonomous since the space environment does not afford the opportunities for repair and communications is limited by its consumption of power and the bandwidth and delay in transmission over great distances. Software power management is utilized since it is a zero-footprint enhancement that improves power availability through optimal management and power capability through strategic management. The architecture, therefore, becomes a mission-enabling technology for such vehicles.

In figure 5.3 the implementation of the whole power management system for the picosatellite is illustrated. The battery device is used as an example with other devices being incorporated similarly. Sample methods for the software device objects (SDO) and sample rules for the expert system are also shown in their proper location. The pseudo code block for the battery determines the demand and reserve outputs to the system layer and performs actions based on the command input from the system layer. Internal methods determine the battery's voltage, current, and temperature and scale these from 0-100%. The neural network performs classification and feature extraction of these SDO outputs (input vector ***I***) to serve as additional information for building predicates in the expert system rule set. The expert system then takes the output classification, ***O***, along with the outputs and statuses of the SDOs to determine commands for the SDOs using the rule set. This is shown in the pseudo code block for the expert system.



Pseudocode block (battery)

```
V = sensor ()          #battery volts
I = sensor ()          #battery amps
T = sensor ()          #battery temperature

D = V * I              #power supplied
R = F (V, T)           #function of V and T

Demand_Out = scale (D, 0-100%)
Reserve_Out = scale (R, 0-100%)

If T>T_max Then
        Status_Out = TempAlarm
If V<V_min Then
        Status_Out = ChargeLow
If Command_In = Disconnect Then
        Disconnect from Power bus
etc
```

```
Total_Reserve = I[2] +Ultracapacitor.Reserve_Out
Total_Demand = Radio.Demand_Out + others

If Battery.Status_Out = ChargeLow Then
        If Solar.Available > Solar.Demand Then
                C[1] = ChargeBattery
If Battery.Status_Out = TempAlarm Then
        C[1] = Disconnect
If Ultracapacitor.Status_Out = ChargeLow Then
        Radio.Xmit.Permissive = False
If O[1] = Hi and O[3] = Hi and O[2] = Low Then
        Volatility = Hi          #used to influence other rules
```

Pseudocode block (rule set)

Figure 5.3. Picosatellite power management system.

The picosatellite application is intended to make space vehicle research affordable to educational institutions and small businesses. The picosatellite vehicle is also composed of multiple systems and therefore provides the opportunity for multiple diverse teams to learn to cooperate while developing their individual devices. In educational institutions, this involves more students and teaches them how to participate in larger teams and with outside teams having different needs and goals.

Herrell [Herrell, 2007] provides great detail and sources for the planning of space missions including picosatellites in conjunction with NASA's *New Millennium Program* at http://nmp.jpl.nasa.gov. Table 3 of Herrell [Herrell, 2007] lists 29 parameters that define a mission and are used to test against other planned flights for rideshare (additional spacecraft on an existing launch vehicle) or piggyback (additional experiment on an existing spacecraft) compatibility. As discussed in *Chapter II*, picosatellites are launched in a bundled group. Details of previous missions are well summarized at Michael's list of CubeSat missions [Thomsen, 2008].

Costs for individual CubeSat components and the CubeSat kit are listed at the Pumpkin Inc. web site [Pumpkin-1, 2008]. Budgets for the base picosatellite vehicle and space on a multi-picosatellite launch vehicle can be less than $100k making this approach to space research affordable to educational institutions that share costs and resources across multiple teams as well as small businesses.

REFERENCES

M. Amin, "Restructuring the Electric Enterprise," Chapter 3 in *Market Analysis and Resource Management*, Kluwer Publishers, Mar 2002.

AMSAT web site for amateur radio CubeSat frequencies, found at http://www.amsat.org, April 2008.

Aspen Technologies Inc., "Power Generation Industry Integrated Solution," from company web site http://www.aspentech.com, Oct 2002.

K. Astrom and T. Hagglund, *PID Controllers: Theory, Design, and Tuning*, Research Triangle Park, *NC: Instr. Soc. America*, 1995.

J. Bauman and M. Kazerani, "A Comparative Study of Fuel Cell-Battery, Fuel Cell-Ultracapacitor, and Fuel Cell-Battery-Ultracapacitor Vehicles, *IEEE Transactions on Vehicular Technology*, pp. 1-9, Accepted for future publication, 2007.

A. J. Beaumont, A. D. Noble, M. Gebhardt, G. Stier, and S. Furry, "Automation of ECU Software Development: from Concept to Production Level Code," *International Congress and Exposition: Electronic Engine Controls*, Detroit, Mi., March 1999.

L. A. Belady and C. J. Evangelisti, "System Partitioning and its Measure," *IBM Res. Rep. RC7560*, 1979.

D. E. Bernard and S. M. Krasner, "Integrating Autonomy Technologies into an Embedded Spacecraft System - Flight Software System Engineering for New Millennium," *Proceedings IEEE Aerospace Conference,* Aspen, CO., vol. 2, pp. 409-420, 1997.

R. C. Booth and W. B. Roland, "Neural Network-based Combustion Optimization Reduces NOx Emissions While Improving Performance," *Proceedings IEEE Industry Applications Dynamic Modeling Control Applications Industry Workshop*, pp. 1-6, 1998.

J. W. Botts, "How Accurate Primary Airflow Measurements Improve Plant Performance", *Power Magazine*, vol. 150, num, 4, 42-47, 2006.

F. P. Brooks Jr., The Mythical Man-Month: Essays on Software Engineering. Reading, MA: Addison-Wesley, 1975.

E. Camponogara, D. Jia, B. H. Krogh, and S. Talukdar, "Distributed Model Predictive Control," *IEEE Control Systems Magazine*, vol. 2, pp. 44–52, 2002.

G. D. Chandler, D. T. McClure, S. F. Hishmeh, J. E. Lumpp. Jr., J. B. Carter, B. K. Malphrus, D. M. Erb, W. C. Hutchison III, G. R. Strickler, J. W. Cutler, and R. J. Twiggs, "Development of an Off-the-Shelf Bus for Small Satellites," *IEEE Aerospace Conference*, pp. 1-16, Mar 2007.

J. Chang, K. Y. Lee, R. Garduno-Ramirez, "Multiagent Control System for a Fossil-Fuel Power Unit," *Power Engineering Society General Meeting IEEE*, vol. 3, pp. 1461-1465, July 2003.

J. Chang and K. Y. Lee, "Feedback Control Agents of the Multiagent Power Plant Control System," *Proc Second International Conference on Machine Learning and Cybernetics*, Nov 2003.

P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman, "AutoClass: A Bayesian Classification System," *Proceedings of the Fifth International Conference on Machine Learning*, 1988.

B. T. Clough, "Metrics, Schmetrics! How the Heck Do You Determine a UAV's Autonomy Anyway?," 2002 PerMIS Conference Proceedings, 2002.

Clyde Space Limited, at web site http://www.clyde-space.com, March 2008.

Company web site for CubeSat Kit, http://www.cubesatkit.com, Apr 2008.

T. Doering, D. Erb, A. White, J. Lumpp, and B. Malphrus, "KySat-1 Mission Overview Draft-A," found at http://www.kysat.com, document 080207-KY1MO-PA, pp. 1-10, Feb 2008.

M. B. Djukanovic, M. S. Calovic, B. V. Vesovic, and D. J. Sobajic, "Neuro-Fuzzy Controller of Low Head Hydropower Plants Using Adaptive-Network Based Fuzzy Inference System," *IEEE Transactions on Energy Conversion*, vol. 12, no. 4, pp. 375-381, Dec 1997.

U. M. Fayyad and E. Simoudis, "Data Mining and Knowledge Discovery," *Tutorial Notes at PADD 1997 First Conference, Practical Applications of Knowledge Discovery and Data Mining*, 1997.

R. K. Fisher, S. Brown, and D. Mathur, "The Importance of the Operation of a Kaplan Turbine on Fish Survivability", *Proceedings of the International Conference on Hydropower*, American Society of Civil Engineers, Atlanta, GA, pp. 392-401, Aug 1997.

C. Foreman and R. Ragade, "Coordinated Optimization at a Hydro Generating Plant by Software Agents," *IEEE Transactions on Control Systems Technology*, accepted for future publication, pp. 1-10, Mar 2008.

Y. Gao and M. Ehsani, "Systematic Design of a Fuel Cell-Powered Hybrid Vehicle Drive Train," *Electric Machines and Drives Conference*, 2001.

P. R. Gluck, "Spacecraft Power Management Software for the New Millennium," *Energy Conversion Engineering Conference*, pp. 2228-2230, July 1997.

W. Graco and R. W. Cooksey, "Feature Selection with Data Mining," *Proceedings PADD 1998 Second Conference, Practical Applications of Knowledge Discovery and Data Mining*, 1998.

R. Hanson, J. Stutz, P. Cheeseman, "Bayesian Classification Theory," *NASA Technical Report FIA-90-12-7-01*, Artificial Intelligence Research Branch, Ames Research Center, 1990.

V. Havlena, J. Findejs, and D. Pachner, "Combustion Optimization with Inferential Sensing," *Proceedings of the American Control Conference*, vol. 5, pp. 3890-3895, May 2002.

V. Havlena and J. Findejs, "Application of Model Predictive Control to Advanced Combustion Optimization," *Control Engineering Practice*, vol. 13, Num 6, pp. 671-680, 2005.

H. Heidt, J. Puig-Suari, A. S. Moore, S. Nakasuka, R. J. Heidt, "CubeSat: A New Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation," *14th Annual/USU Conference on Small Satellites*, August 2000.

J. Henrikson and J. Luk, "Advanced Control and Operational Strategies for NOx Reduction and Heat Rate Optimization at Ontario Hydro's Lambton Generating Station," Ontario Hydro.

S. M. Henry and C. Selig, "Predicting Source-Code Complexity at the Design Stage," *IEEE Software*, vol. 7, num. 2, pp. 36-44, March 1990.

S. M. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering*, vol. SE-7, num. 5, pp. 510-518, Sep 1981.

L. M. Herrell, "Access to Space for Technology Validation Missions: A Practical Guide," *IEEE Aerospace Conference*, pp. 1-8, Mar 2007.

S.-J. Huang, "Enhancement of Hydroelectric Generation Scheduling Using Ant Colony System Based Optimization Approaches," *IEEE Transactions on Energy Conversion*, vol. 16, no. 3, pp. 296-301, Sep 2001.

A. Hugo, "Limitations of Model Predictive Controllers," *Hydrocarbon Processing*, Jan 2000.

Company web site for IAR Systems, http://www.iar.com, April 2008.

N. R. Jennings and S. Bussmann, "Agent-Based Control Systems," *IEEE Control Systems Magazine*, pp. 61-73, Jun 2003.

W. Kempton and J. Tomić, "Vehicle to Grid Fundamentals: Calculating Capacity and Net Revenue," *Journal of Power Sources*, vol. 144, no. 1, pp. 268-279, June 2005.

W. Kempton and J. Tomić, "Vehicle to Grid Implementation: From Stabilizing the Grid to Supporting Large-Scale Renewable Energy," *Journal of Power Sources*, vol. 144, no. 1, pp. 280-294, June 2005.

G. Kimnach, "Design of a Power Management and Distribution System for a Thermionic-Diode Powered Spacecraft," *IECEC-96 Energy Conversion and Engineering Conference*, vol. 1, pp. 529-533, Aug 1996.

C. Knospe, "PID Control," *IEEE Control Systems Magazine*, vol. 26, pp. 30-31, Jan 2006.

R. Kulhavy, J. Lu, and T. Samad, "Emerging Technologies for Enterprise Optimization in Process Industries," *Sixth International Chemical Process Control Meeting (CPC 6)*, Tucson, Arizona, Jan 2001.

A. Kusiak, A. Burns, S. Shah, and N. Novotny, "Detection of Events Causing Pluggage of a Coal-Fired Boiler: A Data Mining Approach," *Combustion Science and Technology*, vol. 177, pp. 2327-2348, 2005.

A. Kusiak and Z. Song, "Combustion Efficiency Optimization and Virtual Testing: A Data-Mining Approach," *IEEE Transactions on Industrial Informatics*, pp. 176-184, Aug 2006.

KySat Group, at web site http://www.kysat.com, March 2008.

C.-C. Lin, H. Peng, J.-M. Kang, and J. Grizzle, "Power Management Strategy for a Parallel Hybrid Electric Truck," *IEEE Transactions on Control Systems Technology*, vol. 11, no. 6, pp. 839–849, 2003.

P. C. K. Luk and L. C. Rosario, "Towards a Negotiation-Based Multi-Agent Power Management System for Electric Vehicles," *Proceedings of the International*

Conference on Machine Learning and Cybernetics, vol. 1, pp. 410-416, Aug 2005.

P. A. March, "Quantifying the Maintenance Costs Associated with Variable Operating Conditions," *Waterpower XIII Conference*, Buffalo, NY, pp. 98-107, Jul. 2003.

R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, Chp. 11, Object Mentor web site, http://www.objectmentor.com/resources/articles/dip.pdf, Oct 2002.

M. McVay and P.D. Patterson, "Illinois Power's Online Dynamic Optimization of Cyclone Boilers for Efficiency and Emissions Improvement," *Int'l Joint Power Generation Conference*, Baltimore, Maryland, Aug 1998.

G. Miller, K. Hall, W. Willis, and W. Pless, "The Evolution of Powertrain Microcontrollers and its Impact on Development Processes and Tools," *Proceedings of the 1998 International Congress on Transportation Electronics*, pp. 423-435, Oct 1998.

D. Morrison, "Is it Time to Replace PID?" *InTech Magazine*, Mar 2005.

NeuCo company web site http://www.neuco.net, Feb 2008.

M. Obland, D. M. Klumpar, S. Kirn, G. Hunyadi, S. Jepsen, and B. Larsen, "Power Subsystem Design for the Montana EaRth Orbiting Pico-Explorer (MEROPE) CubeSat-class Satellite," *IEEE Aerospace Conference Proceedings*, vol. 1, pp. 1.465-1.472, 2002.

T. Ogilvie, E. Swidenbank, and B. W. Hogg, "Use of Data Mining Techniques in the Performance Monitoring and Optimization of a Thermal Power Plant," *Proc. Inst. Elect. Eng. Colloq. Knowledge Discovery and Data Mining*, pp. 7/1-7/4, 1998.

G. A. Oluwande, "Exploitation of advanced control techniques in power generation," *Computing and Control Engineering Journal*, vol. 12, num. 2, pp. 63–67, April 2001.

K. Paul Jr. et al, *The Guide to Hydropower Mechanical Design.* Kansas City, MO, HCI Publications, pp. 3.1-3.8, 1996.

Pavilion Technologies company web site at http://www.pavtech.com, Mar 2008.

Pegasus Technologies was acquired by NeuCo in 2006. Details are at the web site, http://www.neuco.net/pdfs/NeuCo acquires pegasus 5-16-06.pdf, 2006.

R.-E. Precup, Z. Preitl, and S. Kilyeni, "Fuzzy Control Solution for Hydro Turbine Generators," *International Conference on Control and Automation*, pp. 83-88, Jun 2005.

Company web site for Pumpkin Inc, http://www.pumpkininc.com, April 2008.

Pumpkin Inc, "Building a Salvo Application with IAR's MSP430 C Compiler and Embedded Workbench IDE," found at http://www.pumpkininc.com, April 2008.

Pumpkin Inc, "Salvo Compiler Reference Manual – IAR MSP430C," found at http://www.pumpkininc.com, April 2008.

Pumpkin Inc, "Salvo User Manual v.4.1.2," found at http://www.pumpkininc.com, April 2008.

B. J. Radl, "Advanced Control and Operating Strategies for Power Generating Companies," Net Power Solutions, LLC.

S. F. Railsback and D. A. Dixon, "Individual-Based Fish Models: Ready for Business in Hydro Licensing", *Waterpower XIII Conference*, Buffalo, NY, pp. 90-97, Jul 2003.

G. Ramond, D. Dumur, A. Libaux, and P. Boucher, "Direct Adaptive Predictive Control of an Hydro-Electric Plant," *Proceedings of the International Conference on Control Applications*, pp. 606-611, Sep 2001.

J. Schindall, "The Charge of the Ultra – Capacitors," *IEEE Spectrum*, http://spectrum.ieee.org/nov07/5636, Nov 2007.

R. Schupbach and J. Balda, "The Role of Ultracapacitors in an Energy Storage unit for Vehicle Power Management", in *Proc. of 2003 IEEE Vehicular Technology Conference*, Orlando, pp. 3236-3240, 2003.

M. Smith and M. Elbs, "Towards a More Efficient Approach to Automotive Embedded Control System Development," *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pp. 219-224, August 1999.

J. Stallings, "Power Plant Optimization Guidelines," *Report TR-110718, Electric Power Research Institute*, Dec 1998.

J. Stallings, "Proceedings: Second Annual EPRI Workshop on Power Plant Optimization," *Report TR-111316, Electric Power Research Institute*, St Louis, MO, Sep 1998.

Company web site for Texas Instruments, http://www.ti.com, April 2008.

M. Thomsen, "Michael's List of CubeSat Satellite Missions," at web site http://mtech.dk/thomsen/space/cubesat.php, 2008.

L. Tolbert, H. Qi, and F. Peng, "Scalable Multi-Agent System for Real-Time Electric Power Management," *Power Engineering Society Summer Meeting*, vol. 3, pp. 1676-1679, 2001.

Ultramax company web site http://www.umaxcorp.com, Feb 2008.

A. Vahidi and W. Greenwell, "A Decentralized Model Predictive Approach to Power Management of a Fuel Cell-Ultracapacitor Hybrid," *American Control Conference*, pp. 5431-5437, July 2007.

A. Vahidi, A. Stefanopoulou, and H. Peng, "Current Management in a Hybrid Fuel Cell Power System: A Model Predictive Control Approach," *IEEE Transactions on Control Systems Technology*, vol. 14, pp. 1047-1057, 2006.

E. Vera and W. Kinsner, "Autonomous Power Management System for a Small Satellite," *IEEE WESCANEX 95 Communications, Power, and Computing Conference*, vol. 2, pp. 312-317, May 1995.

V. Viswanathan, V. Krishnan, and L. H. Tsoukalas, "Novel AI Approaches in Power Systems," *Proceedings on Information Intelligence and Systems*, pp. 275-280, 1999.

V. Vishwanathan, J. McCalley, and V. Honavar, "A Multiagent System Infrastructure and Negotiation Framework for Electric Power Systems," *IEEE Porto Power Tech Conference*, Porto Portugal, 2001.

X. Z. Wang, "Data Mining and Knowledge Discovery for Process Monitoring and Control," Springer, London, 1999.

M. Wildberger and M. Amin, "Simulator for Electric Power Industry Agents (SEPIA)," *Report TR-112816, Electric Power Research Institute*, Nov 1999.

M. Woodridge, "Agent-based Software Engineering," *Proceedings Instr. and Elec. Engineering*, vol. 144, pp. 26-37, 1997.

Z. Yongqin, W. Xudong, and Z. Meilan, "The Research and Realization for Passenger Car CAN Bus," *First International Forum on Strategic Technology*, pp. 244-247, Oct 2006.

Z. Zhang and X. Yuan, "Research on Hydro Electric Generating unit Controller Based on Fuzzy Neural Network," *Proceedings of the 6th World Congress on Intelligent Control and Automation*, pp. 6559-6563, June 2006.

X.-Y. Zhang and M.-G. Zhang, "An Adaptive Fuzzy PID Control of Hydro-Turbine Governor," *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics*, pp. 325-329, Aug 2006.

CURRICULUM VITAE

NAME: James Christopher Foreman

ADDRESS: Department of Computer Science and Engineering
Speed Scientific School
University of Louisville
Louisville, Ky. 40292

DOB: Valdosta, Ga – June 1968

EDUCATION: B.S., Electrical Engineering
University of Louisville
August 1990

MENG, Electrical Engineering
University of Louisville
August 1996

Ph.D., Computer Science and Engineering
University of Louisville
August 2008

PROFESSIONAL SOCIETIES:

- Institute of Electrical and Electronics Engineers

PUBLICATIONS:

- K. M. Walsh, T. R. Hanley, W. K. Pitts, M. Crain, J. Cole, D. Hensel, J. Hernandez, and C. Foreman, "Development of a New Microfabrication/MEMS Course at the University of Louisville," *Proceedings of the 13th Biennial IEEE University Government Industry Microelectronics Symposium*, Minneapolis, MN, pp. 26-32, June 1999.
- C. Foreman and R. Ragade, "Coordinated Optimization at a Hydro Generating Plant by Software Agents," *IEEE Transactions on Control Systems Technology*, accepted for future publication, pp. 1-10, Mar 2008.

BOOKS:

- C. Foreman, "Development of Wavelength Division Multiplexed Fiber Optic Communication Link," MENG Thesis, 1996.