

**AN AUTOMATED PATTERN-BASED SYSTEM FOR
REAL-TIME PROCESS MONITORING**

By

ROMAN ALEXANDROVICH SHAPOVALOV

Diplom, Chemical Cybernetics

Mendeleyev University of Chemical Technology of Russia

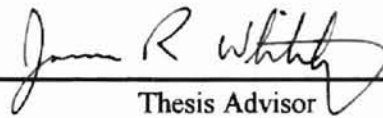
Moscow, Russian Federation

1996

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1999

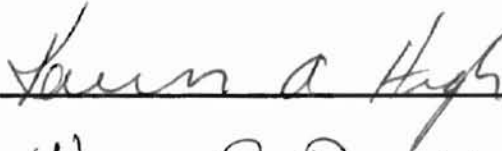
**AN AUTOMATED PATTERN-BASED SYSTEM FOR
REAL-TIME PROCESS MONITORING**

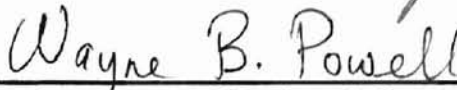
Thesis approved:



Thesis Advisor







Dean of Graduate College

ACKNOWLEDGMENTS

I would like to take this opportunity to thank the School of Chemical Engineering, Oklahoma State University for providing me with financial support.

I would like to thank Dr. James R. Whiteley for being my advisor, and wish to acknowledge his constant support and guidance throughout my stay at OSU. In addition, a large note of appreciation goes to Dr. Russell Rhinehart and Dr. Karen High for serving on my thesis committee.

I would like to thank Bruce Colgate, Loy Morrison, Julia Rumsey at Phillips Petroleum Research Center in Bartlesville, Oklahoma, and Sharad Bhartiya at Oklahoma State University for their valuable help, comments, and suggestions.

A personal note of gratitude goes to my friends and family in Moscow, Russia for their moral support.

TABLE OF CONTENTS

Chapter	Page
1. Introduction.....	1
2. Pattern-based and generic methods for fault diagnosis.....	3
2.1 Introduction to automated pattern-based fault diagnosis.....	3
2.2 Pattern-based methods for automated fault diagnosis.....	6
2.2.1 Multivariate statistical methods.....	7
2.2.2 Probability distribution-based methods.....	9
2.2.3 Artificial neural network-based methods.....	10
2.2.4 Selection of a pattern-based method for automated fault monitoring.....	17
2.3 Pattern generation.....	18
2.4 Computer systems for automated fault diagnosis.....	21
2.5 Chapter summary.....	25
3. Conceptual design of a generic system for pattern-based real-time process monitoring.....	26
3.1 Introduction.....	26
3.2 Basic concepts of generic pattern-based monitoring system design.....	27
3.2.1 Monitoring different processes by different monitoring applications with a single control center.....	28
3.2.2 Simultaneous monitoring of several different processes.....	31
3.2.3 Quick and easy generation of new monitoring applications.....	33
3.2.4 Section summary.....	35
3.3 Real-time programming techniques for generic pattern-based monitoring.....	35
3.3.1 Multithreading.....	36

Chapter	Page
3.3.2 Dynamic linking of monitoring applications at run time.....	38
3.3.3 Real-time data acquisition techniques.....	40
3.3.4 Section summary.....	43
3.4 Chapter summary.....	43
4. Implementation of a generic pattern-based real-time monitoring system.....	45
4.1 System design.....	45
4.2 Implementation of real-time monitoring.....	47
4.2.1 Retrieval of historical data by Historical Data Acquisition Utility.....	49
4.2.2 Real-time handling of monitoring data and scheduling execution of monitoring applications.....	50
4.2.3 Running monitoring applications.....	57
4.2.4 Interaction with the user during real-time monitoring.....	58
4.2.5 Section summary.....	61
4.3 Developing monitoring applications for Generic Real Time Monitoring System.....	64
4.3.1 Developing new executable code for monitoring applications from templates.....	64
4.3.2 Configuring monitoring applications with Monitoring Application Editor.....	65
4.3.3 Testing monitoring application performance on recorded data with Monitoring Data Player.....	69
4.3.4 Section summary.....	72
4.4 Chapter summary.....	72
5. Conclusions and future work.....	73
5.1 Conclusions.....	73
5.2 Future work.....	74
Bibliography.....	76

LIST OF FIGURES

Figure	Page
2.1 Hierarchy of methods for automated fault diagnosis.....	4
2.2 An example of an RBF neural network for fault diagnosis.....	13
2.3. Extraction of a pattern for classification from plant signatures using discrete wavelet transforms.....	19
3.1 Design of a generic system for real-time pattern-based monitoring.....	28
3.2 An example of scheduling calculation of monitored conditions for different processes.....	32
3.3 Building a monitoring application.....	34
3.4 Run-time dynamic linking of monitoring applications.	41
4.1 Hierarchical structure of the Generic Real-Time Monitoring System (GRTMS).....	46
4.2 Data flowchart for real-time monitoring with the GRTMM.....	48
4.3 Application Data Queue (ADQ).....	53
4.4 Illustration of updating a monitoring application input data file.....	56
4.5 The main window of the GRTMS user interface.....	59
4.6 GRTMM configuration selection screen.....	60
4.7 GRTMM Taskbar window.....	62
4.8 GRTMM Trendpad window.....	63
4.9 The main window of the Monitoring Application Editor.....	67
4.10 The window for editing parameters of monitored variables.....	68
4.11 The window for setting the order of smoothened trends in smoothened data file.....	68
4.12 The window for setting parameters of displaying monitored variable trends.....	69
4.13 The main window for the Monitoring Data Player.....	71

CHAPTER 1

INTRODUCTION

The efficiency of automated supervision of chemical processes can be significantly improved through the development of generic multitasking computer systems for pattern-based real-time fault monitoring. Pattern-based methods for fault diagnosis, implemented as computer programs, can be quickly and easily taught to recognize malfunctions of process equipment without long and expensive first-principle modeling or manual generation of knowledge bases. Real-time monitoring can provide a quick identification of equipment failure and can initiate prompt actions to avoid the serious consequences that may be incurred by these failures. Generic and multitasking features greatly facilitate the job of configuring the system to monitor new chemical processes and allow the simultaneous monitoring of several different processes using different user-defined pattern-based fault identification methods within the framework of the same monitoring system. A process monitoring system that successfully combines pattern-based, real-time, generic and multitasking approaches to automated fault diagnosis will be a great asset to the process industry.

A number of real-time computer systems for automated fault diagnosis in industry have appeared in the recent years. However, almost none of them, to the author's knowledge, are designed for pattern-based monitoring, are fully generic, and multitasking at the same time. Therefore, there is a need for a generic pattern-based real-time multitasking monitoring system to fill this gap.

The author of this thesis has created a generic pattern-based, real-time multitasking system for automated fault monitoring in the process industry. This system is capable of monitoring the operating conditions of several industrial processes simultaneously. Each process is monitored by a custom monitoring application. The proposed system has tools for configuring the existing monitoring applications to enable them to monitor new processes and for designing completely new

monitoring applications that implement novel, user-defined pattern-based methods for automated fault detection and diagnosis. The proposed system is user-friendly and allows the user to create his or her own monitoring application for any process monitoring task, based on the user's expertise. The proposed system can serve as a building block for other more advanced process monitoring systems.

This work gives a brief summary of the existing generic and pattern-based methods for automated process supervision and a detailed description of a new generic pattern-based, real-time multitasking monitoring system. Chapter 2 describes various pattern-based methods for real-time fault identification and process monitoring systems implemented in industry in the recent years. Chapter 3 presents the key concepts about the generic multitasking design of the novel monitoring system and the programming techniques employed for automated generic real-time process monitoring. Chapter 4 focuses on the details of system implementation and describes the components of the new monitoring system. Chapter 5 concludes the thesis and offers directions for future work. The operating manual for the proposed system is documented in a separate technical report [Shapovalov and Whiteley, 1999]. This documentation is also embedded in the monitoring system code as a Microsoft® Windows® help file. The reader is frequently referred to this report for specific details.

The work presented is pioneering in the area of computer-aided process monitoring. Although numerous real-time computer systems for process monitoring have been developed, almost all of these systems offer only a limited choice of the method for fault diagnosis, and none of these system are as flexible and easy to use as the one described in this thesis. The new features provided the proposed monitoring system will enable broad application of automated pattern-based fault monitoring, improve the efficiency of industrial process supervision, and enhance safety in the process industry.

PATTERN-BASED AND GENERIC METHODS FOR FAULT DIAGNOSIS**2.1 Introduction to automated pattern-based fault diagnosis**

The efficiency of supervision of chemical processes can be significantly increased when computers perform the task of fault detection. Just like human operators, computers can be taught to detect process malfunctions. At the same time, computers can be more unfailing and productive than humans at performing this job. Unlike human beings, computers do not fall asleep, do not get distracted, and are less prone to making mistakes. Due to the recent advances in hardware manufacturing, a program running on a typical desktop can monitor dozens processes simultaneously. Finally, the purchase price and maintenance costs of a computer are many times lower than the salary of a plant operator. Therefore, computerization of process monitoring is a good solution to reduce operating costs and increase safety in the modern chemical industry.

Automated computer-aided fault monitoring is a complex task that can be described in brief as follows [Lee, 1995]. Process data is continuously collected from sensors installed on process equipment and programmable logic controllers. A special computer program uses this data to detect occurrence of a fault, i.e. the inability of the monitored system to correctly perform the expected functions. If a fault occurrence is detected, the recently collected plant data, called plant signatures, may be analyzed by a computer program to diagnose a particular type of fault. Once the type of fault has been determined, either the plant operation strategy is changed or the operation of the plant is completely stopped to rectify the fault. Very often, especially in the process industry, fault detection and fault diagnosis are merged into one operation. This sequence of actions is called automated process (or fault) monitoring.

There are a great number of various methods for automated computer-aided fault diagnosis. A recent survey on automated process monitoring [Sharif and Grosvenor, 1998] briefly

describes eight of the most common fault diagnostic techniques: logical method, algorithmic method, functional systems documentation, expert systems, multivariate statistical methods, model-based approaches, artificial neural networks, and Petri nets. Each of these methods is based either on knowledge engineering and logical inference, first-principle process modeling, or on classification of patterns (pattern-based) obtained from the current plant signatures, i.e. the trends of the sensors measuring the monitored process variables. A hierarchy of computer-aided methods for fault diagnosis is presented on Figure 2.1.

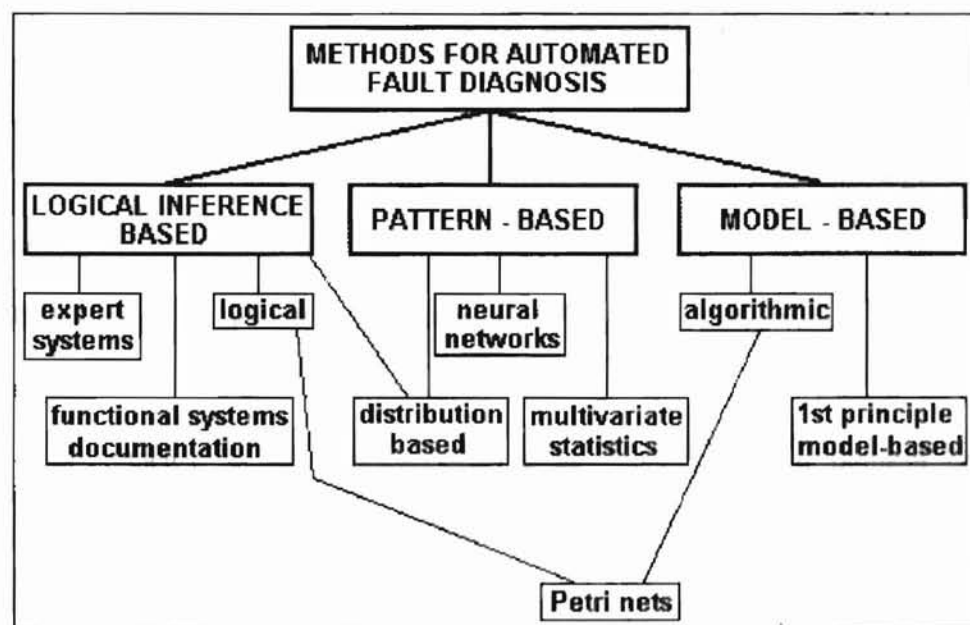


Figure 2.1. Hierarchy of methods for automated fault diagnosis.

The scope of application of each group of methods for automated fault monitoring is limited. Computer-aided monitoring of faults in chemical processes is usually a very nontrivial problem. Usually fault occurrence in an industrial process is an extremely complicated and highly nonlinear function of monitored variables. Very often this function cannot be derived from a first principle-based process model or identified using conventional regression techniques. Moreover,

the number of inputs to the function is normally quite large and the inputs are frequently correlated. Therefore, fault detection techniques based on first principle modeling are infrequently applied for monitoring chemical processes.

The methods for automated fault monitoring based on knowledge engineering and logical inference (usually expert systems) have been extensively used to diagnose manufacturing equipment, however, it is difficult to apply these methods for the monitoring of chemical processes. The reason is that the design of knowledge engineering and logical inference-based monitoring systems requires extensive knowledge about the monitored processes, as well as the nature and causes of the diagnosed faults. For chemical processes, which are usually very complex, this knowledge is often unavailable in explicit form. Besides that, the process of generating knowledge bases for monitoring based on logical inference is often very tedious. As a result, logical inference-based systems have not found a wide application for the monitoring of chemical processes.

Pattern-based techniques are the group of real-time monitoring methods most suitable for diagnosis of faults in chemical processes. Pattern-based methods in general solve pattern classification problems. In fault monitoring, the pattern classification problem consists of classifying the current monitored condition (state of operation) of the process as normal or as a certain fault using a pattern extracted from the latest trends of the monitored variables (latest plant signatures). Essentially, pattern classification consists of assigning class membership to pattern vectors using a mapping from the space of patterns to a finite set of classes (operation states in the case of fault monitoring). This mapping is "taught" using a set of training patterns generated from the historical trends of the monitored variables recorded at various operating states of the monitored process. A first-principle model of the monitored process is not required (pattern-based techniques belong to the family of the so-called "black box" approaches). This feature makes pattern-based methods especially attractive for fault monitoring in the process industry.

Computer-aided automatic fault diagnosis is a rapidly developing field. Recent years have seen an explosive growth in the number of methods, including pattern-based techniques, which have been developed and tested for automated monitoring of many industrial processes. Different pattern-based methods have a variety of advantages and shortcomings. Successful solution of any process-monitoring problem is possible only with the selection of a suitable method for fault diagnosis. The next section discusses various pattern-based techniques successfully applied for real-time diagnosis of faults in the process industries.

2.2 Pattern-based methods for automated fault diagnosis

There exist a great number of pattern classification methods. However, few are applicable for real-time fault diagnosis in the process industry. First, an applicable classifier should be able to recognize data patterns with very highly interrelated parameters that have nonlinear relationships with the classified monitored conditions (operation states). The classification must be performed with a very high degree of accuracy with noisy data as input. It is absolutely unacceptable to classify a fault as a normal operating condition, even if some fault patterns were not available at the time when the classifier was trained. Furthermore, a classification must be performed within a limited time because fault occurrence must be detected as soon as possible after the problem arises. It is very desirable that the classifier be capable of incremental learning during the monitoring as patterns from new regions of the state space become available. Finally, the classifier should be easy to use and maintain. Only the methods that comply with all of these requirements can generally be used in pattern-based monitoring.

Pattern-based techniques can be divided into three subgroups (see Figure 2.1): multivariate statistical methods, methods using probability distribution, and artificial neural network-based

methods. This section describes and compares various pattern-based techniques and groups of techniques developed for process monitoring in the recent years.

2.2.1 Multivariate statistical methods

Multivariate statistical methods typically project input data into lower dimensional spaces that contain all the relevant information in possibly as few as two or three latent variables that are a linear combination of the original variables. The best combinations are found using process historical data. Linear regression methods are then applied to classify the projected pattern vector.

There are two basic techniques for multivariate statistical classification: principal component analysis (PCA) and projection to latent structures (PLS). Both techniques have been tested for fault identification [Martin and Morris, 1995] in the process industry.

Principal component analysis (PCA) linearly decomposes the input pattern vector \mathbf{X} of dimension n into m orthogonal principal components t_i :

$$\mathbf{X} = \mathbf{TP}^T = \sum_{i=1}^m t_i p_i^T \quad (1)$$

where p_i is the direction of i -th greatest variability in \mathbf{X} (essentially p_i is an i -th eigenvector of the covariance matrix of \mathbf{X}). Extensions of PCA and PLS called Multiway Partial Component Analysis (MPCA) and Multiway Projection to Latent Structures (MPLS) showed good performance in monitoring a continuous stirred tank reactor [Martin and Morris, 1995] and an emulsion polymerization batch process [Neogi and Schlags, 1998].

Projection to latent structures (PLS) [Martin and Morris, 1995] can be used to simultaneously monitor several process conditions (operation states) using the same classified pattern \mathbf{X} . Unlike PCA, PLS linearly decomposes both the pattern vector \mathbf{X} and the vector of operation states (monitored conditions) \mathbf{Y} into the most highly correlated orthogonal components of \mathbf{X} and \mathbf{Y} generating a biased linear regression between \mathbf{X} and \mathbf{Y} .

Experiments indicated [Martin et al., 1996] that multivariate statistical methods show an unsurpassed performance on correctly scaled data with linearly correlated variables. Multivariate statistical methods can predict severity of faults in the monitored processes. At the same time, an incorrect scaling degrades the performance of multivariate statistical methods drastically. Unfortunately, there is no completely reliable scaling technique, especially when the variables are the measurements of quantities of different nature, e.g., pressure and temperature. If there are large differences between the variances of the elements of the classified pattern X , those pattern elements, whose variances are large, will tend to dominate the first few principle components that play the key role in pattern classification. However, the variables with high variances are not necessarily of prime importance in detecting process malfunctions and, as a result, an incorrect scaling can lead to an incorrect classification.

Another problem with multivariate statistical methods is that these techniques are not effective in the analysis of data from highly nonlinear processes. To deal with nonlinear processes, a nonlinear PCA method was introduced [Martin et al., 1996]. The primary difference from the conventional PCA technique is the introduction of nonlinear mappings (most often sigmoid functions) between the original and reduced spaces of pattern variables. In this method the nonlinear principal components are generated by a neural network, therefore, nonlinear PCA is a combined neural network-multivariate statistical method. A comparative study of detecting faults in the operation of a batch reactor by linear and nonlinear PCA [Martin et al, 1996] showed that the nonlinear PCA performs significantly better. With linear PCA three principle components were explaining 67% of the variability in the pattern vector X , whereas with the nonlinear PCA three principle components explained as much as 90% of the data variance.

2.2.2 Probability distribution-based methods

Probability distribution-based methods use distributions of class membership probabilities in the space of the variables of the classified pattern. The probability distribution-based classification technique called F-curve-improved Bayesian method was developed, tested, and showed an acceptable performance for real-time automated monitoring of the operation of a continuous stirred tank reactor [Won and Modarres, 1998]. Probability distribution methods can be applied to upgrade other diagnostic methods to detect not only occurrence of a failure, but also the severity of the failure. In general, it is a hybrid pattern-based and inference-based method.

The classical Bayesian method is a statistical approach to reasoning under uncertainty that calculates the probability of fault existence. The method uses Bayes' formula relating the conditional (a priori) and unconditional (posterior) probabilities to calculate the probability of fault existence:

$$P(h|X) = P(h) \prod_{i=1}^n \frac{P(x_i|h)}{P(x_i)} \quad (2)$$

where X is the vector of the observed events with n elements denoted x_i (in the pattern-based fault diagnosis X is the pattern vector), h is the occurrence of a specific fault, P is a probability of a certain event ($P(a)$ means the unconditional probability of event a , $P(a|b)$ means the conditional probability of event a in case event b has occurred). Being a product of the observed events, Bayes' formula is valid only if the events of vector X are independent of each other. In reality, this is seldom the case with patterns extracted from plant signatures. To deal with interdependent events (elements of the pattern vector in our case) the F-curve improved Bayesian method corrects this formula with dependency coefficients. Unfortunately, specification of the dependency coefficients is not straightforward. This represents a major drawback to the widespread application of this method.

2.2.3 Artificial neural network-based methods

Artificial neural network-based methods are the most commonly used techniques for pattern-based fault monitoring in the process industry [Sharif and Grosvenor, 1998]. The most important advantage of neural networks is the ability to model nonlinear dependencies with highly correlated inputs. At present most of the research on the methods for pattern-based fault diagnosis is focused on artificial neural networks.

A number of different types of neural network have been developed and some of them have established a good reputation as a tool for process monitoring. The following types of neural network are known to have been applied for pattern-based process monitoring: MLP (Multilayer Perceptron), GMDH (Group Method and Data Handling) network, RBF (Radial Basis Function) network, WSBF (Wavelet Sigmoid Basis Function) network, an RBF-type network with ellipsoidal activation functions, and a family of ART (Adaptive Resonance Theory) networks, such as ART2 and Fuzzy ARTMAP. In some cases, ensembles of networks or neural-fuzzy classifiers have been applied for automated fault diagnosis. Below is a brief review of the neural networks tested for pattern-based process monitoring.

Perceptron-based networks were the first group of neural networks applied for fault diagnosis. The most widely used neural network of this group is the multilayer perceptron (MLP). An MLP network usually consists of three layers with neurons of perceptron type in the hidden layer and simple summation neurons in the output layer. The vector of outputs from the hidden layer is calculated the following way:

$$\mathbf{a} = T(\mathbf{W}\mathbf{p} + \mathbf{b}) \quad (3)$$

where \mathbf{W} is the weight matrix, \mathbf{p} is input vector (pattern), \mathbf{b} is the bias vector, and $T()$ is a transfer function.

The most popular technique for training an MLP neural network is back-propagation [M.T.Hagan et al., 1996]. Fault diagnosis in a distillation-reactor system using an MLP network was one of the first reported applications of neural networks in automated process monitoring [Hoskins and Himmelblau, 1988].

The advantage of MLP networks is the ability to perform classification within a very short time because of the low computational complexity of the classification. However, it takes a large amount of historical data and computer time to train an MLP network. Furthermore, MLP may misclassify patterns in the regions where no training samples are available. One more problem with MLPs consists in finding the optimal number of neurons in the hidden layer. So far, this problem has not been solved in general, and the number of neurons has to be determined in an ad hoc manner by the user.

An extension of MLP called the GMDH (Group Method and Data Handling) network has been tested to diagnose a gravimetric dustimeter [Kobricz and Kuš, 1998]. This type of network calculates the output for each neuron in a hidden layer using a polynomial of input variables (e.g. $a = \text{logsig}(\mathbf{W}_1 \mathbf{p}^2 + \mathbf{W}_2 \mathbf{p} + \mathbf{b})$ where \mathbf{W}_1 and \mathbf{W}_2 are weight matrices, \mathbf{p} is input pattern, \mathbf{p}^2 is a cross-product of two identical input pattern vectors, \mathbf{b} is the bias vector, and $\text{logsig}()$ is the log-sigmoid transfer function). The GMDH network can be trained to minimize the following objective function:

$$n_{dev} = \frac{\sum_{i=1}^{n_B} (y_i^* - y_i)^2}{\sum_{i=1}^{n_B} y_i^2} \quad (4)$$

where n_{dev} is regularity, n_B is the size of the testing data set, y_i^* is a computed output and y_i is the measured output.

The GMDH is a new type of neural network, and its performance in process monitoring in comparison with other methods is unknown. It is also unclear how to select the type of polynomial for each neuron. Similar to the MLP network, it is not clear how to select the correct number of neurons for a GMDH neural network.

The second group of neural networks applied for process monitoring is RBF (Radial Basis Function) type networks. An RBF network usually has three layers: an input layer with no data processing, a hidden layer with RBF neurons, and an output layer containing perceptron-type neurons (usually with unit weights and without biases). Each neuron in the output layer corresponds to one monitored condition. In general, the output a from a classical RBF neuron with the Gaussian activation function is calculated using the following relationship:

$$a = \exp[-(\mathbf{p} - \mathbf{w})^T \mathbf{S}^{-1} (\mathbf{p} - \mathbf{w})] \quad (5)$$

where \mathbf{p} is input pattern, \mathbf{w} is the weight vector for the neuron, and \mathbf{S} is a normalization matrix.

The schematics of an RBF network for pattern classification are presented in Figure 2.2. The input pattern is supplied to each RBF neuron in the hidden layer. Each hidden neuron calculates the “proximity” of the input pattern to the neuron center using a radial-basis function and outputs a signal corresponding to the calculated proximity. The output from each RBF neuron is passed to one output perceptron with a positive linear transfer function. The calculated operating state will be the one that corresponds to the output perceptron that fires a non-zero (or, sometimes, the strongest) signal.

RBF-type neural networks have been tested for automated fault diagnosis in many different chemical processes, for example, in monitoring the operation of a continuous stirred tank reactor with a recycle [Leonard and Kramer, 1991]. Many techniques for training RBF networks automatically select the number of hidden neurons [Reilly et al, 1982] and support incremental



4

5

training. Incremental training improves prediction accuracy with time. However, RBF networks are not good at the classification of noisy data.

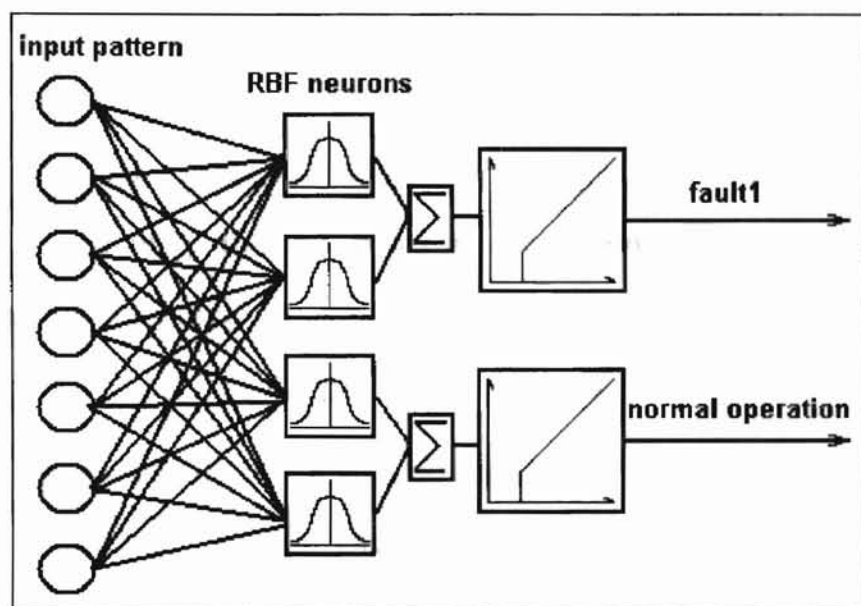


Figure 2.2. An example of an RBF neural network for fault diagnosis.

An RBF-type neural network called WSBFN (Wavelet Sigmoid-Basis Function Network) was proposed recently [Zhao et al., 1998] to improve the accuracy of pattern classification. WSBFN is similar to the classical RBF network, except the hidden layer employs wavelet-sigmoid neurons. The i -th element of WSBFN output vector is calculated as follows:

$$y_i = S \left\{ \sum_{j=1}^{n_s} w_{ij} \exp \left[- \left(\sum_{l=1}^m \frac{x_l - b_{jl}}{a_{jl}} \right)^2 \right] + \sum_{j=n_s+1}^{n_h} w_{ij} \sum_{l=1}^m \frac{x_l - b_{jl}}{a_{jl}} \exp \left[- \sum_{l=1}^m \left(\frac{x_l - b_{jl}}{a_{jl}} \right)^2 \right] \right\} \quad (6)$$

where $S()$ is the sigmoid function, n_s is the number of scaling hidden units, n_h is the total number of hidden units, m is the number of inputs to the network, w_{ij} is the weight for i -th output and j -th hidden unit, x_l is l -th element of the input vector, and a_{jl} and b_{jl} are the scaling and translation factors, respectively, for j -th hidden unit and l -th input. A heuristic algorithm for training the WSBFN was developed [Zhao et al., 1998]. The WSBFN was tested for monitoring the operation

of a hydrocarbon-cracking reactor [Zhao et al., 1998] and showed better performance (100% correct classification of the monitored conditions) than the MLP that was previously used to monitor the operation of the same reactor.

To avoid extrapolation by an RBF network in the regions with no training patterns, a neural network with ellipsoidal activation function was proposed [Kavuri and Venkatasubramanian, 1993]. This network has three layers and outputs of each neuron in the hidden layer are calculated using:

$$a = (\mathbf{p} - \mathbf{m})^T (\mathbf{D}^T \mathbf{D})^{-1} (\mathbf{p} - \mathbf{m}) + 1 \quad (7)$$

where \mathbf{p} is the input vector, a is the neuron output signal, \mathbf{m} is the vector of the coordinates of the neuron center, and \mathbf{D} is the diagonal matrix consisting of the half-lengths of principal axes of the ellipsoid. The network was used to monitor a reactor-distillation system. The network was trained with a proprietary algorithm that randomly creates hidden neurons, splits them if they cover wrong patterns, and eliminates neurons if the sum of “activations” of the covered patterns is too small. The aim is to minimize the mean square error of the classification of the patterns on which the training is performed.

The problem with this network is a very long training time and a relatively long classification time because of the extremely large size of the resulting network. In their later work [Kavuri and Venkatasubramanian, 1994] the authors proposed to avoid the creation of an excessively large network by creating a smaller, individual network for each fault class.

The third group of neural networks studied for process monitoring utilizes Associative Resonance Theory (ART). A number of ART-based neural networks for recognition of analog patterns have been developed by S. Grossberg and his associates [Carpenter and Grossberg, 1987; Carpenter and Grossberg, 1990; Carpenter et al., 1991; Carpenter et al., 1992; Carpenter and Ross, 1995]. The design of these associative type networks was inspired by investigation of the

operation of the human brain. The principal building block for these networks is the so-called “leaky integrator” whose response n to an input p as a function of time t is

$$n(t) = n(0) \exp\left(-\frac{t}{\varepsilon}\right) + \frac{1}{\varepsilon} \int_0^t p(t - \tau) \exp\left(-\frac{\tau - t}{\varepsilon}\right) d\tau \quad (8)$$

where ε is the system time constant. The design of ART networks is very complex and lies outside the scope of this thesis.

Just like RBF networks, the networks from the ART family can be taught incrementally as new plant data becomes available. Similar to the network with an ellipsoidal activation function, ART neural networks will not classify a pattern in the region of state space where there were no training samples. ART neural networks can handle noisy patterns. The problems with ART neural networks are the complexity of their implementation, a relatively slow speed of pattern recognition, and the need to adjust certain network parameters manually.

Two members of the family of ART neural networks: ART2 [Carpenter and Grossberg, 1987] and Fuzzy ARTMAP [Carpenter et al., 1992] were tested for process monitoring. ART2 was tested for monitoring a simulated recycle reactor [Whitely and Davis, 1994]. The network determined the monitored condition with 100% accuracy. At about the same time an experiment was conducted where ART2, MLP, RBF with RCE learning, Cascade and Fuzzy ARTMAP neural networks were tested for monitoring the operation of a nuclear plant [Keyvan et al, 1993]. Fuzzy ARTMAP performed better than all the other tested neural networks in processing both crisp and noisy data. In fact, of all the neural networks tested for the classification of the nuclear plant data with a high level of noise, only Fuzzy ARTMAP provided acceptable performance (95% of all test patterns were classified correctly).

To increase the reliability of pattern classification during automated process monitoring, it has been proposed to combine several neural networks into a hybrid network. Reported

applications for monitoring a reciprocating compressor [Kotani et al., 1993] and a continuous stirred tank reactor cascade [Tsai and Chang, 1995] utilized two MLPs. In both test cases, the first MLP was used to extract patterns from the input data, and the second MLP was trained to classify the patterns obtained by the first network. Hybrid networks diagnosed faults with a higher robustness than a single MLP.

Hybrid neural networks of another proposed type [Sharkey and Sharkey, 1997] consist of several independent networks. These independent networks concurrently classify either patterns from several different sets of sensors or patterns obtained from the original pattern by different nonlinear transformations. The authors of this method claim that several independent neural networks can diagnose faults with a greater reliability than a single neural network.

Most pattern-based methods do not use heuristic rules explicitly. The classification algorithms are effectively "black boxes." This drawback can be overcome with the help of neural-fuzzy classifiers. The classifiers of this type are trained the same way as regular neural networks, but, in addition to that, the network weights are later converted into a set of fuzzy rules. With these fuzzy rules, the patterns are classified using a fuzzy inference engine. Unlike the weights of a neural network, the fuzzy rules can be interpreted, edited, and amended by the user, based on the user's experience.

The first fuzzy-neural classifiers were Fuzzy ARTMAP [Carpenter et al., 1992] and ANFIS [Jang, 1993]. A number of fuzzy-neural applications for fault diagnosis, that allow the user to edit and add new classification rules, have been developed, e.g., a fuzzy-neural system for monitoring the operation of an electric motor [Goode and Chow, 1993]. Several fuzzy-neural classifiers have been applied for fault diagnosis in the chemical industry [Zhang and Morris, 1994; Ozyurt and Kandel, 1996; Calado and Sa da Costa, 1998], however, unlike the fuzzy-neural system proposed by Goode and Chow, none of these applications allow the user to modify and amend the fuzzy rule base manually.

A pattern-based technique related to fuzzy-neural classifiers called Bayesian neural network was proposed for fault diagnosis [Kirsch and Kroschel, 1994]. The classification algorithm of the Bayesian neural network can be easily interpreted and modified by a human being. Unfortunately, the Bayesian neural network, as proposed by Kirsch and Kroschel, has never been applied for fault diagnosis in the industry because the training of the Bayesian neural network takes an unacceptably large amount of computer time.

2.2.4 Selection of a pattern-based method for automated fault monitoring

Selection of an appropriate pattern-based method for automated fault monitoring is difficult. There are no clearly defined rules how to perform the selection. To achieve the desired performance, several different pattern classification methods may have to be tried. In general, multivariate statistical methods are the best choice for the classification of small patterns generated from the trends of plant sensors that measure quantities of a similar nature (e.g., all monitored variables are temperatures). Large patterns with highly correlated variables and nonlinear relationships with the monitored operation state are probably classified best with Fuzzy ARTMAP.

In summary, there is no “universally best” classifier for solving any pattern-based monitoring problem. Moreover, the existing pattern recognition methods are far from full compliance with the requirements outlined in the first paragraph of Section 2.2. Considering this fact, a good monitoring system that can simultaneously monitor several processes should be able to run several classifiers of different type for monitoring different processes in the same monitoring session. The monitoring system should also allow the user to easily select, train and test different pattern-based methods, including those that had not existed or had not been implemented as a computer code at the time when the system shell was developed.

2.3 Pattern generation

Pattern-based monitoring techniques require a compact and crisp representation of sensor data in the patterns. Input patterns used for real-time pattern-based monitoring should be compact to minimize the computation time. Reducing sensor noise is also very critical for classifying the monitored operation states with a high degree of accuracy. Therefore, pattern generation techniques employed in pattern-based monitoring systems should be able to extract patterns of the minimal size necessary for the correct classification and to filter out the noise contained in the sensor trends.

Several different techniques of pattern extraction can be used for pattern-based monitoring. The most commonly used techniques are wavelet transforms, autoregressive modeling (AR), and hypothesis feedback modeling (HFM).

Discrete wavelet transforms have been proposed for smoothening input data for pattern-based process monitoring [Raghavan, 1995]. This approach represents the trended signal $f(t)$ as a sum of scaling $\alpha(t)$ and wavelet $\beta(t)$ functions

$$f(t) = \sum_k g_k \alpha_k(t) + \sum_k h_k \beta_k(t) \quad (9)$$

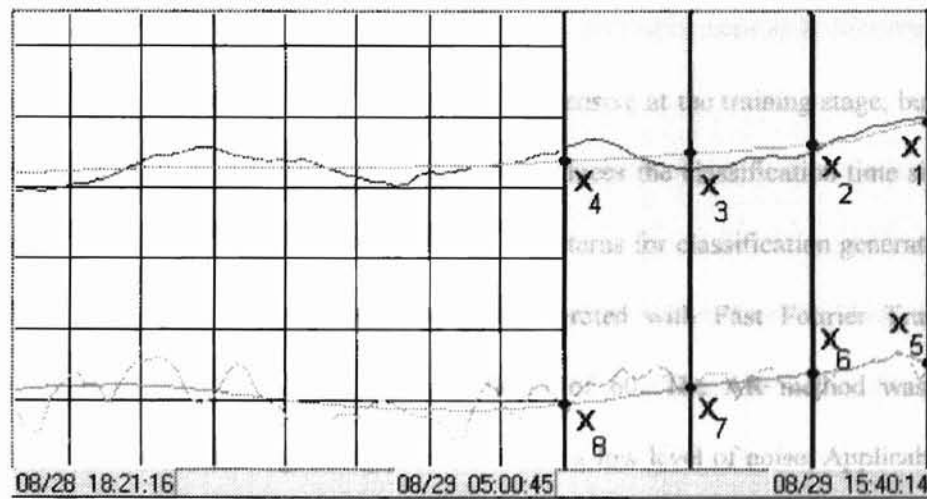
where g_k and h_k are decomposition coefficients. Scaling functions and wavelet functions are represented by dilation equations. The general form of a dilation equation is as follows:

$$\varphi(t) = \sum_k l_k \varphi(2t - k) \quad (10)$$

where l_k is a vector of filter coefficients. In discrete wavelet transforms $f(t)$, $\alpha(t)$ and $\beta(t)$ are discrete functions of time for use with sampled trends of the monitored variables.

Wavelet transforms provide variable frequency analysis capability and good time-frequency localization. At the same time, application of wavelet smoothening in real-time process monitoring is complicated by the need for the user to choose the transform parameters, for instance, filter coefficients.

A technique that uses a wavelet transform for pattern extraction from raw trends of monitored process variables has been proposed [Ganti, 1996]. This technique automatically selects all the coefficients of the wavelet transform along with choosing an optimal pattern time window (see Subsection 2.2.3 in [Shapovalov and Whiteley, 1999]) based on the characteristics of the monitored sensor trends. The smoothed trends are subsequently classified using a pattern-based method. This method for pattern generation is illustrated on Figure 2.3. In this example, the extracted pattern for classification contains four samples of the values of the smoothed monitored variables.



Extracted pattern for classification: $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$

Figure 2.3. Extraction of a pattern for classification from plant signatures using discrete wavelet transforms.

A similar method for pattern generation using wavelets was developed [Alexander and Gor, 1998]. The difference between this technique and that proposed by Ganti is that in the method by Alexander and Gor the wavelet transform parameters are selected in such a way that the level of discrimination between patterns corresponding to different monitored conditions is the highest.

A method for compact pattern extraction in pattern-based monitoring called autoregressive modeling (AR) has been proposed [Huang and Wang, 1996]. AR represents a sampled trend of a monitored variable as a time series:

$$X_k = \sum_{i=1}^p \Phi_i X_{k-i} + E_k \quad (11)$$

where X_k is a k -th element of the sample vector, Φ is a vector of AR parameters with elements Φ_i , p is the order of the AR model, and E_k is a residual. The order of the AR model is selected in such a way that for all the training samples the average final prediction error is minimized or the Akaike information criterion is maximized [Lin and Wang, 1993]. The final pattern used as input to the classifier is the vector Φ .

Autoregressive modeling is very computation-intensive at the training stage, but it leads to a reduction of the input pattern size and, as a result, reduces the classification time significantly. During the experiments [Huang and Wang, 1996] the patterns for classification generated with AR were smaller than the patterns for classification generated with Fast Fourier Transforms (a technique similar to wavelet transforms) by a factor of 60. The AR method was tested for monitoring faults in roller bearings using plant data with a low level of noise. Applicability of this method in the chemical industry where the patterns for classification are usually extracted from sensor trends with a high level of noise is unclear.

Another technique for extracting compact patterns from plant data is called hypothesis feedback modeling (HFM) [Farell and Roat, 1994]. HFM is a complex technique that consists of several steps. At the first step, a minimal set of independent variables is selected and a matrix of the sampled recent trends of these variables is formed. At the second step, the values of the selected variables are normalized and the precision of the selected variables is reduced by converting the values of the variables from the floating point to integer type. At the third step the average, mean deviation and numerical partial derivatives with respect to time are calculated for each normalized

trend. The resulting pattern is formed by the values calculated at step three for each normalized trend.

HFM was tested for monitoring a simulated continuous stirred tank reactor using an MLP network to recognize faults [Farell and Roat, 1994]. The performance of the method was generally satisfactory, but still well below the expected level. It is also unclear how good HFM is at generating patterns from noisy trends of monitored variables or from the trends smoothed with discrete wavelet transforms.

To imitate the process of extracting important qualitative features that describe the essential aspects of process behavior, a syntactic approach to pattern generation was developed [Rengaswamy and Venkatasubramanian, 1995]. The syntactic approach describes complex patterns using a small set of primitive patterns that indicate the presence or absence of faults. Rengaswamy and Venkatasubramanian proposed to identify the primitive patterns in plant signatures by an MLP neural network.

In addition to the lack of a single approach to pattern classification, there exist several techniques for generation of patterns from the trends of the monitored variables. None is best for every situation. Therefore, the user of a pattern-based method of fault diagnosis should be able to select, test, and apply several different techniques for generating patterns for classification.

2.4 Computer systems for automated fault diagnosis

A successful pattern-based system for fault monitoring in chemical industry must be flexible. The continuing advances in computing power provide the capability of real-time monitoring of many different processes on the same computer. The existing methods for pattern-based monitoring are far from perfection, with better methods under development. Therefore, a good monitoring system should be able to use the newest pattern-based techniques implemented as

modules within a general purpose monitoring shell. Furthermore, a good monitoring system should be able to run several different monitoring programs simultaneously. Finally, a monitoring system is especially valuable if it can serve as a building block for other more sophisticated process automation applications. A good pattern-based monitoring system should have all of these features which make the system fully generic, multitasking, and therefore capable of solving most real-time process monitoring problems in the process industry.

A number of computer systems for automated fault diagnosis in industry have been proposed in the recent years. However, almost none of these systems are designed for pattern-based monitoring, are fully generic, and multitasking at the same time.

One of these pattern-based monitoring systems is described in detail [Huang and Wang, 1996]. The system includes data acquisition, pattern generation, neural network, and fuzzy logic inference engine blocks. The system collects real-time data from plant sensors, generates patterns for classification using autoregressive modeling, classifies the patterns with a modified ARTMAP neural network and displays the classification results on the user interface. If the modified ARTMAP network cannot classify the current pattern, the fuzzy logic inference engine performs the classification.

The algorithms implemented in Huang and Wang's pattern-based monitoring system are very powerful. However, their system can monitor only one process. It is unclear if several copies of the system can simultaneously run on the same computer and monitor different processes. The system is not generic because the system components are hard-coded as a single module, and the user cannot change them (e.g. upgrade the modified ARTMAP network that performs the pattern classification) to other alternate components that perform the same operations.

A generic distributed system for real-time fault diagnosis was developed and tested for monitoring the operation of a paper mill [Rao et al., 1998]. The system consists of several independent modules for data acquisition, data calibration, data conversion, condition monitoring,

fault diagnosis, maintenance assistance, and includes on-line system operation and accident handling manuals. The monitored sensor data are acquired in real time, calibrated, converted, and input to a rule-based inference engine that determines if there is a fault in the monitored process. If a fault is detected, the diagnostic block determines the type of the fault, and the system suggests how to rectify the fault.

The described system is generic because it consists of blocks that perform different tasks, and each block can be replaced or upgraded. However, this system does not have the facilities that help the user in generating pattern-based fault diagnosis blocks. The system is also designed to monitor only one process.

A fully generic monitoring system was developed [Karsai et al., 1996] for model-based and inference-based monitoring. The system is designed to create custom standalone monitoring applications in a special programming environment. The environment allows the user to generate very complex monitoring applications that handle fault propagation. The user can program custom fault diagnosis models and even incorporate custom executable code, designed to diagnose faults and/or handle fault recovery, into the user-generated monitoring applications.

Unlike the monitoring systems described previously, this one can be used for generic, multitasking pattern-based real-time fault diagnosis. However, implementation of pattern-based monitoring using this system would be extremely complicated and inefficient for three reasons. First, the proposed system does not have special facilities for pattern-based applications. These facilities include the environment and standard templates that helps the user generate, train and test pattern-based monitoring applications. Second, the resulting monitoring applications run continuously and independently of each other without a kernel that schedules execution of each monitoring application in real time. If a large number of different processes are to be monitored using the proposed system, all the applications that monitor these processes must run in parallel. Running many monitoring applications in parallel is difficult because of hardware limitations

(primarily because the computer memory is always limited and capable of accommodating the executable code and data only for a limited number of applications). It also drastically slows down the operating system that may not be able to accomplish the execution of pattern-based fault detection subroutines within the required time limits. The third problem is that the proposed system does not have a data interface that synchronizes real-time and historical data, which is necessary to generate patterns for classification at the beginning of a monitoring session (at least there is no mention that such a facility exists: only real-time data interface is mentioned).

In the recent years a number of commercial process monitoring systems, such as AIM-Supervisor® by SimSci®, IFIX® by Intellution®, and FactorySuite2000® by Wonderware®, have appeared in the market. Being similar to the monitoring system described above [Karsai et al., 1996], these systems have similar shortcomings: they do not have facilities for developing pattern-based monitoring applications and managing real-time execution of these applications.

Another recently developed commercial monitoring system, G2 Diagnostic Assistant® by Gensym®, does have facilities for developing and testing pattern-based monitoring applications and for managing execution of these monitoring applications in real time. In fact, this system can be considered pattern-based, multitasking, generic and capable of real-time operation at the same time. However, the choice of pattern recognition methods used by G2 Diagnostic Assistant® is limited to a set of a few standard neural networks. The system can import pattern-based monitoring applications developed in other environments, but cannot modify these monitoring applications to enable them to monitor different processes.

There are other computer systems for real-time monitoring of industrial processes [Prock, 1992; Rengaswamy and Venkatasubramanian, 1993; Linkens and Abbod, 1994]. However, these other systems are even farther away from the fully generic, pattern-based, and multitasking ideal. Therefore, there is a need for creating a generic multitasking system for pattern-based real-time

process monitoring. Such a system has been created and the design and operation of this system is described in the subsequent chapters of this work.

2.5 Chapter summary

This chapter explained what pattern-based methods for fault diagnosis are and why they should be used for real-time fault monitoring in the process industry. A brief overview of various pattern generation and pattern classification techniques was presented. It was explained why a good pattern-based monitoring system should be generic and multitasking. The chapter concluded by describing the monitoring systems whose design is closest to the generic pattern-based multitasking ideal and motivated the need for a new monitoring system that would be an implementation of this ideal.

CHAPTER 3

CONCEPTUAL DESIGN OF A GENERIC SYSTEM FOR PATTERN-BASED REAL-TIME PROCESS MONITORING

3.1 Introduction

The design of a successful computer system for fault monitoring should address a number of issues [Russell, 1994; Lee, 1995]. First, the system should be flexible and economic. This means that the system should be able to monitor a variety of industrial processes and the human effort necessary to adjust the system to monitor a new process should be minimal. Second, the monitoring system should integrate human operators into the process of automated fault supervision instead of completely removing humans from the industrial production cycle. Third, the system should provide early detection of process malfunctions to give plant personnel an opportunity to remedy the faults in the preventive mode. The basic concepts of the monitoring system design that implements the principles of flexibility, economy, human involvement and real-time operation are presented in this chapter.

Flexible computer software is usually developed in a modular manner. A computer program generated using this approach consists of several building blocks: a base program or framework and a number of different interchangeable modules that customize the program and provide the desired flexibility. The resulting monitoring tool is "generic" in a sense that the framework can be used to address any type and number of specific tasks.

Minimization of the development effort required to integrate new monitoring applications into a monitoring system can be achieved with the help of special user-friendly facilities. These facilities simplify generation of new modules and modification of existing application-specific modules. In addition to minimizing human effort, these facilities solve the problem of integrating human operators into the monitoring system and employing the operators' knowledge about the

monitored processes. System developers should take advantage of the best way to teach plant operators to understand and value the process monitoring system by letting the operators themselves participate in the creation and customization of the system.

Timely detection of malfunctions is possible only if the system runs with the latest process data. To have access to the latest readings of the monitored sensors, a monitoring system should acquire these readings using real-time data acquisition techniques. Correct scheduling of tasks in a complex generic system is possible only if some jobs are executed in parallel. This requires a programming technique called multithreading. The ability of the overall monitoring system to control execution of different modules is enhanced if the modules are implemented as dynamic-link libraries (DLLs). Use of real-time techniques is critical for an efficient real-time performance of the system.

This chapter proposes a methodology for design of a generic real-time multitasking system for pattern-based process monitoring. Section 3.2 discusses the basic design concepts. Section 3.3 is devoted to the programming techniques that were used to implement the proposed monitoring system. The proposed concepts and real-time programming techniques are presented for a generic pattern-based monitoring system that runs on a PC under Microsoft Windows®. However, the proposed methods are also generally applicable for other multitasking environments and other hardware platforms.

3.2 Basic concepts of generic pattern-based monitoring system design

The conceptual design of a generic system for pattern-based real-time process monitoring proposed in this work is presented in Figure 3.1. The monitoring system has a kernel that schedules, initiates, and controls (based on the situation and user actions) execution of different tasks within the monitoring system. The kernel is also responsible for formatting data for

“monitoring applications” that perform pattern classification. Finally, the kernel displays the state diagnosed by each individual monitoring application. The system has custom modules for acquisition of historical and real-time data needed to determine the monitored conditions, displaying the latest trends of the monitored variables, and an arbitrary number of custom monitoring programs called monitoring applications. This section discusses the most important concepts of the proposed monitoring system design.

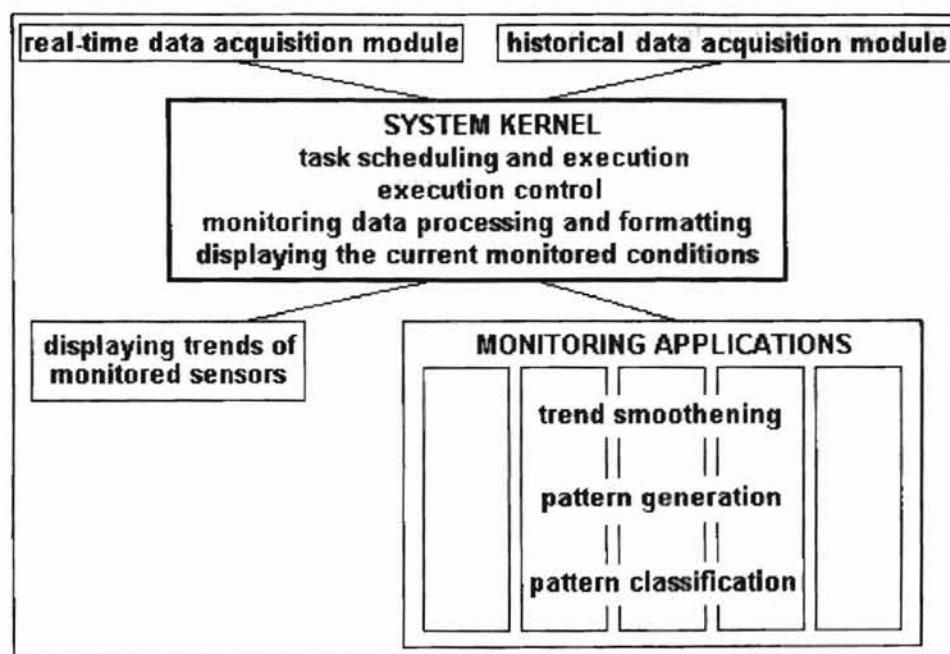


Figure 3.1. Design of a generic system for real-time pattern-based monitoring.

3.2.1 Monitoring different processes by different monitoring applications with a single control center

The proposed monitoring system is capable of using different methods for fault monitoring. A number of different pattern-based methods for fault diagnosis were discussed in Sections 2.2 and 2.3 with the conclusion that there is no universally best fault monitoring technique. Moreover, new methods for pattern classification are being developed, and these

methods may show a better performance in fault monitoring than the existing techniques. Consequentially, the ability to upgrade or replace a monitoring application with relative ease is essential. Therefore, the best monitoring performance can be achieved if a generic process monitoring system is able to use different process monitoring methods implemented as independent, easily replaceable modules.

The most reasonable approach to developing a pattern-based process monitoring system that allows the user to choose fault diagnosis methods is to implement the modules for pattern generation and pattern classification as separate programs that can be created by the user. Compared to the existing monitoring systems that generate fault recognition subroutines using a rigid framework defined by the system manufacturer [e.g. Karsai et al., 1996], this design gives the user more freedom in choosing and applying different pattern-based fault monitoring methods.

The generic monitoring system described in this section is primarily intended for pattern-based fault diagnosis. However, the proposed design allows the user to run non-pattern-based monitoring applications as well. The user can upgrade the monitoring system at any time by supplementing it with non-pattern-based monitoring applications that comply with the configuration and communication standards described in Subsection 3.2.3.

In the proposed monitoring system design, monitoring applications are special programs configured to run within the monitoring system framework. Each monitoring application is designed and customized by the user to monitor one user-defined process. Every monitoring application has its own configuration file that contains the information on how the monitoring system kernel should prepare data for the monitoring application and interpret the result generated by the monitoring application. The monitoring application should be executable code that generates patterns from the data provided by the monitoring system kernel and classifies the input patterns to determine the monitored condition. In general, any program for pattern generation and classification that can run in a batch mode (i.e., after being launched in such a mode the program

reads process data, generates a pattern for classification, classifies the pattern, and terminates) can be configured to run with the proposed monitoring system.

A monitoring application should not run as an independent program. As opposed to other concepts of generic monitoring system design [e.g. Karsai et al., 1996] where each monitoring application independently executes, collects real-time plant data, determines the monitored condition and displays the result along with the monitored plant signatures, in the monitoring system design described in this work, monitoring applications do not run independently, do not collect plant data and do not display anything. The tasks of plant data acquisition, displaying the diagnosed monitored conditions, and displaying the trends of the monitored variables are carried out by other parts of the monitoring system. Monitoring applications run in the background without opening any windows and simply create and classify input patterns. Each monitoring application is launched and supplied with current plant data prepared in an application-specific format from the monitoring system kernel.

There are two reasons for delegating the tasks of collecting plant data and displaying the monitored conditions and trends from monitoring applications to other parts of the proposed monitoring system. The first reason is efficiency. Running a single data collection utility and a single user interface for multiple monitoring applications that simultaneously monitor different processes takes much less processor time and memory than running individual data collection modules and user interfaces for each monitoring application. The second reason is simplicity. There is no need for the user to write custom user and data interface code for each monitoring application, and there is no need for a facility that would generate a user and data interface for each monitoring application.

The most important concept of the generic monitoring system design proposed in this work is the use of multiple monitoring applications controlled from a single center. Each process is monitored by one monitoring application developed by the user. After being launched by the system

kernel, the monitoring application reads the current plant data, calculates the current monitored condition, and terminates. Other parts of the monitoring system perform plant data acquisition and formatting for each monitoring application and display the monitored trends together with the operating states identified by the individual monitoring applications. This generic design concept makes the monitoring system highly flexible, computationally efficient, and capable of monitoring an almost unlimited number of processes on the same computer. The user of a monitoring system, that implements the outlined concept, can choose virtually any fault diagnosis method for any monitored process.

3.2.2 Simultaneous monitoring of several different processes

Due to the recent achievements in computer hardware and software manufacturing, current computers are powerful enough to perform pattern-based monitoring of several processes concurrently on the same hardware platform. A system monitoring several processes on the same computer is more convenient to plant operators and requires a much smaller investment in hardware than a system that can monitor only one process.

The proposed system monitors different processes with different priorities. High priorities are attached to more dynamic processes where the monitored conditions change frequently and a prompt response is required in the case of fault appearance. The monitored condition of a high priority process should be updated more frequently than the monitored state of a low priority process.

The proposed monitoring system determines the current condition (operation state) of each monitored process sequentially, by executing not more than one monitoring application in a batch mode at any time during a monitoring session. There are three reasons for executing monitoring applications sequentially rather than in parallel. First, the simultaneous execution of all monitoring applications may be impossible due to limited hardware resources, primarily the finite-size memory

where the monitoring applications load their code and data. Second, it is difficult for the system to enforce the monitoring priorities for each process if all the monitored conditions are calculated in parallel. The system does not know when each monitoring application will complete calculation of the monitored condition. Therefore, if monitoring applications are run concurrently, the system will update more often the monitored conditions whose calculation takes less processor time. On the other hand, the system where monitoring applications are executed sequentially will calculate and update more frequently the monitored conditions of the high priority processes. The third reason (simplification of application linking) is discussed in Subsection 3.3.2. In summary, a system that calculates the monitored conditions for each monitored process sequentially can supervise a virtually unlimited number of processes with a great efficiency.

Figure 3.2 illustrates the concept of sequential priority scheduling for a system that monitors three different processes simultaneously using three different monitoring applications, one for each process. Monitoring application 1 has a high priority, and is executed very frequently. Monitoring of processes 2 and 3 has a low priority, and, as a result, monitoring applications 2 and 3 are executed less frequently.

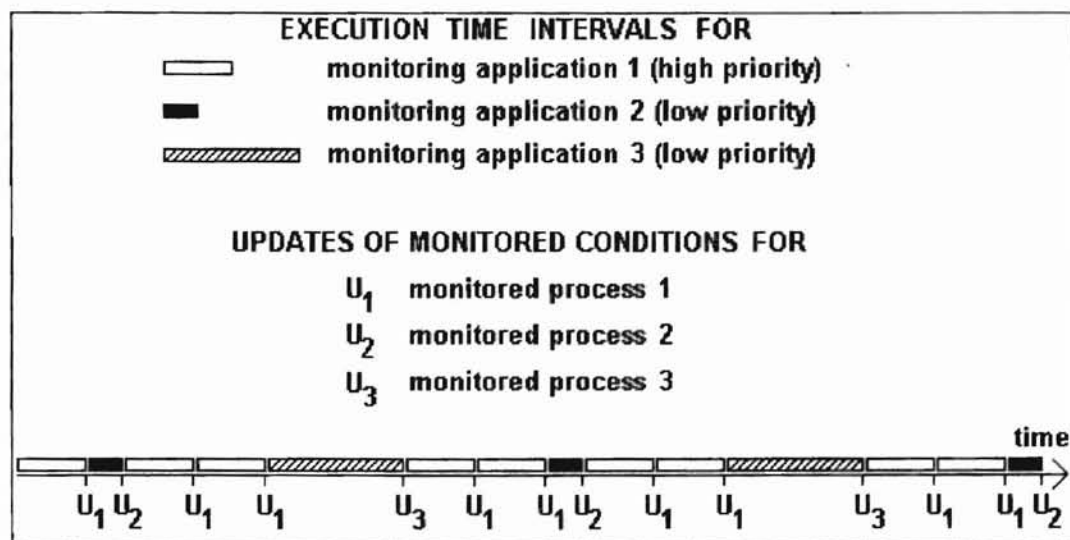


Figure 3.2. An example of scheduling calculation of monitored conditions for different processes.

3.2.3 Quick and easy generation of new monitoring applications

The success of any computer software, including a monitoring system, strongly depends on how easy it is to understand and use the software. A computer program is useless, no matter how capable it may be, until the user can operate and appreciate the program. The proposed system has been developed with an emphasis on ensuring that the design and operation of the system make sense to the operators of the plants where the system will be installed.

The part of a monitoring system, which is most difficult to implement in a fashion that the user easily understands, is the monitoring application builder. Implementation of the real-time modules of a monitoring system is quite straightforward: a typical design allows the user to start a monitoring session, check the current monitored conditions, and view the trends of the monitored variables by pressing some command buttons or selecting menu items. However, to create a custom monitoring application, the user has to specify many parameters, user actions have to be application-dependent, and sometimes the user has to have an advanced knowledge of the monitoring application or some specific skills, e.g., the ability to program in a computer language. In order to substantially facilitate the process of designing monitoring applications by monitoring system users, the user interface and the process of creating monitoring applications must be simplified and standardized. The proposed system satisfies this requirement.

The proposed generic pattern-based monitoring system makes it possible to create monitoring applications in a simple user-friendly fashion for both advanced and basic level users. The system has standard routines for communicating with the kernel and a detailed description of how to develop a new monitoring application from scratch using a set of these routines as a template and how to integrate these routines into already existing pattern classification programs. The standard routines accelerate the process of generating executable programs that employ new pattern generation and classification methods. A library of standard pattern-based methods for

fault diagnosis can be provided for basic level users. A simple-to-use editor for creation and modification of monitoring application configuration files is a part of the monitoring system.

The data communication and configuration protocols for all monitoring applications must also be standardized. The monitoring system kernel must pass plant data to a monitoring application and receive classification results from the application in a standard form. The application-specific format of this data, the monitored conditions and other application-specific details must be maintained in configuration files in a standard format that can be recognized by both the kernel and the monitoring applications. A standard configuration file generation utility and data transfer routines are provided with the proposed generic pattern-based monitoring system.

Figure 3.3 shows the modules provided to meet the previously discussed concept.

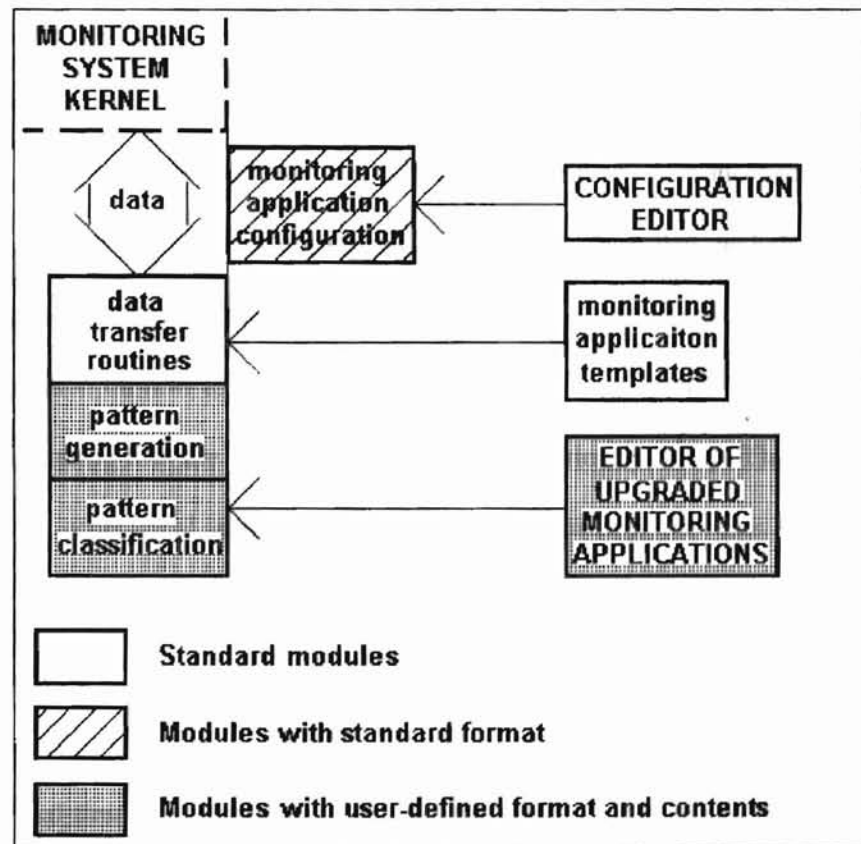


Figure 3.3. Building a monitoring application.

3.2.4 Section summary

Section 3.2 described the design concepts underlying the proposed generic pattern-based real-time process monitoring system.

1. The concept of monitoring different processes by different monitoring applications with a single control center. This design concept provides the capability to monitor different processes using custom methods. This concept also makes it possible to easily upgrade the monitoring system to use new methods for fault diagnosis in the future.
2. The concept of simultaneous monitoring of several processes by running the corresponding monitoring applications sequentially, one at a time, in a batch mode. This design concept provides the capability to efficiently monitor a virtually unlimited number of processes on one computer with limited hardware resources.
3. Each monitoring application communicates with the monitoring system kernel using standard system-defined protocols. The monitoring system has standard data transfer and processing routines that can be a part of new monitoring applications. New monitoring applications can be developed using special templates that include data transfer and processing routines. Each monitoring application has a standard configuration file with the information on how to run and communicate with the application. Monitoring application configuration files are created with a special user-friendly utility. This concept is aimed at making economical monitoring systems and involving the user into the process of monitoring system creation and customization.

3.3 Real-time programming techniques for generic pattern-based monitoring

Although not necessarily obvious, reliable operation of a real-time monitoring system requires the use of sophisticated real-time programming techniques. These techniques enable the

monitoring system to perform time-critical tasks as fast as possible and to accomplish these tasks within specified time intervals. The most time-critical tasks are acquisition of process data from external sources, data processing, displaying the monitored condition together with monitored variable trends, and running individual monitoring applications. Real-time data acquisition is best performed using Dynamic Data Exchange (DDE) and other related techniques that transfer data between programs without using permanent data storage devices (e.g. hard drives) as an intermediate buffer. Timely processing of monitoring data and calculation of monitored conditions can be achieved by using multithreading (makes it possible for several subroutines of the same program to run simultaneously). Fast and efficient execution of monitoring applications can be achieved with dynamic linking. We use these real-time techniques (DDE, multithreading, and dynamic linking) in Microsoft® Windows® operating environment. Details are presented in the remainder of this section.

3.3.1 Multithreading

The proposed monitoring system design was developed specifically for a multitasking operating system, i.e., an operating system capable of performing several tasks simultaneously. The system must be able to collect plant data, run monitoring applications, and interact with the user at the same time. It is impractical to execute all the required system tasks sequentially, one after another, in the same manner as the monitoring applications. For example, real-time data acquisition cannot wait while a user interacts with the monitoring system interface or until a monitoring application finishes running. Likewise, the user expects prompt response to keyboard or mouse entries and should not wait for the current monitoring application to finish running or for the current data sample collection to be completed.

Although Microsoft Windows® and other operating systems can execute several independent applications simultaneously, it is impractical to make a monitoring system

multitasking by implementing it as a set of standalone executable files. There are three reasons why each task should not be implemented as a separate executable file. The first reason is that task synchronization (initiating, suspending, and terminating execution of a monitoring system task at the request of other tasks) and exchange of data between the tasks requires shared memory (a Win32 facility that allows several processes to access the same virtual memory addresses [Microsoft Corporation, 1995c]). The second reason is that under Microsoft® Windows® separate processes (in computer science terminology process means an executing program that consists of a private virtual address space, code, data, and other operating system resources) are limited in the ability to control each other. One process cannot be suspended by another process (unless the first process is a debugger for the second process). In addition, termination of one process by another may affect the stability of the operating system. The third reason is that initializing a new process to perform a monitoring system task and terminating the process when the task is completed may take a significant share of central processing unit (CPU) time and make the monitoring system slow and inefficient. Therefore, a real-time monitoring system should run as a single process and use advanced multitasking techniques.

The multitasking technique best suited for a generic pattern-based real-time monitoring system is multithreading [Kleiman, 1996]. This technique is available under Microsoft Windows® and other operating systems. Multithreading is a method for simultaneous execution of several subroutines within the same process. The subroutines that are executed concurrently within the same process are called threads. Threads share the same memory space, therefore execution of different threads can be easily synchronized, and data can be easily exchanged between the threads using global variables, arrays, and structures. Under Microsoft Windows®, each thread can be allowed (depending on the mode of thread creation) to fully control execution of any other thread of the same process [Microsoft Corporation, 1995d]. A thread is assigned one of seven possible priorities [Microsoft Corporation, 1995d] that designate how much CPU time should be allocated

for the thread. The initialization and termination of a thread takes much less CPU time than the initialization and termination of an individual process. A multithreaded monitoring system will use much less memory than a monitoring system where each task is performed by a separate process. Our proposed monitoring system implements different tasks or modules as different threads of the same process.

3.3.2 Dynamic linking of monitoring applications at run time

To run as one process, all executable parts of the monitoring system must be linked together. However, monitoring applications are supposed to be compiled and linked with their run-time libraries independently of the compiling and linking of the monitoring system kernel. One possibility to solve this problem would be to include a tool for the conventional static linking of selected monitoring applications with the system kernel before each real-time monitoring session. Unfortunately, in such an implementation, the function called by the monitoring system kernel to execute a monitoring application (monitoring application entry function) must have a unique name known to the kernel for each selected monitoring application. Otherwise, the linker will produce an error caused by multiple declarations of the same entry function identifier in different monitoring applications. A more serious drawback of such a design is that the static linking usually requires the executable code of all the monitoring applications and the monitoring system kernel to be written in the same programming language. Finally, the resulting statically linked code may be too large and use too much memory.

To simplify the design of our generic real-time monitoring system, provide the ability to run monitoring applications written in different programming languages, and reduce the size of memory required by the monitoring system, our system kernel links standalone executable code of the monitoring applications dynamically at run time.

Dynamic linking is a way for a program to run a subroutine that is not a part of the executable code of the program. Under Microsoft® Windows®, instances of executable code that can be linked dynamically to a program are usually located in dynamic-link libraries [Microsoft Corporation, 1995b]. A process running under Windows can map the executable code of a dynamic-link library (DLL) into the address space of the process and execute functions (more generally, subroutines) whose code is contained in the DLL. Programs written in different programming languages can call the same DLL function as long as the programs follow the function's calling convention. After completing execution of a DLL function, a process can terminate the dynamic link by unloading the DLL from the process address space. The ability to unload DLLs provides for efficient use of hardware resources and allows a process to use the same identifier to call different functions contained in different DLLs. Calling different functions using the same identifier in the same process is not a conventional use of DLLs, but, as discussed in the next paragraph, this method is very useful for running monitoring applications.

The monitoring system kernel should be dynamically linked only to the currently running monitoring application. According to Subsection 3.2.2, monitoring applications should always be executed one at a time, therefore the monitoring system kernel can be designed to have not more than one monitoring application linked to the kernel at any time. In this case, the entry function (called by the monitoring system kernel to run the application) for each monitoring application can have a standard prototype (function prototype consists of function name, set of arguments and return value type). In our monitoring system running as a single process, this standard prototype will unambiguously refer to the entry function of the currently loaded monitoring application. To simplify the process of developing monitoring applications, the standard entry function prototype is included in monitoring application templates. Besides the simplification of monitoring application development, dynamic linking of only one monitoring application simplifies the development of the monitoring system kernel. It is easier to develop a generic real-time monitoring system that

executes all the monitoring applications by calling the same function than a system that has to keep track of all the monitoring application entry function prototypes.

Figure 3.4 illustrates the dynamic linking of monitoring applications described in this section. In this monitoring session the monitoring system kernel runs monitoring applications A, B, and C. Monitoring application B is currently loaded into the monitoring system memory space and is being executed by the kernel. To execute monitoring application B, the kernel called a standard entry function, and this call referred to the entry function of monitoring application B currently linked to the kernel. Once the execution of monitoring application B is completed, it will be unloaded, and the dynamic link with B will be automatically terminated. After that the kernel will choose the next monitoring application to execute, load the selected monitoring application, and call the standard entry function again. This time the standard entry function call will start the execution of the newly loaded monitoring application which is not necessarily application B.

3.3.3 Real-time data acquisition techniques

In real-time monitoring, it is essential to have the very latest readings of the monitored process variables. Therefore, the monitoring system must use a fast and efficient method for data acquisition.

Real-time data for pattern-based process monitoring should be acquired from the installed software and hardware systems that control the process and not directly from the sensors. Making use of already collected data is less expensive than establishing independent hardware links with the sensors measuring process variables. Process control systems typically log sensor readings in a continuous mode, however there is a significant delay between the collection of data from sensors and appending this data to the log file. Therefore, the real-time data acquisition by a pattern-based monitoring system should be implemented as direct requests of the current sensor readings from the process controlling and process data logging software.

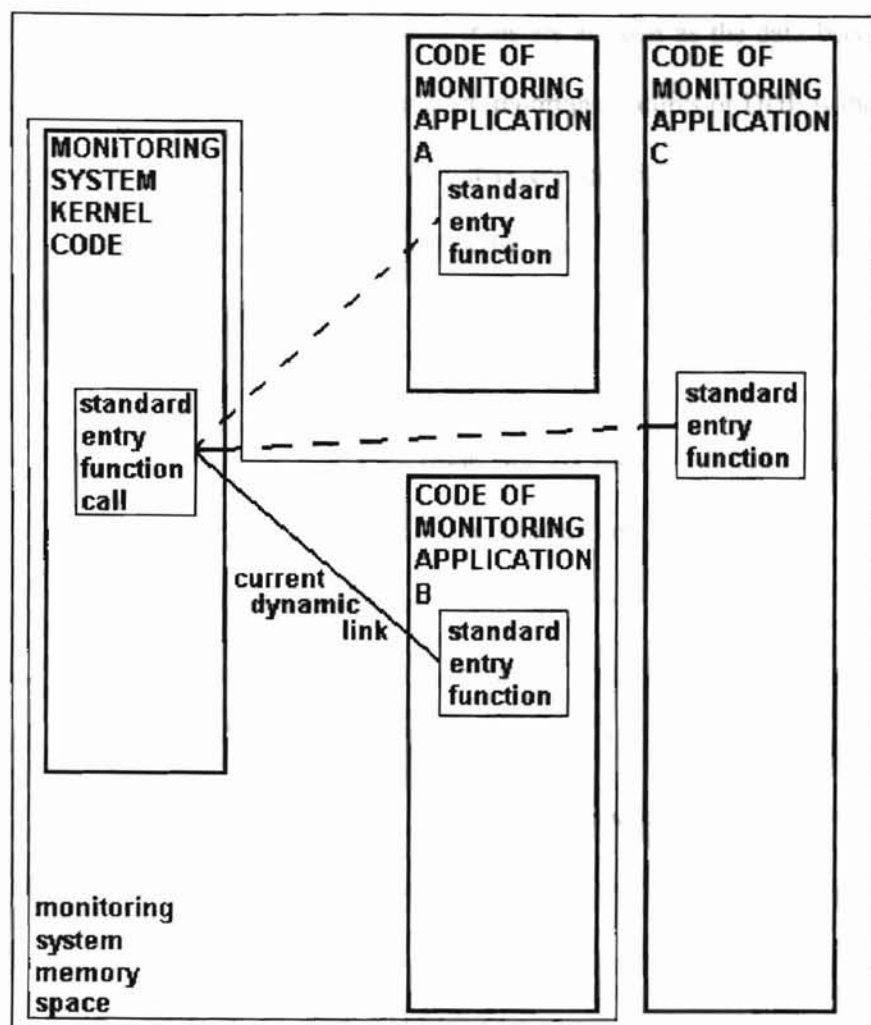


Figure 3.4. Run-time dynamic linking of monitoring applications.

One technique for fast data transfer developed by Microsoft Corporation® and supported by many industrial data processing systems is Dynamic Data Exchange [see Microsoft Corporation, 1995a]. Dynamic Data Exchange (DDE) is a protocol for data transfer between two independent programs. One of these two programs runs as a DDE server and the other one runs as a DDE client. To perform data transfer, the DDE client establishes a connection with the DDE server and sends a description of the requested data. Upon receiving the request, the server prepares the requested data and sends it back to the client. DDE does not require an ongoing user interaction to transfer data. DDE has a hot-link option that allows a DDE server to supply the

client with data continuously by performing data transfers as soon as the data becomes available without waiting for client requests. One of the most important features of DDE is the capability of linking two programs running on different computers connected by a local network (NetDDE). Therefore, one good technique for real-time data collection by a generic pattern-based monitoring system is acquisition of the current sensor readings from industrial data processing software via a DDE link.

DDE is not the only technique appropriate for acquisition of real-time data in pattern-based process monitoring. One extension of DDE is Object Linking and Embedding (OLE). OLE lets Microsoft Windows® applications achieve a very high degree of integration. OLE provides a set of standard interfaces so that any OLE program can interact fully with any other OLE program without any built-in knowledge of its possible partners. A modification of OLE called OPC (OLE for Process Control) is supported by a majority of commercial automated control systems developed in the recent years.

Making use of Structured Query Language (SQL) servers is a third possible method for real-time data acquisition by a pattern-based monitoring system. An SQL server is a relational database written in SQL. Windows applications can transfer data to and from SQL servers in real time using a special protocol similar to DDE. If installed plant data processing software sends the current sensor readings to an SQL server, it may be very convenient for a monitoring system to use this SQL server for real-time data acquisition.

Real-time data acquisition techniques applicable for pattern-based monitoring are not limited to DDE, OLE, and communication with SQL servers. Any method that allows a quick transfer of data without writing it on hard drives or other permanent data storage devices and is supported by the installed software and hardware is appropriate. As described in Chapter 4, our monitoring system uses DDE and NetDDE for real-time data acquisition.

3.3.4 Section summary

Section 3.3 described the programming techniques (multithreading, run-time dynamic linking, and real-time data transfer) that should be used to develop an efficient generic real-time monitoring system operating under Microsoft Windows®. Application of these programming techniques affects the monitoring system implementation as follows.

1. The monitoring system should be able to run certain tasks simultaneously. The system should run as a single process with simultaneously running tasks implemented as different threads. More urgent tasks should be assigned higher priorities, and less urgent tasks should have lower priorities.
2. Monitoring applications should be implemented as dynamic-link libraries. The monitoring system kernel should execute each monitoring application by loading an application in the monitoring system memory space and calling a function with a standard prototype. This function has the same prototype and is defined as an entry function in all monitoring applications.
3. The monitoring system should collect the current values of the monitored variables from the existing plant control system. The process of data collection should be performed using a real-time technique such as DDE, OLE, SQL data interface or any other method that allows a fast transfer of data without using hard drives and other permanent storage devices as an intermediate buffer.

3.4 Chapter summary

This chapter discussed general requirements for a generic pattern-based real-time monitoring system. Basic concepts of monitoring system design were proposed. According to these concepts, the system should be able to simultaneously monitor several different processes, each

process should be monitored with an independently developed custom monitoring application, and the development and design of the monitoring applications should be within a framework of certain standards and facilities. This chapter also described the most important programming techniques required for the monitoring system to operate in real time under Microsoft® Windows®. These techniques are multithreading, run-time dynamic linking, and real-time data acquisition.

CHAPTER 4

IMPLEMENTATION OF A GENERIC PATTERN-BASED REAL-TIME MONITORING SYSTEM

4.1 System design

This chapter describes our proposed generic pattern-based real-time monitoring system. We refer to the composite system as GRTMS (Generic Real-Time Monitoring System). The system described in this chapter provides the ability to perform simultaneous monitoring of multiple industrial processes. The GRTMS software also provides the tools necessary to develop and test custom monitoring applications. The GRTMS was written in the ANSI C programming language and compiled and linked using Borland C++ into a 32-bit code that runs under Microsoft® Windows®. The user interface for the system was developed with CVI LabWindows®. The GRTMS was developed specifically for the industrial processes that use the Intouch® plant historian software by Wonderware Corporation®. The GRTMS is the first generic real-time monitoring system specifically intended for pattern-based process monitoring.

GRTMS was developed according to the principles proposed in Chapter 3. An overview of the different components of the GRTMS is presented in Figure 4.1. The system consists of a kernel that can simultaneously monitor multiple processes using separate monitoring applications. GRTMS also includes the tools and standards for developing monitoring applications. The GRTMS runs as a single multithreaded process with monitoring applications implemented as dynamic-link libraries (DLLs) with standard entry functions. The system performs acquisition of real-time data using DDE or NetDDE (networked version of the DDE protocol). The details of how the methods presented in Chapter 3 are implemented in the GRTMS are given further in this chapter.

The GRTMS performs real-time monitoring using a Generic Real-time Monitoring Module (GRTMM). The GRTMM collects historical and real-time data, formats this data for monitoring applications, schedules and runs monitoring applications, and displays data trends and the diagnosed conditions. The GRTMM can be configured and controlled from a user-friendly graphical interface. The most important component of the GRTMM is Application Data Queue (ADQ) that processes monitoring data in real time. The GRTMM serves as the control center (kernel) for real-time monitoring.

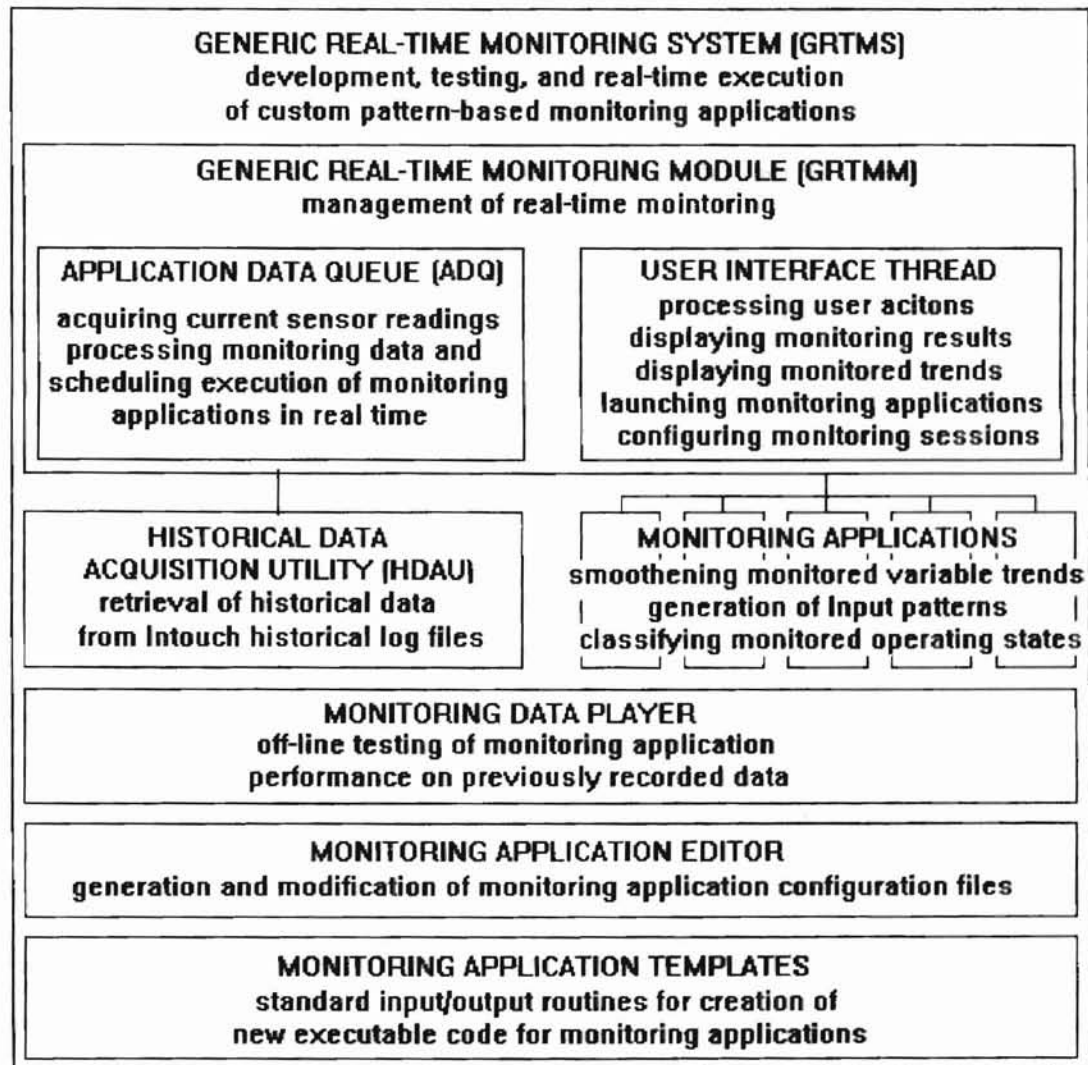


Figure 4.1. Hierarchical structure of the Generic Real-Time Monitoring System (GRTMS).

The composite GRTMS also has facilities for development of monitoring applications. Monitoring application code can be created using special templates included in the GRTMS. To enable the GRTMS to monitor new processes, the existing monitoring applications can be configured with a Monitoring Application Editor. Performance of monitoring applications can be tested off-line on recorded historical plant data with a Monitoring Data Player that emulates the GRTMM. The application templates, Monitoring Application Editor, and Monitoring Data Player simplify the job of creating new monitoring applications.

In addition to the principles discussed in Chapter 3, the GRTMS was built to satisfy two more requirements. The first requirement is ease of use. Although the GRTMS is a highly sophisticated piece of computer code, it was developed to be user-friendly without any special user training requirements. The second requirement is reliability. The GRTMS provides stable real-time operation even when unusual situations are encountered. The GRTMS code incorporates a significant number of sophisticated error traps. In the case when a problem is detected, the system provides a dialog box that describes the problem with an easily understandable explanation of what is wrong.

4.2 Implementation of real-time monitoring

Generic Real-Time Monitoring Module (GRTMM) is a part of the GRTMS that performs real-time monitoring. The GRTMM can simultaneously monitor several industrial processes in real time using one custom monitoring application for each supervised process. The GRTMM performs several principal tasks simultaneously: collecting monitoring data, processing monitoring data for each monitoring application, running monitoring applications on a priority-based, user-defined schedule, and interacting with the user.

A data flowchart for the GRTMM is shown in Figure 4.2.

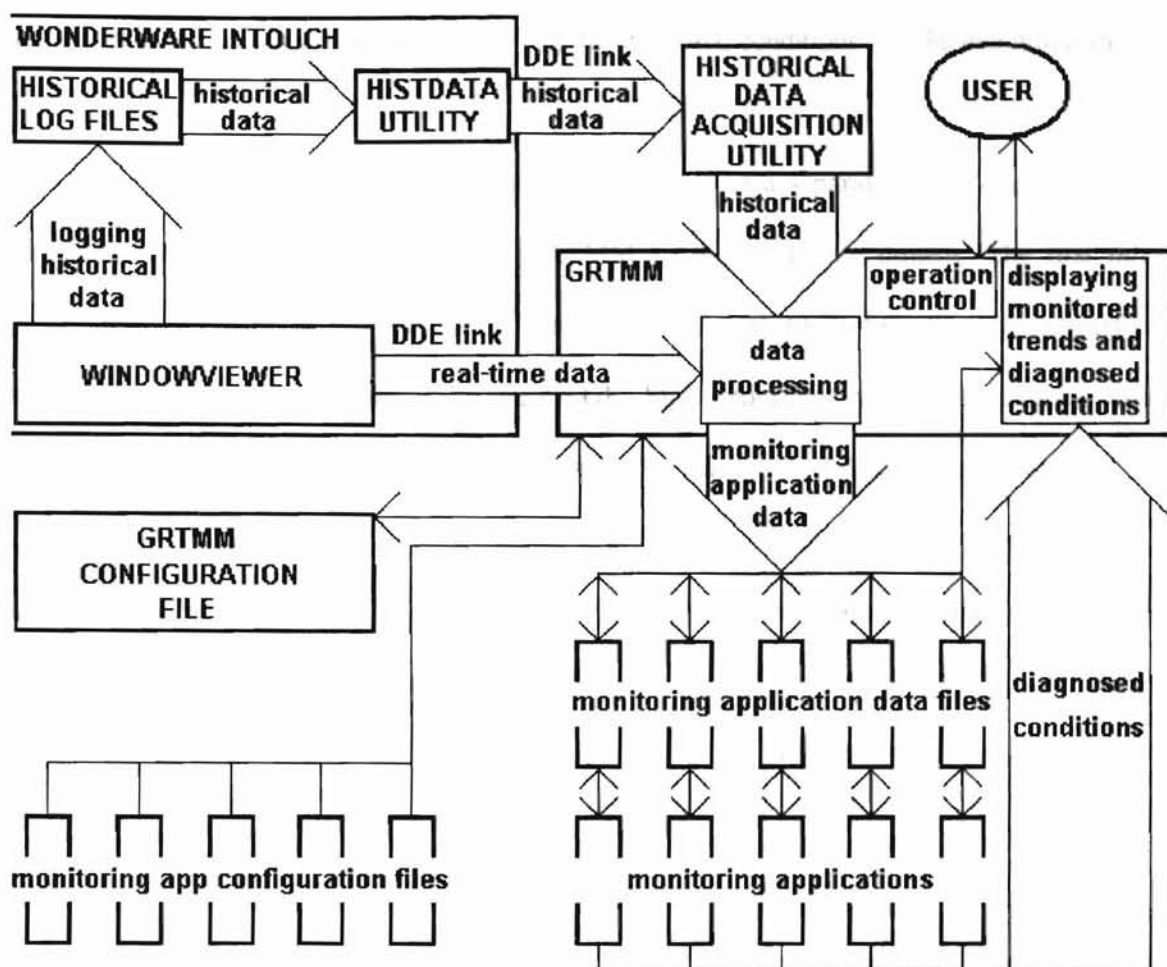


Figure 4.2. Data flowchart for real-time monitoring with the GRTMM.

Real-time monitoring involves performing a combination of seven tasks enumerated below:

- 1) retrieval of historical monitoring data from Wonderware® Intouch® log files using the Intouch® Histdata® utility;
- 2) acquisition of real-time monitoring data from Intouch® Windowviewer®;
- 3) synchronization of historical and real-time data;
- 4) scheduling execution of the different monitoring applications that the user desires to run;
- 5) preparing "fresh" monitoring data for monitoring applications and for display;
- 6) running monitoring applications;

- 7) processing user actions and displaying the diagnosed conditions and the monitored variable trends.

Task 1) is performed by a standalone utility implemented as a separate executable file. The six remaining tasks are implemented in the GRTMM as three separate threads. The first thread performs tasks 2) through 5). The second thread performs task 6), and the third thread performs task 7). The third thread can also configure the GRTMM before each real-time monitoring session at user request. Details associated with each task are provided in the following subsections.

4.2.1 Retrieval of historical data by Historical Data Acquisition Utility

The GRTMM uses both real-time and historical (i.e., previously logged) data for pattern-based real-time fault diagnosis. At the beginning of each real-time monitoring session, the GRTMM must be initialized not only with the current value of each monitoring variable, but also with a finite number of prior measurements. This subsection describes the process of acquisition of historical data by the GRTMM. Subsection 4.2.2 includes a description of the process of collecting real-time data and synchronizing the real-time data with the historical data.

To acquire historical data at the beginning of each monitoring session, GRTMM runs a standalone Historical Data Acquisition Utility (HDAU). The process of retrieving historical data is schematically shown at the top of Figure 4.2. The HDAU works with Wonderware® Intouch®, a commonly used data logging PC application for industrial processes. After being launched by the GRTMM at the beginning of a real-time monitoring session, the HDAU establishes a DDE or NetDDE link with Intouch® Histdata®. Histdata® is a utility that accesses historical trends logged by Intouch®. The HDAU uses the DDE or NetDDE link to request and receive the historical data required by the monitoring applications. Upon receiving the requested data, the HDAU logs the historical data in the format that the GRTMM can read, notifies the GRTMM of the data availability, and terminates. The HDAU is implemented as a separate process (and not as a thread

within the GRTMM) because of conflicts associated with real-time data acquisition. The Intouch® software is incapable of establishing safe parallel DDE or NetDDE links. However, implementing the HDAU as a separate batch process is not a problem because of the limited frequency of use of this facility.

4.2.2 Real-time handling of monitoring data and scheduling execution of monitoring applications

The tasks of acquiring real-time process data from Wonderware® Intouch®, synchronization of real-time and historical data, scheduling monitoring applications, and preparing “fresh” data for the currently active monitoring applications are performed by a single thread created at the beginning of each monitoring session and terminated at the session end. During a monitoring session, this thread reads the current values of all the monitored variables from Intouch® WindowViewer® (a utility that collects, displays, and logs the current trends of the monitored variables) via a DDE or NetDDE link after equal time intervals. The monitored variable values acquired in real time from WindowViewer® are supplemented, as required, with the historical data samples retrieved by the HDAU. This operation requires synchronization of the real-time and historical data. Once the GRTMM collects enough data to run monitoring applications, a decision must be made as to which application to execute next. This task is called “scheduling monitoring applications.” Before setting the flag that invokes execution of the scheduled monitoring application, the thread logs the latest available monitored variable value samples into the corresponding application input data file.

Each of the tasks listed above involve access and modification of collected monitoring data and require little central processing unit (CPU) time. To avoid dealing with access conflicts and to maximize the execution speed, these tasks are implemented as a single thread.

Monitoring applications are scheduled according to user-specified priorities and the consideration of how much time has elapsed since each monitoring application was executed last time. Monitoring application priorities are specified in the monitoring application configuration files generated by the user with the help of the Monitoring Application Editor. To set the application priorities, the user specifies the desired time interval between consecutive executions of the monitoring application. When determining which monitoring application to run next, the user-specified time interval is divided by the time currently elapsed since the last execution for each monitoring application. The resulting ratios are compared and the monitoring application with the smallest ratio is selected. No monitoring application is scheduled when all the calculated ratios are greater than one. As soon as a monitoring application finishes running, the procedure of scheduling is repeated. This method of scheduling allows the system to match the monitoring frequency of any monitoring application with the dynamics of the process being monitored.

The GRTMM handles monitoring data with the help of a custom data structure developed specifically for GRTMM and called Application Data Queue (ADQ). The ADQ is an extension of a common computer science data structure called a “queue” [Weiss, 1997]. The classic queue is a list, i.e., a series of structures of a certain type. There are two standard public (available to the code not belonging to the structure) operations on a queue: ENQUEUE that consists in inserting one element at the end of the list and DEQUEUE that consists in deleting from the list and returning the element located at the start of the list. It is convenient to keep monitoring data in a structure similar to a queue whose elements are vectors containing the values of all the monitored variables observed at a certain time. In this structure, ENQUEUE will add the samples acquired via a DDE or NetDDE link from WindowViewer®. DEQUEUE will transfer the monitored variable value samples into the monitoring application data input files and delete them from the structure.

Unfortunately, a simple queue is not suitable as a structure for the monitoring data in the GRTMM. Data samples must be added not only at the end, but also at the start of the queue during data synchronization. Second, monitoring data is transferred to each monitoring application input data file separately, therefore deletion of any data should not be performed until the data have been transferred to all the monitoring application input files that require the data. An extension of the conventional queue, ADQ, was developed to handle addition of elements at the queue start and to allow multiple DEQUEUE operations on the same queue element.

Design of the ADQ is shown in Figure 4.3. The ADQ is organized as two dynamically allocated arrays of equal length. One array stores monitoring data samples, and the other array stores vectors of monitoring application flags. The ADQ has two pointers: the first pointer shows which array element corresponds to the beginning of the queue and the second pointer shows which array element corresponds to the queue end. Samples of monitored variable values are stored in the vectors located between the beginning and the end of the queue. Each monitoring data sample has a corresponding vector of monitoring application flags in the array of application flag vectors at the same position as the position of the data sample in the array of monitoring data. The length of each vector of application flags is equal to the number of monitoring applications being run by the GRTMM. Each element of the application flag vector corresponds to one monitoring application. The vector element is equal to zero if the data from the monitored variable value sample corresponding to the application flag vector has been transferred to the monitoring application input data file that corresponds to the vector element. Otherwise, the element is equal to one. This design provides the means for the ADQ to keep track of DEQUEUE operations performed for each monitoring application.

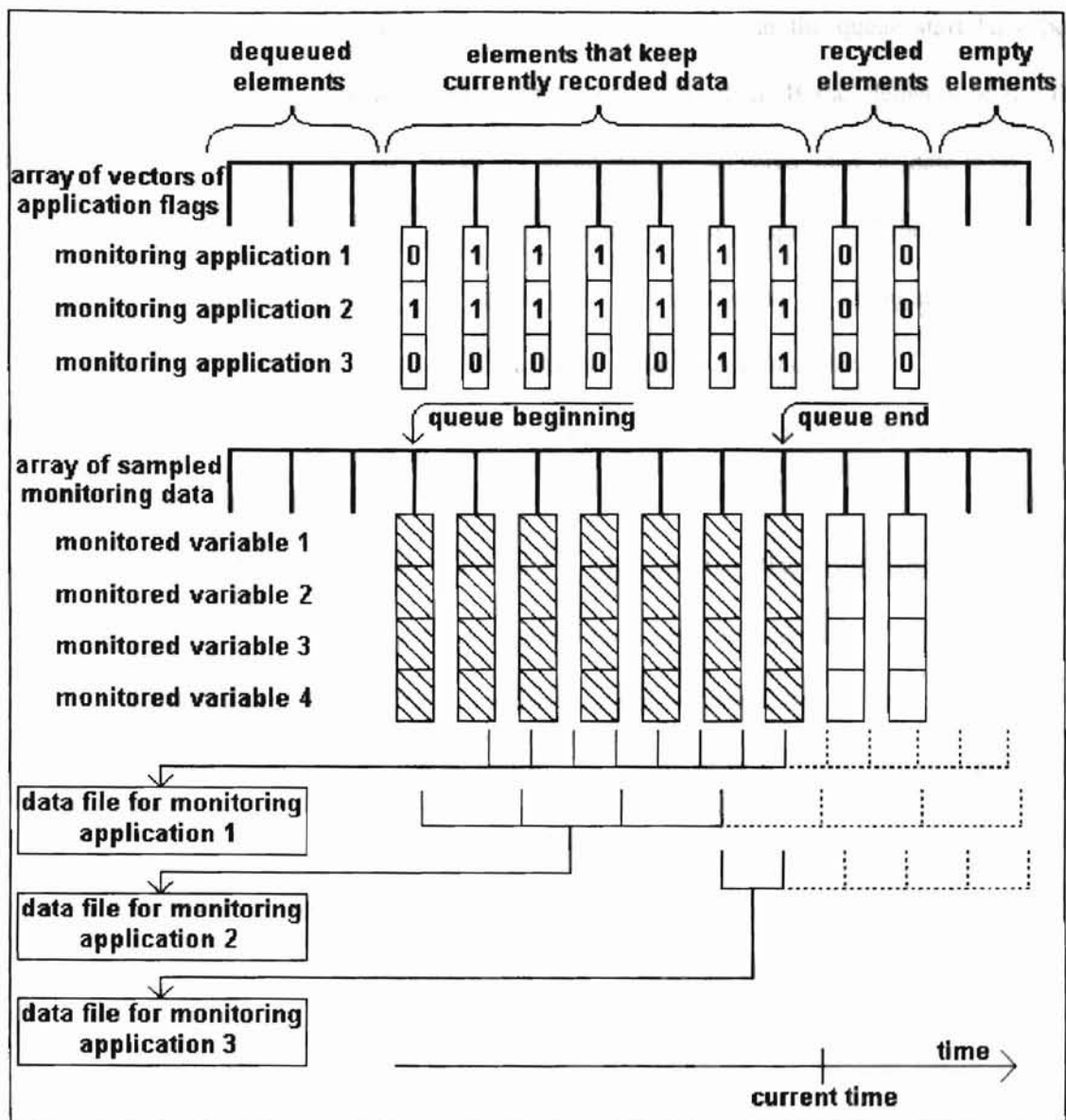


Figure 4.3. Application Data Queue (ADQ).

The first public operation performed by the ADQ is ENQUEUE, which adds a real-time data sample at the end of the queue. The ADQ either allocates memory for two new vectors attaching them immediately after the queue end in the arrays of sampled monitoring data and application flags or uses the previously allocated memory recycled to that location. After that the newly acquired values are copied to that memory location, all the elements of the corresponding application flag vector are set to one, and the pointer that indicates the queue end is set to the new

sample. At the same time, the ADQ checks if the data samples at the queue start have been transferred to all the monitoring applications that require this data. If the elements of the flag vectors corresponding to the data samples at the queue start are all zeros, then the data is no longer needed. In this case, the ADQ copies (recycles) to the locations after queue end the pointers to the memory taken by the samples that are no longer needed and by the corresponding application flag vectors. If all the positions after queue end are already occupied with recycled vectors, the memory taken by the used samples and the corresponding flag vectors is de-allocated. The pointer showing the queue beginning is updated accordingly.

If the physical end of the monitoring data and application flag vector arrays is reached, then new arrays for monitoring data and application flag vectors are allocated with an extra space for the pointers to the new samples and corresponding flag vectors to be added in the future. The pointers to the existing flag vectors and monitoring data samples are copied to the new flag vector and monitoring data arrays, and the old monitoring data and application flag vector arrays are de-allocated. This implementation of ENQUEUE provides efficient utilization of computer memory and minimizes the number of memory allocations and de-allocations required for real-time monitoring.

The second public operation of ADQ, INSERT, places a block of historical data samples at the start of the queue. As soon as the GRTMM receives the historical data samples retrieved by the HDAU, the ADQ attaches these samples immediately before the queue beginning, allocates memory for the corresponding vectors in the application flag array, sets all the elements of those vectors equal to one, and updates the pointer showing the start of the queue. The GRTMM requires the ADQ to perform the INSERT operation only when historical data is required to run the selected monitoring applications.

The third public operation returns the sample of monitored variable values from the specified location in the queue once for each monitoring application. This operation is called

DEQUEUE. When a monitoring application is scheduled to run, the GRTMM updates the input data file for the scheduled monitoring application with the data that have become available since the application was last run. Each monitoring application utilizes its own user-specified data-sampling interval. If the monitored variable values were not sampled at the specified time, the ADQ will return a linear interpolation between the two nearest available samples. After returning the requested variable value sample, the ADQ updates the application flags. All the flags of the scheduled monitoring application contained in the vectors corresponding to the samples located on the queue before the requested sample are set to zero. This setting indicates that the data preceding the requested sample are no longer needed for the monitoring application. When performing the ENQUEUE operation, the ADQ will check if the samples at the beginning of the queue have already been transferred to all the selected monitoring applications and remove the samples if the samples are no longer needed. In this implementation, DEQUEUE does not remove queue elements (as would occur in a classical queue) because this is the responsibility only of the ENQUEUE operation.

Monitoring data from the ADQ is transferred to monitoring application input data files organized as conventional queues of fixed length. Each application input data file has a fixed number of samples formatted as lines with a fixed number of symbols. These samples are queue elements. The queue starts with the earliest collected sample and ends with the latest one. The start of the queue immediately follows the queue end. Before running a monitoring application, the GRTMM performs ENQUEUE and DEQUEUE operations on the application's input data file by replacing the earliest collected data samples at the queue start with the samples of data that have become available since the last execution of the application. The process of updating monitoring application input data files is shown in Figure 4.4. More details about the format of application data files are given in Subsection 2.2.6 of the GRTMS Operating Manual [Shapovalov and Whiteley, 1999]. The design of the monitoring application input data files enables updates by

writing only the samples that have become available since the last file update instead of creating a whole new data file.

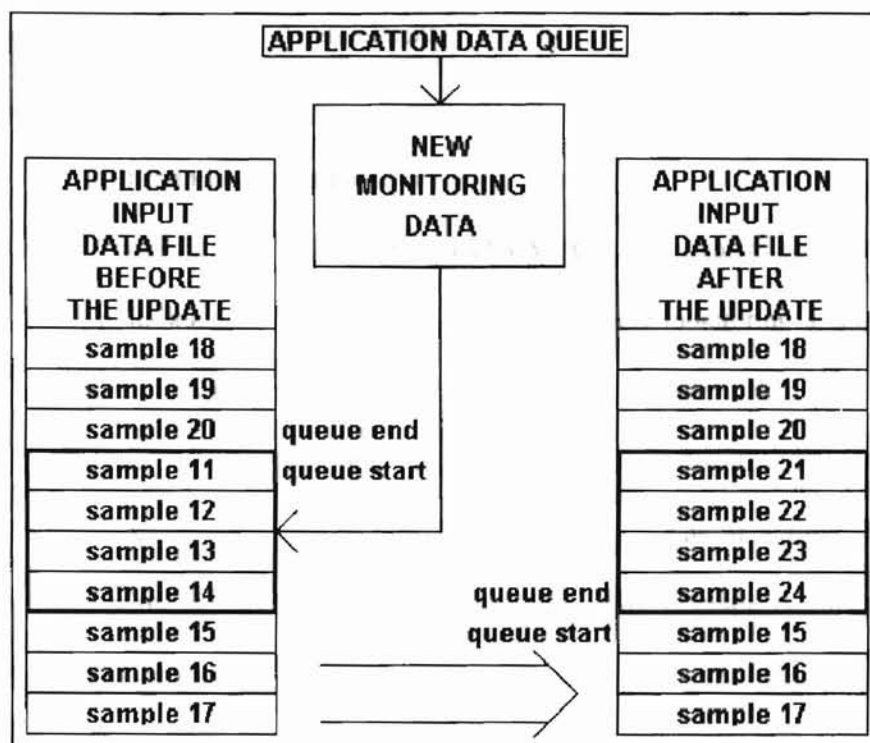


Figure 4.4. Illustration of updating a monitoring application input data file.

The scheduling and data handling approaches described previously are key design elements that provide the requisite flexibility when the choice of the monitoring application to be executed next and the amount of monitoring data to be stored in the memory at any given moment of time cannot be planned beforehand. The priority-based scheduling ensures that CPU (central processing unit) time is used efficiently with the most critical needs addressed first. Updating application data files only immediately before they are used by monitoring applications (and not continuously as new monitoring data becomes available) and writing only new data samples minimizes the use of the hard drive. This is important for a real-time application since the hard drive handles data by an order of magnitude slower than the random access memory. The flexible and relatively simple

design of the ADQ minimizes the processing time, requires simplified computer code, and optimizes the use of computer memory. The procedure of recycling dynamic memory, from the start of the ADQ to the queue end, also saves CPU time and decreases memory fragmentation caused by memory allocations and de-allocations.

4.2.3 Running monitoring applications

Monitoring applications run by the GRTMM can be created as either standalone executables or dynamic-link libraries (DLLs). In either case, the monitoring applications run as separate threads or as separate processes and do not delay real-time data processing by the GRTMM or operations with the user interface. As described in Subsection 3.2.2, the GRTMM executes monitoring applications sequentially, in a batch mode, one at a time. The GRTMM schedules execution of the monitoring applications according to the principle outlined in Subsection 4.2.2. As soon as a monitoring application is scheduled to run, the GRTMM updates the corresponding application input data file and checks the application type. If the scheduled monitoring application is a DLL, the GRTMM creates a new application thread that loads the DLL and calls the standard entry function that starts DLL execution. When a DLL-based monitoring application finishes running, it assigns the calculated result to a global variable. After that, the application thread unloads the DLL and terminates. If the scheduled monitoring application is a standalone executable file, the GRTMM launches this file and waits until it finishes executing. As soon as the application completes the monitoring analysis and terminates, the GRTMM reads the calculated condition and stores it in the same global variable as the one where it would store the condition calculated by a DLL-based monitoring application. The ability of GRTMM to run monitoring applications as both DLLs and standalone executables makes the GRTMS more convenient and friendly to monitoring application developers as well as the end users.

4.2.4 Interaction with the user during real-time monitoring

The user interface for the GRTMM is controlled by the primary process thread created at program launch. This thread is responsible for creation of all the other threads of the process. To accelerate fulfilling user requests sent to the GRTMM user interface, new threads are not created when user actions are being processed.

The GRTMM user interface allows the user to configure, start, and terminate real-time monitoring sessions, check the monitored conditions, view the trends of the monitored variables for each monitoring application, and receive error notifications. The main window of GRTMM user interface is shown in Figure 4.5. This window is used for starting and terminating monitoring sessions. The main window also displays information about the currently selected monitoring applications, the current monitoring status, and error messages. From the main window, the user can open the GRTMM configuration screen. More information on operating the main window and starting monitoring sessions can be found in Sections 3.3 and 3.4 of the GRTMS Operating Manual [Shapovalov and Whiteley, 1999].

Prior to initializing a monitoring session, the GRTMM must be configured using the configuration window shown in Figure 4.6. The user chooses which processes he or she wants to monitor by selecting the corresponding monitoring applications in the configuration selection window. The user also specifies the paths required to access the Intouch® historical data log files as well as several control parameters for the real-time monitoring session. Configuration information can be saved in the GRTMM configuration file, so that the operators can easily launch their own custom monitoring sessions. Full description of how to configure the GRTMM can be found in the GRTMS Operating Manual [Shapovalov and Whiteley, 1999], Section 3.2.

GENERIC REAL-TIME MONITORING MODULE

Help

CONFIGURATION **RUN MONITORING** **STOP MONITORING** **QUIT**

MONITORING RUNS: ☐ **MONITORING STARTED AT:** 07/26/99 09:55:06 **LAST UPDATED FROM WW WINVIEWER AT:** 09:56:21

STATUS BOX: monitoring runs **MESSAGE BOX:** Run Histdata at monitoring start ☐ yes ☒ no

WONDERWARE APPLICATION DIRECTORY:
c:\intouch.32\app2

ACTIVE MONITORING APPLICATIONS:	CURRENT READINGS
monitoring distillation column 05 (c:\cvi40\app1\task1.app)	07/26/99 09:56:21
monitoring distillation column 04 (c:\cvi40\borger1\rtborger1.app)	VAR3 72.21
	VAR4 72.455
	VAR5 76.413
	VAR6 87.492
	VAR1 49.448
	VAR2 64.982
	VAR7 44.463
	VAR8 72.555
	VAR9 44.799

Figure 4.5. The main window of the GRTMM user interface.

☒ **GENERIC R-T MONITORING MODULE - ADD/REMOVE MONITORING APPLICATIONS TO BE RUN** [] [] [X]

ADD APPLICATION TO THE LIST **REMOVE SELECTED APPS FROM THE LIST** **DONE** **QUIT PROGRAM**

MONITORING APPLICATIONS

monitoring distillation column 05
monitoring distillation column 04

PATH WITH INTOUCH TAGNAME DATABASE (INTOUCH APPLICATION DIRECTORY):

c:\intouch.32\app2

PATH WITH HISTDATA UTILITY

c:\intouch.32

PATH WITH INTOUCH HISTORICAL DATA FILES:

c:\intouch.32\app2

Max. delay allowed in real-time data sampling, s: 10 Historical database reading timeout, s: 1000 Application execution timeout, s: 30

Figure 4.6. GRTMM configuration screen.

As soon as a real-time monitoring session is started, the GRTMM Taskbar window (see Figure 4.7) opens. The Taskbar window displays the current state for all monitored processes. The Taskbar window has scroll buttons that can be used to display monitored conditions for an arbitrary number of monitoring applications. The GRTMM Taskbar window can also be used to open the Trendpad window that displays the current trends of the monitored variables for any monitoring application. The trends for each monitoring application are displayed in separate windows. Operation of GRTMM Taskbar window is described in detail in Section 3.4 of the GRTMS Operating Manual [Shapovalov and Whiteley, 1999].

The Trendpad window (see Figure 4.8) shows the monitored trends for a selected monitoring application. The Trendpad has two separate charts for the trends of the application variables used in generation of patterns for fault diagnosis (monitoring variables) and for the trends of the variables needed for the user to validate the diagnosed condition visually (validation variables). Inputs for the trends displayed on the Trendpad are retrieved from the current monitoring application input data file. Monitoring applications include an option of smoothening the monitored variable trends and supplying these smoothened trends along with the raw data for display on the Trendpad. More details on operating the Trendpad are given in the GRTMS Operating Manual [Shapovalov and Whiteley, 1999], Section 3.5.

4.2.5 Section summary

Section 4.2 outlined the most important details for real-time monitoring with the Generic Real-Time Monitoring System. The basic concepts of design and operation of the Generic Real-time Monitoring Module (GRTMM), such as the multithreaded design, the implementation of historical data collection, real-time data processing, scheduling and executing monitoring applications, and the design of GRTMM user interface, were described. A considerable amount of effort was devoted to addressing the issue of real time monitoring efficiency in the GRTMM.

TASKBAR

TOTAL NUMBER OF APPLICATIONS: 2

STOP MONITORING

APP #

APPLICATION TITLE

FIRST RUN LAST CHANGE

LAST RUN

MONITORING RESULT

scroll up

1	monitoring distillation column 05	07/26/99 10:09:16	10:09:16	NORMAL	TRENDS
2	monitoring distillation column 04	07/26/99 10:13:53	08:12:17	N/A	TRENDS
					TRENDS
					TRENDS
					TRENDS
					TRENDS
					TRENDS
					TRENDS

scroll down

Currently running

Figure 4.7. GRTMM Taskbar window

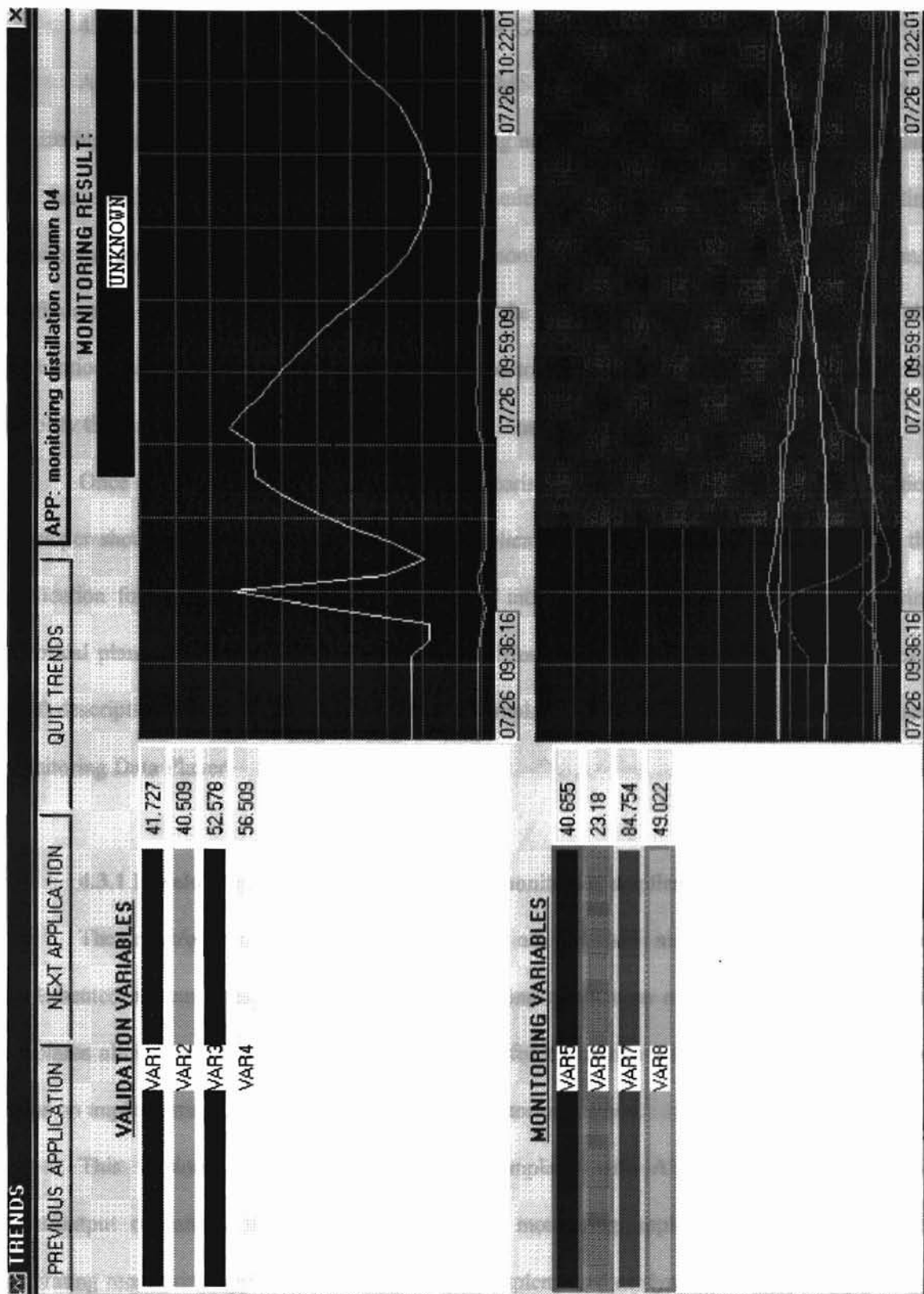


Figure 4.8. GRTMM Trendpad window.

4.3 Developing monitoring applications for Generic Real Time Monitoring System

As discussed in Subsection 3.2.3, the GRTMS has standard facilities for development of monitoring applications. Designing a new monitoring application normally starts from a template that has standard input/output routines for communication with the GRTMM and for reading monitoring application configuration files. Each monitoring application has a standard-format configuration file containing information on how the GRTMM should communicate with the application and interpret application output. A Monitoring Application Editor was developed to simplify the task of creating new monitoring applications.

Once development of a pattern-based monitoring application is complete, the application developer should make sure that the monitoring application is operating properly before using the application for real-time monitoring. Operation of monitoring applications can be tested using historical plant data with a GRTMS utility called Monitoring Data Player. This section gives a brief description of the monitoring application templates, Monitoring Application Editor, and Monitoring Data Player.

4.3.1 Developing new executable code for monitoring applications from templates

The input/output operations, standard for all GRTMS monitoring applications, are implemented as templates. Developing new monitoring application executable code using these templates allows the user to avoid writing the code for the standard input/output operations and focus on implementing the desired pattern-based monitoring method.

This version of GRTMS includes two templates with ANSI C code implementing input/output operations standard for all GRTMS monitoring applications: one template for generating monitoring application executable code implemented as dynamic-link libraries (DLLs), and the other template for the executable code implemented as standalone executable files. Executable file-based monitoring applications are more forgiving than DLL-based ones when there

are errors in the application code. If a monitoring application implemented as a standalone executable file and running as a separate process is abnormally terminated at run time, the GRTMM will stop the monitoring session and display an error message. If the same problem occurs with a DLL-based monitoring application, the GRTMM itself will be abnormally terminated by the operating system with potentially undesirable side effects. On the other hand, standalone executable files take considerably more time to run, are larger, use more hardware resources, and limit CPU management capability.

It makes sense to first develop a new monitoring application as a standalone executable file using the appropriate template, extensively test it with the GRTMS, and, after the tests have been passed, compile and link the code into a DLL file using the DLL template. The ability of the GRTMS to generate and run monitoring applications of both DLL and standalone executable types gives the user a greater convenience in developing new monitoring applications and improves the reliability of the system. More information on how to generate monitoring application executable code can be found in the GRTMS Operating Manual [Shapovalov and Whiteley, 1999], Section 2.4.

4.3.2 Configuring monitoring applications with Monitoring Application Editor

GRTMS-compatible monitoring applications have many adjustable user-specified parameters. Each monitoring application has a configuration file with specifications for all the adjustable parameters. The configuration file for a monitoring application specifies how to supply monitoring data to the application, run the application executable code, interpret the result of the diagnosis performed by the application, read the application-generated file with smoothened monitored variable trends, and how to display the trends of the monitored variables. Section 2.2 of the GRTMS Operating Manual [Shapovalov and Whiteley, 1999] contains a comprehensive description of the structure and format of GRTMS monitoring application configuration files.

A standard utility called Monitoring Application Editor was developed to help the user generate and edit monitoring application configuration files. Monitoring application file names, time-related application parameters, monitored variables, and the conditions to be diagnosed by the monitoring application are specified using the main window in the Monitoring Application Editor (see Figure 4.9). The Monitoring Application Editor also has a window where the user specifies special parameters for each monitored variable (see Figure 4.10), a window to set the parameters for displaying monitored variable trends (see Figure 4.12), and a window to set the order of smoothened trends in the smoothened data file generated by the monitoring application (see Figure 4.11). Operation of the Monitoring Application Editor is discussed in further detail in Section 2.3 of the GRTMS Operating Manual [Shapovalov and Whiteley, 1999]. Using the Monitoring Application Editor, a GRTMS-compatible monitoring application can be created within minutes by simply clicking control buttons, selecting menu items and colors, and filling in text boxes.

The GRTMS supports the use of flexible executable code for monitoring applications. The same executable code can be shared by several monitoring applications that use the same pattern-based monitoring method. When the GRTMM runs a monitoring application, the name of the application's internal configuration file is passed to the executable code of the application. At the same time, the default directory is changed to the one where additional application-specific data is contained (e.g., neural network weights). The executable code reads the internal configuration file together with the additional application-specific data and performs the classification according to the method being employed. Subsection 2.3.9 of the GRTMS Operating Manual [Shapovalov and Whiteley, 1999] explains how internal monitoring application configuration files are generated and formatted.

MONITORING APPLICATION EDITOR

File Help

APPLICATION TITLE: monitoring distillation column 05 SET ALL FILENAMES TO DEFAULT IN CURRENT APP CONFIG FILE

APP CONFIGURATION FILE: c:\cvi40\app1\task1.app SELECT

APP EXEC CODE FILE NAME: c:\cvi40\app1\ddqtask1.dll SELECT

APPLICATION DATA FILE: c:\cvi40\app1\ddqtask1.log SELECT

MONITOR RESULT LOG FILE: c:\cvi40\app1\ddqtask1.dat SELECT

INTERNAL APP CONFIG FILE: c:\cvi40\app1\ddqtask1.ctr SELECT

Smoothered data available: ☒ Update internal app config file when saving: ☒ Executable code is a Generic R-T Monitoring System - compatible *.dll file: ☒

SMOOTHENED DATA FILE: c:\cvi40\app1\ddqtask1.smt SELECT

App data file parameters: Data time window (hr : min : sec): 00:20:25 Data sampling interval (min : sec): 00:25 Data update frequency (min : sec): 00:25

Pattern generation: Pattern time window (hr : min : sec): 00:03:20 Pattern data sampling interval (min : sec): 03:20

Trendpad parameters: Time window for validation stripchart (hr : min : sec): 00:12:05 Time window for monitoring stripchart (hr : min : sec): 00:12:05

Application responses and their codes:

- ☒ UNKNOWN [0]
- ☐ FAULT [1]
- ☐ NORMAL [2]

Monitored variables (list of tagnames) in the same order as in the app data file:

- ☒ VAR3 [1] <V>
- ☐ VAR4 [2] <V>
- ☐ VAR5 [3] <M>
- ☐ VAR6 [4] <M>

Figure 4.9. The main window of the Monitoring Application Editor.

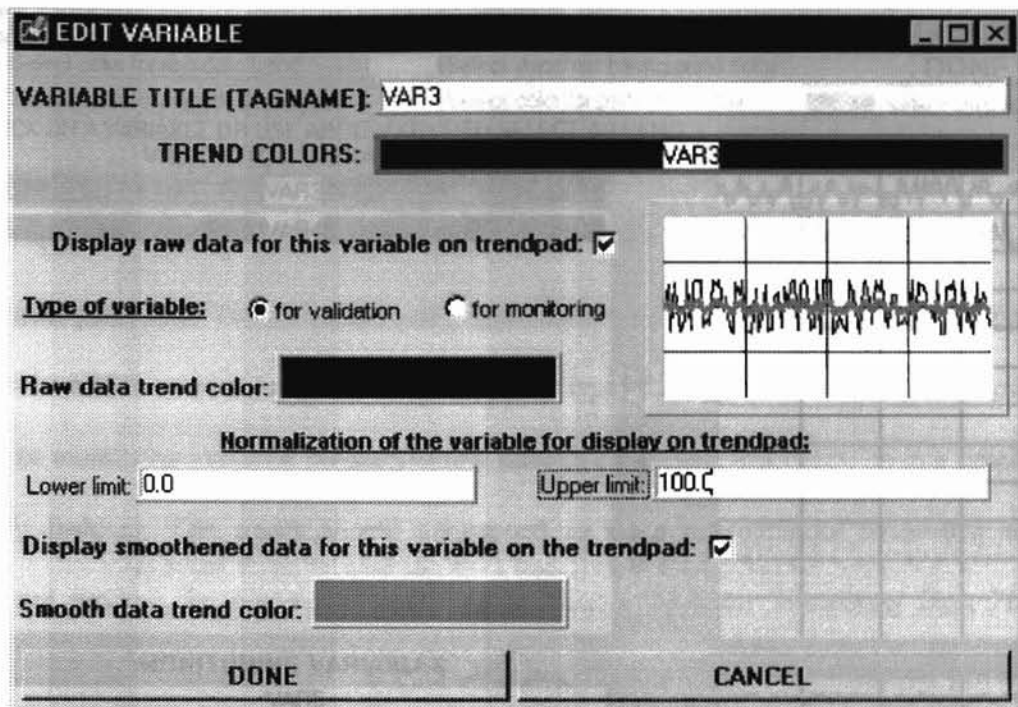


Figure 4.10. The window for editing parameters of monitored variables.

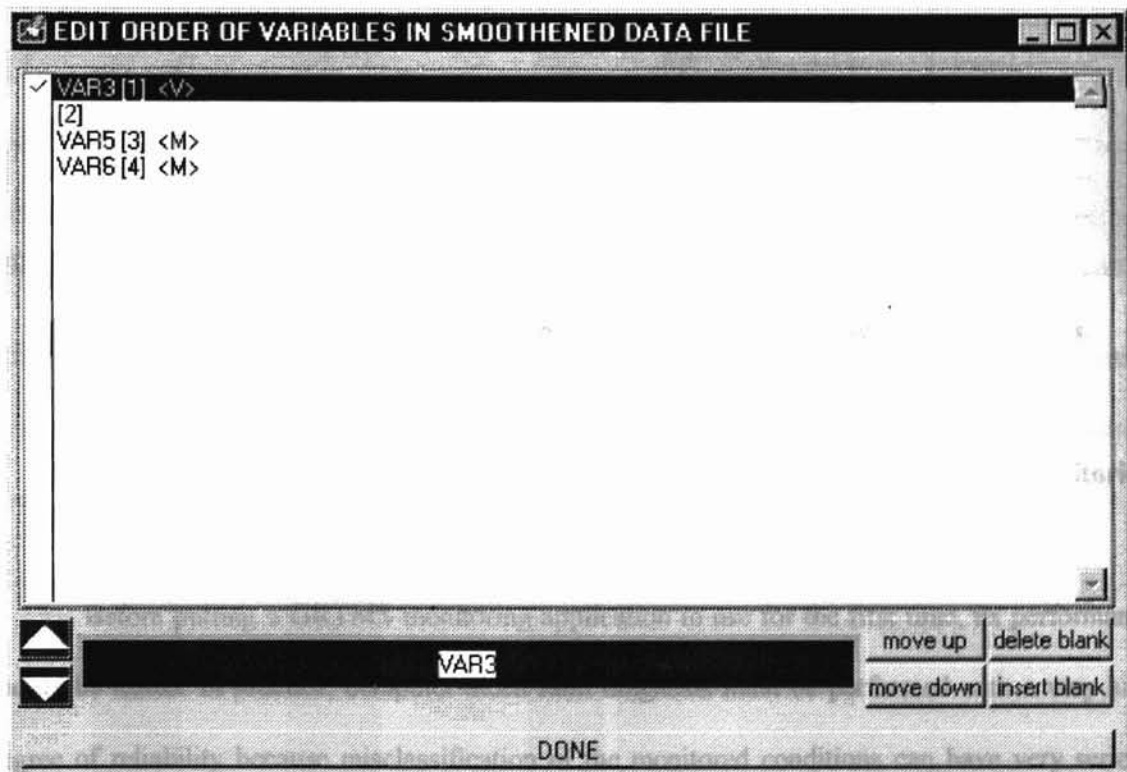


Figure 4.11. The window for setting the order of smoothened trends in smoothened data files.

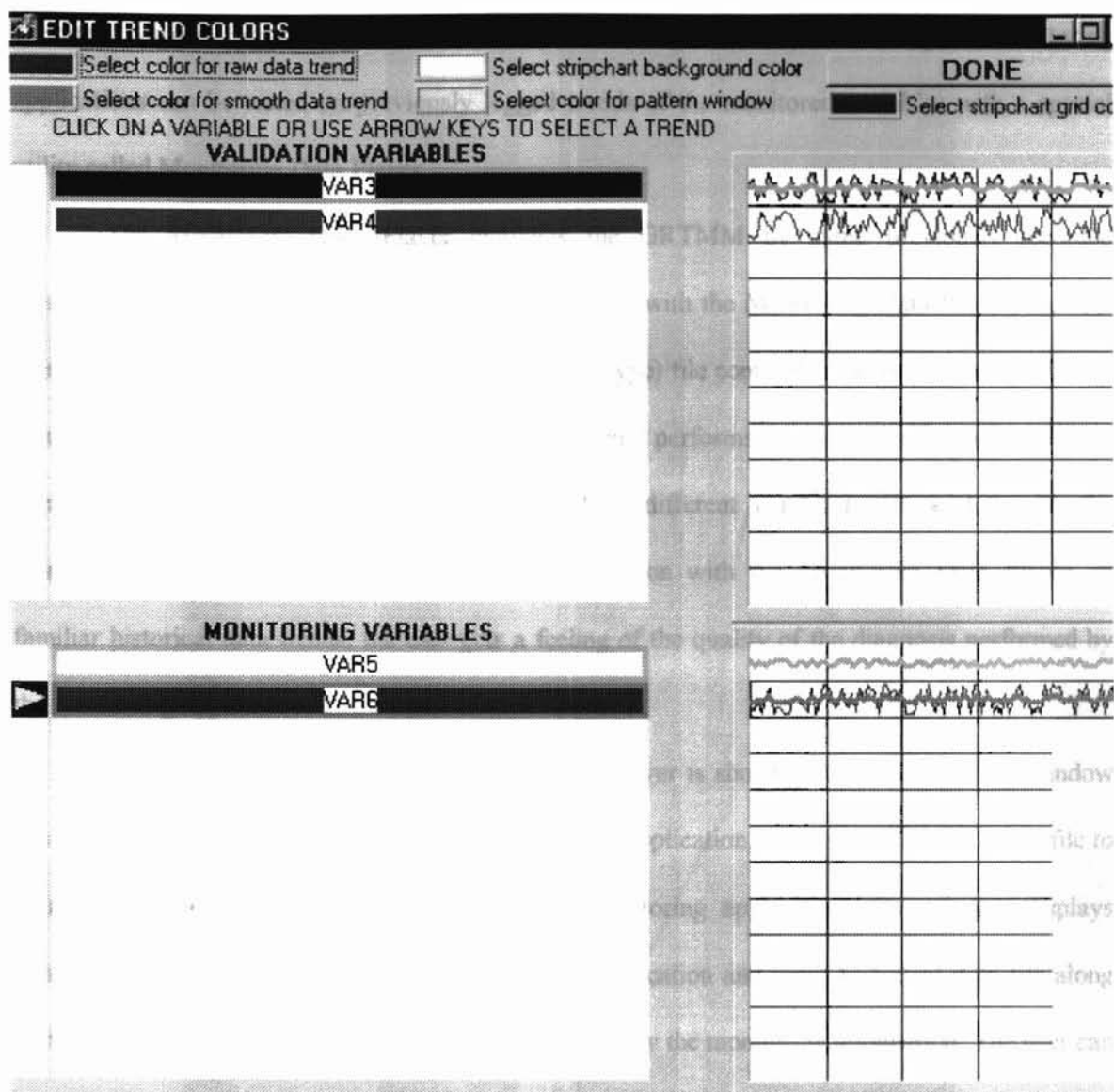


Figure 4.12. The window for setting parameters of displaying monitored variable trends.

4.3.3 Testing monitoring application performance on recorded data with Monitoring Data Player

Before putting a GRTMS monitoring application to use for the first time, its performance should be tested. In practice, computer-aided fault diagnosis must be performed with a very high degree of reliability because misclassification of the monitored conditions can have very serious consequences. There is also a need to ensure that newly generated monitoring application code

executes without run-time errors. In the GRTMS, operation of user-designed monitoring applications can be tested on previously logged trends of the monitored variables with a special utility called Monitoring Data Player.

The Monitoring Data Player emulates the GRTMM by using previously recorded (historical) plant data. To test a monitoring application with the Monitoring Data Player, the user creates an ASCII comma-separated variable (*.CSV type) file containing sampled historical trends of all the monitoring variables (Wonderware® Intouch® performs this operation in a simple, user-friendly fashion). The trends should correspond to different conditions associated with the monitored process. By running a monitoring application with the Monitoring Data Player on familiar historical data trends, the user gets a feeling of the quality of the diagnosis performed by the monitoring application.

The main window for the Monitoring Data Player is shown in Figure 4.13. The window has control buttons for the user to select a monitoring application, the historical ASCII data file to “play,” and controls for the “playback” of the monitoring application. The window displays detailed information about the selected monitoring application and input historical data file along with the current diagnosed process condition generated by the monitoring application. The user can control the playback speed of the Monitoring Data Player during an evaluation session. Just like with the GRTMM, the user can call the Trendpad to view the monitored variable trends currently being “played” by the Monitoring Data Player. Operation of the Monitoring Data Player is described in detail in Chapter 4 of the GRTMS Operating Manual.

MONITORING DATA PLAYER

Help

APPLICATION | DATA FILE | **PLAY** | PAUSE | REWIND | QUIT

STATUS BOX: playing data MESSAGE BOX:

MONITORING APPLICATION FILE:
c:\cvi40\app1\task1.app

APPLICATION VARIABLES:
VAR3,VAR4,VAR5,VAR6

HISTORICAL DATA SOURCE:
c:\cvi40\borger1\source1.csv

FIRST DATA SAMPLE: 08/29/95 15:31:57 MONITORING STARTED AT: 08/29/95 15:53:17 SAMPLING INTERVAL, S: 61

CURRENT DATA SAMPLE: 08/29/95 16:00:24

DATA PLAYING ON: ☐ PAUSED: ☐ END OF DATA: ☐ APPLICATION TIMEOUT, S: 60 PLAY SPEED: 1-30-max-
☐ Terminate an exe application on rewind and exit: ☐

APPLICATION TITLE: monitoring distillation column 05 LAST RUN: 16:00:17 MONITORING RESULT: UNKNOWN

TRENDS

Figure 4.13. The main window for the Monitoring Data Player.

4.3.4 Section summary

Section 4.3 described in brief the utilities included with this version of the GRTMS to simplify the process of developing pattern-based monitoring applications. This version of GRTMS provides help in three areas of development of a new pattern-based monitoring application: implementing a pattern-based fault detection method as a computer code, configuring a monitoring application, and testing the application performance. The most important features of these utilities that include the monitoring application templates, Monitoring Application Editor, and Monitoring Data Player, were described in this section.

4.4 Chapter summary

This chapter described our generic pattern-based real-time monitoring system called GRTMS. The system was developed from the principles outlined in Chapter 3. The implementation of simultaneous real-time generic pattern-based monitoring of several different processes in the GRTMS, using multithreading, run-time dynamic linking, and real-time data acquisition with DDE and NetDDE, was explained. The basic features of the GRTMS that allow the user to configure and control real-time monitoring sessions were discussed. This chapter also described the components of the GRTMS included with the system to facilitate the user's job of building monitoring applications: monitoring application templates, the Monitoring Application Editor, and the Monitoring Data Player. Operability of the GRTMS has been demonstrated on an industrial scale. Documentation of this demonstration is not provided for reasons of confidentiality.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Completion of this work resulted in development of a novel generic computer system for real-time pattern-based process monitoring. The developed monitoring system is called GRTMS (Generic Real-Time Monitoring System). The system consists of a GRTMM (Generic Real-Time Monitoring Module) that serves as a kernel for executing the user's monitoring applications, application templates that implement communication protocols for monitoring applications, a Monitoring Application Editor that can configure monitoring applications, and a Monitoring Data Player that can be used to test the performance of monitoring applications on previously recorded process data. The specific contributions of the work presented in this thesis are listed below.

- a) The system utilizes real-time programming techniques and runs on a PC under Microsoft® Windows® operating system.
- b) The system is capable of monitoring different processes with independently developed custom monitoring applications
- c) The system monitors multiple processes simultaneously using priority-based scheduling.
- d) The system communicates with monitoring applications using standard protocols.
- e) Facilities for developing new monitoring applications and customizing old applications to monitor new processes are provided.
- f) To perform real-time monitoring, the system runs as a single process with different tasks implemented as separate threads.
- g) The system uses run-time dynamic linking to attach monitoring applications.
- h) The system provides industrial real-time data acquisition capability.

The significance of the work presented is as follows.

1. This work described one of the first known generic, real-time, multitasking, pattern-based monitoring system.
2. The system proposed in this work facilitates application of new methods of fault diagnosis in the process industry.
3. The proposed monitoring system is an essential tool for developing and testing new monitoring methods.

5.2 Future work

This section lists the potential improvements to the proposed GRTMS.

1. The GRTMS can be made even more generic. The GRTMS can be enhanced to acquire real-time data from any data historian. This can be achieved if real-time data acquisition is performed using custom dynamic-link libraries that run as a separate thread. Besides real-time data acquisition, display of monitored trends can also be made more generic to allow the user to view trends of the monitored variables using his or her own custom dynamic-link libraries. If the GRTMM is made more generic, it will be more widely applicable, convenient to the user, and more suitable for further upgrades.
2. The facilities responsible for development of new monitoring applications can be upgraded and supplemented with new components. Monitoring application templates can also be implemented as object files that can be added to newly developed monitoring application executable code at the link stage. A user-friendly module that trains pattern-based monitoring applications to classify the monitored operation states, using recorded process data, can be added. The value of the GRTMS would increase further if the proposed training module supported incremental learning during real-time monitoring.

3. The user interface of the GRTMM and Monitoring Data Player can be improved. The main window and the window displaying operating states (the Taskbar) can be made user-friendlier while requiring less CPU time. One improvement that can make the GRTMM more convenient to the user is adding alarm notifications for each monitored process (if a monitored condition is classified as abnormal, the GRTMM would open a window explaining what is wrong and use alarm multimedia). The window that displays monitored variable trends (the Trendpad) can also be improved. Each strip chart can have axes with date and monitored variable value labels. The Trendpad of the Monitoring Data Player can display two monitored conditions: the actual one and the diagnosed one. With the proposed interface improvements, it will be easier to operate the GRTMM and evaluate the performance of monitoring applications tested with the Monitoring Data Player.

BIBLIOGRAPHY

1. S.M.Alexander and T.B.Gor, "Monitoring, Diagnostics and Control of Industrial Processes," *Computers and Industrial Engineering*, **35(1-2)**, pp.193-197 (1998).
2. J.M.F.Calado and J.M.G.Sá da Costa, "A Hierarchical Fuzzy Neural Network Approach for Multiple Fault Diagnosis," *Proceedings of UKACC International Conference on Control '98, 1-4 September 1998*, pp.1498-1503 (1998).
3. G.Carpenter and S.Grossberg, "ART2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics*, **26(23)**, pp.4919-4930 (1987).
4. G.Carpenter and S.Grossberg, "ART3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures," *Neural Networks*, **3(23)**, pp.129-152 (1990).
5. G.Carpenter, S.Grossberg, N.Markuzon, J.Reynolds and D.Rosen, "Fuzzy ARTMAP: A Neural Network Architecture for Supervised Incremental Learning of Analog Multidimensional Maps," *IEEE Transactions on Neural Networks*, **3(5)**, pp.698-713 (1992).
6. G.Carpenter, S.Grossberg and J.Reynolds, "ARTMAP: Supervised Real-Time Learning and Classification of Non-Stationary Data by a Self-Organizing Neural Network," *Neural Networks*, **4(5)**, pp.565-588 (1991).
7. G.Carpenter and W.Ross, "ART-EMAP: A Neural Network Architecture for Object Recognition by Evidence Accumulation," *IEEE Transactions on Neural Networks*, **6(4)**, pp.805-818 (1995).

8. A.E.Farell and S.D.Roat, "Framework for Enhancing Fault Diagnosis Capabilities of Artificial Neural Network," *Computers and Chemical Engineering*, **18(7)**, pp.613-635 (1994).
9. S.S.Ganti, "Automated Trend Extraction of Sensor Signals for Pattern-Based Data Analysis," M.S. Thesis, Oklahoma State University, School of Chemical Engineering, Stillwater OK (1996).
10. P.V.Goode and M.Chow, "Neural-Fuzzy Systems for Incipient Fault Detection in Induction Motors," *Proceedings of the 19th International conference on Industrial Electronics*, pp.332-337 (1993).
11. M.T.Hagan, H.B.Demuth and M.Beale, *Neural Network Design*, PWS Publishing Company, Boston, MA (1996).
12. J.C.Hoskins and D.M.Himmelblau, "Artificial Neural Network Models of Knowledge Representation in Chemical Engineering," *Computers and Chemical Engineering*, **12(9)**, pp.881-890 (1988).
13. H.-H. Huang and H.P.-B.Wang, "Integrated Monitoring and Diagnosis System for Roller Bearings," *International Journal of Advanced Manufacturing Technology*, **12(1)**, pp.37-46 (1996).
14. J.S.R.Jang, "ANFIS: Adaptive Network-Based Fuzzy Inference System," *IEEE Transactions on Systems, Man and Cybernetics*, **23(3)**, pp.665-685 (1992).
15. G.Karsai, S.Padalkar, H.Franke, J.Sztipanovits and F.Decaria, "Practical Method for Creating Plant Diagnostics Applications," *Integrated Computer-Aided Engineering*, **3(4)**, pp.291-304 (1996).

16. S.Kavuri and V.Venkatasubramanian, "Representing Bounded Fault Classes Using Neural Network with Ellipsoidal Activation Functions," *Computers and Chemical Engineering*, **17(2)**, pp.139-163 (1993).
17. S.Kavuri and V.Venkatasubramanian, "Neural Network Decomposition Strategies for Large-Scale Fault Diagnosis," *International Journal of Control*, **59(3)**, pp.767-792 (1994).
18. S.Keyvan, A.Durg and L.Rabelo, "Evaluation Of The Performance Of Various Artificial Neural Networks To The Signal Fault Diagnosis In Nuclear Reactor Systems," *1993 IEEE International Conference on Neural Networks, Publ. by IEEE*, pp.1719-1723 (1993).
19. H.Kirsch and K.Kroschel, "Applying Bayesian Networks to Fault Diagnosis," *Proceedings of the 1994 IEEE Conference on Control Applications Part 2 (of 3) Aug 24-26, 2*, pp.895-900 (1994).
20. S.Kleiman, *Programming with Threads*, Sun Soft Press, Mountain View, CA (1996).
21. J.Korbicz and J.Kuš, "Fault Detection and Isolation System Using GMDH Neural Network," *IEE Conference Publication Proceedings of the 1998 International Conference on Control, part 2 (of 2) Sep 1-4*, pp.952-957 (1998).
22. M.Kotani, H.Matsumoto and T.Kanagawa, "Hybrid Neural Networks as a Tool for the Compressor Diagnosis," *IEICE Transactions on Information and Systems*, **E76-D(8)**, pp.882-889 (1993).
23. J.Lee, "Modern Computer-Aided Maintenance Of Manufacturing Equipment And Systems: Review And Perspective," *Computers & Industrial Engineering*, **28(4)**, pp.793-811 (1995).
24. J.A.Leonard and M.A.Kramer, "Radial-Basis Function Networks for Classifying Process Faults," *IEEE Transactions on Control Systems*, **11(4)**, pp.31-38 (1991).

25. C.C.Lin and H.P.Wang, "Classification of Autoregressive Spectral Estimated Signal Patterns Using Adaptive Resonance Theory," *Computers in Industry*, **22(2)**, pp.143-158 (1993).
26. A.Linkens and M.F.Abbod, "Generic System Architecture for Supervisory Fuzzy Control," *Intelligent Systems Engineering*, **4(3)**, pp.181-193 (1994).
27. E.B.Martin and A.J.Morris, "Multivariate Statistics and Neural Networks in Process Fault Detection," *IEE Colloquium (Digest) IEE Computing and Control Division Colloquium on Qualitative and Quantitative Modelling Methods for Fault Diagnosis Apr 24*, pp.7/1-7/8 (1995).
28. E.B.Martin, A.J.Morris and J.Zhang, "Process Performance Monitoring Using Multivariate Statistical Process Control," *IEE Proceedings in Control Theory Application*, **143(2)**, pp.132-144 (1996).
29. Microsoft Corporation, *Dynamic Data Exchange (DDE): Microsoft Win32 Programmer's Reference*, Microsoft Press, Redmond, WA (1995a).
30. Microsoft Corporation, *Dynamic Link Libraries: Microsoft Win32 Programmer's Reference*, Microsoft Press, Redmond, WA (1995b).
31. Microsoft Corporation, *Memory Management: Microsoft Win32 Programmer's Reference*, Microsoft Press, Redmond, WA (1995c).
32. Microsoft Corporation, *Processes and Threads: Microsoft Win32 Programmer's Reference*, Microsoft Press, Redmond, WA (1995d).
33. D.Neogi and C.E.Schlags, "Multivariate Statistical Analysis of an Emulsion Batch Process," *Industrial and Engineering Chemistry Research*, **37(10)**, pp.3971-3979 (1998).
34. B.Ozyurt and A.Kandel, "Hybrid Hierarchical Neural Network-Fuzzy Expert System Approach to Chemical Process Fault Diagnosis," *Fuzzy Sets and Systems*, **83(1)**, pp.11-25 (1996).

35. J.Prock, "Computer-Based Sensor And Process Fault Diagnosis In Real Time," *IFAC Symposia Series Proceedings of the IFAC Symposium on Control of Power Plants and Power Systems Mar 9-11*, pp.177-182 (1992).
36. V.K.Raghavan, "Wavelet Representation of Sensor Signals for Monitoring and Control," M.S. Thesis, Oklahoma State University, School of Chemical Engineering, Stillwater OK (1995).
37. M.Rao, H.Yang and H.Yang, "Integrated Distributed Intelligent System Architecture for Incident Monitoring and Diagnosis," *Computers in Industry*, **37**, pp.143-151 (1998).
38. D.L.Reilly, L.N. Cooper, and C.Elbaum, "A Neural Model for Category Learning," *Biological Cybernetics*, **45**, pp.35-41 (1982).
39. R.Rengaswamy and V.Venkatasubramanian, "Integrated Framework for Process Monitoring, Diagnosis, and Control Using Knowledge-Based Systems and Neural Networks," *Proceedings of the IFAC Symposium on On-Line Fault Detection and Supervision in the Chemical Process Industries, Apr 22-24, 1992*, **1**, p.49 (1993).
40. R.Rengaswamy and V.Venkatasubramanian, "A Syntactic Pattern-Recognition Approach for Process Monitoring and Fault Diagnosis," *Engineering Applications in Artificial Intelligence*, **8(1)**, pp.35-51 (1995).
41. D.W.Russell, "Software Engineering and Production Monitoring Systems," *Journal of Systems Integration*, **4(3)**, pp.243-256 (1994).
42. R.A.Shapovalov and J.R.Whiteley, "Generic Real Time Monitoring System: Operating Manual," Technical Report, Oklahoma State University (1999).
43. M.A.Sharif and R.I.Grosvenor, "Process Plant Condition Monitoring and Fault Diagnosis," *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, **212(E1)**, pp.13-30 (1998).

44. A.J.C.Sharkey and N.E.Sharkey, "Combining Diverse Neural Nets," *Knowledge Engineering Review*, **12(3)**, pp.231-247 (1997).
45. Ch.-Sh. Tsai and Ch.-T. Chang, "Dynamic Process Diagnosis Via Integrated Neural Networks," *Computers & Chemical Engineering*, **19 nSuppl.1995**, pp.S747-S752 (1995).
46. M.A.Weiss, *Data Structures and Algorithm Analysis in C*, 2nd Edition, Addison-Wesley, Menlo-Park, CA (1997).
47. J.R.Whiteley and J.F.Davis, "A Similarity-Based Approach to Interpretation of Sensor Data Using Adaptive Resonance Theory," *Computers and Chemical Engineering*, **18(7)**, pp.637-661 (1994).
48. J.K.Won and M.Modarres, "Improved Bayesian Method for Diagnosing Equipment Partial Failures in Process Plants," *Computers and Chemical Engineering*, **22(10)**, pp.1503-1515 (1998).
49. J.Zhang and A.J.Morris, "On-line Process Fault Diagnosis Using Fuzzy-Neural Networks," *Intelligent Systems Engineering*, **3(1)**, pp.37-47 (1994).
50. J.Zhao, B.Chen and J.Shen, "Multidimensional Non-Orthogonal Wavelet-Sigmoid Basis Function Neural Network for Dynamic Process Fault Diagnosis," *Computers and Chemical Engineering*, **23(1)**, pp.83-92 (1998).

2
VITA

Roman Shapovalov

Candidate for the Degree of

Master of Science

Thesis: AN AUTOMATED PATTERN-BASED SYSTEM
FOR REAL-TIME PROCESS MONITORING

Major Field: Chemical Engineering

Biographical:

Personal Data:

Born in Moscow, Soviet Union, May 29, 1973, the son of Alexander Shapovalov and Elena Shapovalova.

Education:

Graduated from Secondary School # 218 of the City of Moscow in June, 1990; graduated with honor from Mendeleyev University of Chemical Technology of Russia with Diploma in Chemical Cybernetics in March, 1996; completed requirements for the Master of Science in Chemical Engineering, at Oklahoma State University in December, 1999.

Experience:

Worked as an interpreter and project manager with Joint-Stock Company "Special Bio-Physical Technologies" from 1995 to 1997 and as a Research and Teaching Assistant at the School of Chemical Engineering of Oklahoma State University from 1997 to 1999.

Professional Memberships:

Member of American Institute of Chemical Engineers.