A Wavelet-Based System for Event Detection in Online Real-time Sensor Data

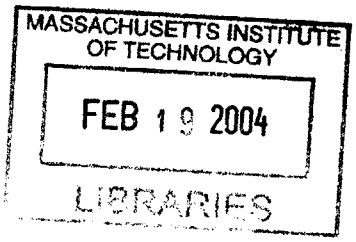
By

Charuleka Varadharajan
Bachelor of Technology, Civil Engineering (2001)
Indian Institute of Technology Madras

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the

Degree of Master of Science

Massachusetts Institute of Technology
February 2004



Author _____
Charuleka Varadharajan
Department of Civil and Environmental Engineering
January 16, 2004

Certified by _____
Steven R. Lerman
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by _____
Heidi Nepf
Chairman, Department Committee on Graduate Students

A Wavelet-Based System for Event Detection in Online Real-time Sensor Data

by

Charuleka Varadharajan

Submitted to the Department of Civil and Environmental Engineering on January 16th, 2004 in partial fulfillment of the requirements for the degree of Master of Science

ABSTRACT

Sensors are increasingly being used for continuous monitoring purposes, the process of which generates huge volumes of data that need to be mined for interesting events in real-time. The purpose of this research is to develop a method to identify these events, and to provide users with an architecture that will allow them to analyze events online and in real-time, to act upon them, and to archive them for future offline analysis.

This thesis is divided into two major portions. The first discusses a general software architecture that performs the functions defined above. The architecture proposed assumes no prior knowledge of the data, and is capable of dealing with multi-source data feed from any type of sensor(s) on one end, and can handle multiple clients on the other. The second part of the thesis discusses a wavelet-based algorithm for detecting certain types of events in real-time in one-dimensional numeric time-series data. Wavelets were judged to be the most appropriate technique for analyzing random sensor signals for which no prior information is available. The wavelet-based method in addition allows users to delve into different levels of abstraction (based on varying time periods) while looking at the data, which cannot be done by any previous method for real-time event detection.

This thesis also touches on the fundamental question of how one defines an event, which is more easily possible in a particular domain, for a specific purpose, but is much harder to do in a generic, domain-independent level.

Thesis Supervisor: Prof. Steven Lerman

Title: Professor of Civil and Environmental Engineering

ACKNOWLEDGEMENTS

I first thank my advisor, Prof. Steve Lerman for his constant guidance and understanding throughout the course of this project.

I am grateful to Prof. Kevin Amaratunga for sparing his valuable time in guiding me with the wavelets aspect of this project.

I thank Judson Harward and Kimberley De Long of CECI, and Prof. Ruaidhiri O'Connor for their help in this research.

I would like to express my appreciation to Microsoft Corporation and the Jewish Women's Archive for funding my stay at MIT through their research projects.

I also thank my mother, father, sister and the rest of my very huge family for supporting and encouraging me throughout my graduate study.

I finally thank my friends, particularly Rupa Das and Anita Komanduri for helping me out whenever I needed them and for having made my stay at MIT such a pleasurable one.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

1.1 MOTIVATION

In today's "intelligent world", an increasing number of environments are being monitored, as sensor costs decrease. This generates a huge volume of data, which then leads to problems of data storage. Hence there is a need to parse sensor signals for critical information that needs to be stored, thus throwing away unnecessary data. Also, one of the main purposes for which data is usually collected in monitoring applications is to alert users when interesting events happen, such as when something abnormal occurs in the data. This is the key motivation behind the event-detection system that has been developed as part of this research.

An example of a domain that could use such a system would be earthquake monitoring, where sensors detect ground motion. Even if one collected samples at a relatively slow rate of 1 sample a second, data would accumulate very quickly, most of which would be unnecessary. Usually the interesting portion of the data occurs during and around an earthquake. The event-detection system described in this thesis attempts to identify events, such as earthquakes, in the data. As high speed Internet is becoming cheaper these days, streaming data online would give users the ability to analyze it at any time, from anywhere in the world. By analyzing it in real-time, one could follow up on the events detected right away, like in this example to issue earthquake warnings when one is detected.

The main goal of this project is to identify in real-time, events of interest to those monitoring signals online, without focusing on any particular domain. Furthermore, the system should alert users when these events occur and if possible, predict future occurrences. Some applications already exist in the real world, which perform similar tasks on specific types of data, such as financial and weather monitors.

Finding events is a non-trivial task, mainly because a generic system designed to work for different domains, cannot assume any prior knowledge of the data. Events differ according to the domain - while some domains need to detect abnormal events, others seek trends in the data. Also, users might want to analyze the data with different resolutions – while some may be interested in finding events that occur on a daily basis, others may want to detect events that occur on a yearly basis. The methodology outlined in this thesis, is one of the first attempts that has been made in this field to address all these issues together.

1.2 ASSUMPTIONS

The following assumptions were made at the start of this project.

1. Data acquisition from sensors has been done by the user and the collected streams then fed into the event-detection system.
2. The events detected by the system are user-defined events of generic interest to all subscribers of the service. Hence the system outlined in this thesis is not intended to serve an individual subscriber's queries on the data. For example, while this system will identify abnormal wind velocities, it is not intended to serve subscribers who are

looking for wind speeds greater than 5mph. STREAM [1] – Stanford's research project on "continuous queries on streaming data" covers the topic of user defined queries comprehensively.

3. The user knows the resolution at which the data is to be sampled. Deciding the sampling resolution is an important step in monitoring signals for events. If the data is too coarsely sampled, events that occur in between samples might be missed, whereas if the data is too finely sampled, the volume of data generated might be too large to be efficiently processed. While the sampling resolution can be changed during the course of monitoring, it is assumed that the user knows what the initial and changed values of the sampling resolutions are.

1.3 RESEARCH OBJECTIVES

The following objectives outline the goals for a general system, and do not relate to any particular implementation of the methodology.

1. To identify and analyze events of interest in real-time and online, without any previous knowledge of the data.
2. To create a framework for event notification (i.e. raising alerts), that then cause subsequent trigger actions.
3. To archive these events in a database for future reference.
4. To predict what will happen in the future, by analyzing event patterns.
5. To evaluate the effectiveness of the event identification algorithm, using a domain specific implementation of the methodology.

1.4 RELATION TO THE iLAB INITIATIVE

The original idea behind this thesis evolved out of the iLab project, which is a part of the Microsoft funded iCampus program. The iCampus initiative aims to enhance university education through information technology. Its goal is to demonstrate leadership in higher education by sponsoring innovative projects which have a significant and sustainable impact at MIT and elsewhere. [2]

The iLab project aims to create a framework for professors to set up online labs easily. These remote laboratories will allow authorized users to carry out real-time experiments through the Internet. One broad category of laboratories that was identified by the iLab team is the "Sensor Lab", where continuous time-series data is generated by sensor sources, which is then processed, archived and transmitted to various clients. One example of this kind of a lab would be the Flagpole project at MIT. This experiment measures and streams real-time acceleration and temperature data of a flagpole over the Internet [3].

Raising alarms at unexpected events was identified as an important component of almost every type of laboratory in an initial iLab requirements analysis. However, its implementation was considered more crucial for the sensor labs, because of the following reasons.

1. A sensor experiment is usually set up with the primary idea of monitoring an environment, which implicitly involves mining the incoming data for events of

interest to the user. In some cases, the user might also be interested in changes that indicate if the equipment is functioning properly.

2. In some sensor experiments, data can be polled at event-driven intervals. For example, the temperature of a heat-exchanger might need to be monitored only when it exceeds a particular value.
3. In some experiments, the user might only be interested in these events and their onset/finish times for archival purposes. For example, an earthquake research experiment usually only needs the data that is measured during or just before and after an earthquake.

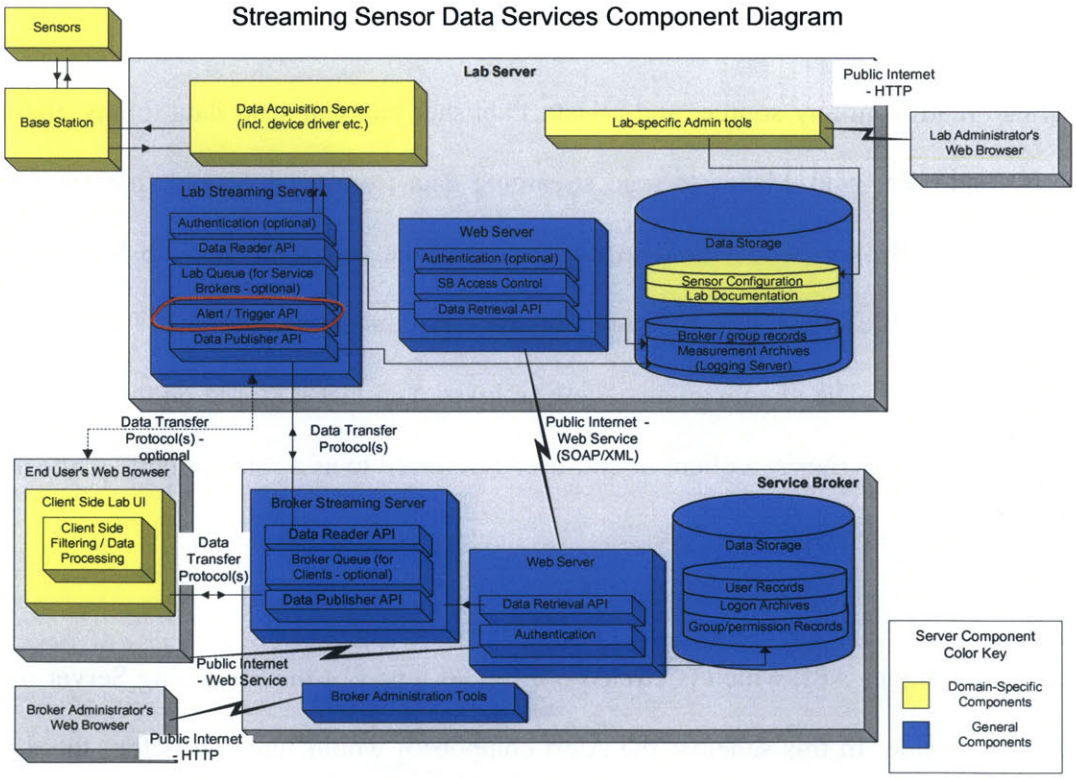


Fig. (1) Block diagram for Sensor Services Architecture

Fig. 1 describes the architecture that the iLab team initially developed for streaming sensor data. The architecture requires that there be two core components [4] –

1. A Lab Server - that is located in the laboratory, which interfaces with the hardware monitoring devices and other lab-specific equipment, as well as implement any domain dependent protocols.
2. A Service Broker – that acts as a liaison between the end-users and the Lab servers. This Broker implements most of the domain independent tasks that are required by a lab, such as user authentication, which will allow lab personnel to only deal with issues related to their domain, and hence ignore any intricate software issues.

Both these components contain a Streaming Server that mainly consists of a Data Reader that can read from any source, and a Data Publisher that can send data to any client. A client (end-user) who wishes to read streaming data from a lab would first connect to their Service Broker, who would in turn establish the link with the Lab Server. Once authenticated, the client has an option of directly connecting with the Lab Server to decrease latency. The client can also view archived data that is stored on the Lab Server. For the rest of this thesis, a client or end-user is referred to as a "subscriber", while a lab-user is simply referred to as a "user".

The Alert/Trigger API, which is circled in Fig.1, is a part of the Streaming Server on the Lab Server side. In this schema, the Alert component would function before the data is sent out to clients, so that users can be notified of any common events that are found. Detected events would then be archived in the data storage present on the Lab Server, and

subscribers of the service would be alerted. Those users who wish to access archived event data would use a web service call to the lab server to retrieve data from storage. The rest of this thesis focuses on the components that exist in the Alert API. Though the event-detection system that has been developed in this research can be used by non-lab applications, users can greatly benefit by integrating it into the iLab architecture, as some of the functionality required by them is then automatically taken care of.

1.5 THESIS OUTLINE

Chapter 2 contains some core definitions that are fundamental to understanding the problem behind event detection. It also describes the steps that lead to event identification in the methodology used in this project. The rest of the thesis is divided into two distinct parts. Chapter 3 discusses the software architecture that was designed for the event detection system. Chapter 4 discusses a wavelet based event detection algorithm and a scaled-down implementation of it that was developed for the purpose of this project. Though several techniques have been used to identify abnormalities online, none use wavelets. The idea of using wavelets follows naturally from some of the expectations of the algorithm. A detailed explanation of why wavelets were chosen for this project is included in Chapter 4. Chapter 5 finally concludes the material presented in the thesis and outlines possible future work.

CHAPTER 2 – EVENTS, INCIDENTS, IRREGULARITIES AND EDGES

2.1 EVENTS IN DIFFERENT DOMAINS

This section discusses a major portion of the literature survey that was done of events in various domains. The rest of the literature that was studied is summarized in Appendix 1.

One of the most common fields where event identification is used heavily is environmental monitoring. Earthquake detectors [6] try to detect earthquakes of potentially catastrophic magnitude (greater than 5.5 on the Richter scale), which are indicated by an increase in the frequency content and signal-to-noise ratio (SNR) of incoming data. The detectors are fed ground displacement (slip) information, which is collected using a GPS approximately every 30 seconds. Commercial weather monitors [7] attempt to find and predict phenomena such as hurricanes and storms by studying the temperature, pressure, wind speed and rainfall in an area. Air pollution monitors try to detect an unusual increase in the toxicity of air or trends in the decrease of the ozone layer. For example, an interesting study [8] found that in recent times, the rate of decrease of the ozone layer has slowed down.

In the domain of traffic monitoring [9], sensors typically read the speed, density and flow rate of traffic in intervals of about 20 to 30 seconds. Event detection techniques are then used to identify bottlenecks in the traffic, which usually occur when the traffic density in an area suddenly increases and the speed of vehicles decreases.

Event identification techniques are also used in the field of network intrusion detection [10]. Common network attacks include reconnaissance (ping sweeps, TCP port scans, etc.), exploits (hackers taking advantage of bugs to gain access to the system) and denial-of-service attacks (overloaded CPU's or network links that prevent machines from being utilized effectively). These attacks can be detected by studying some features of the network such as the number of TCP packets, expired requests, CPU utilization etc. for any unusual occurrences.

Electrocardiogram signals [11, 18] measure the electrical activity in the heart. ECG monitors, that read data from 2-3 leads, try to detect occurrences of cardiac arrhythmia (when the heart does not pump enough blood to the rest of the body) or ischemic episodes (when the body does not get enough oxygen). These are usually indicated by an unusual variation in the QRS complexes (arrhythmia) or an ST segment depression (ischemia), which are calculated by measuring the duration of the waves and intervals in an ECG signal (PQRS and T waves are the responses measured by the electrodes in an ECG that are generated in a normal cardiac cycle [11]).

Cell phone fraud detectors [12] try to pin-point fraud by mining unusual occurrences in multi-channel phone usage data such as location, duration of calls, number of calls made, numbers called and time of calling. The chief concern here is to not generate large numbers of false alarms that can annoy the subscribers.

2.2 KEY DEFINITIONS

The first step towards identifying events is to clearly define what an "event" means. The following definitions specify "*events*" in the context of this project, as well as three other levels that can be used to detect events, namely "*incidents*", '*irregularities*" and "*edges*". These four terms are often used in various contexts in literature as they are very common words in the English language. The literature in this field also uses several alternate words in place of the above terms to refer to the same concepts. Hence, these four definitions had to be specified, in order precisely express and understand the concepts they represent in this project.

2.2.1 Events

Most of the past research in related topics only defines events in the scope of particular domains. In some cases the rules for determining when an event should be generated are well known (for example, the temperature exceeding a pre-defined threshold). However, most of the times defining an event by such explicit rules is not possible, since the phenomenon is either ill-understood or has very unpredictable behaviour.

The literature survey of events in different domains that was described in Section 2.1 reveals two things. The first, which is intuitively obvious, is that different domains seek different types of events. Secondly, even in a particular domain, with the same data, users classify events in different ways, such as in network intrusion detection. Some users want outliers in the data (abnormal values due to a change of state of the system for a brief period of time), such as in ECG monitoring [18]. Others define events as trends (changes

to another state that is the new behaviour of the system) or changes in trends, such as in traffic incident detection.

For this project, an event has been defined as "*a qualitatively significant change in the behaviour of the data, as defined by the domain user*". This is a broad enough definition, which includes trends and outliers, and is flexible enough to cover the various types of events that could possibly occur. It is however, hard to make it more specific, because the nature of the problem at hand is so general.

The scope of this project has been restricted to that class of problems where events are either outliers or sharp trend changes. This was done because the literature survey showed that most events fall under these categories. There are some cases, like the ozone layer study, where a very smooth trend is the event that is to be detected. The definitions that follow will not help in identifying these types of events.

2.2.2 Incidents

Since an event, as defined above, depends on the user, a domain-independent level was needed for a generic event-detection mechanism. Hence, "incidents" were defined as "*the points of transition between the states of the system observed at a particular resolution*". The states need not necessarily be physical, but can be abstract, human-defined conditions of the data, such as high traffic and low traffic. The states also vary according to the resolution at which the data is analyzed. For example, the states that exist in data

analyzed on a yearly basis will be very different from the states in data analyzed on a monthly basis.

Incidents can either be stand alone events or can be combined to produce new events. In other words, an event is constructed of incidents, with every event having at least one incident. The idea behind defining an incident is that, while various users might disagree on what exactly an event is for their purpose, they will all mostly agree on what the incidents are. Since incidents are only a step on the path of identifying events, the users will then have filter out or combine these incidents to find the events that they are ultimately interested in. This allows for the same set of incidents to be combined in different ways to produce different types of events.

An analogy to the incident-event relationship is the Windows UI event of a mouse-click, which consists of mouse-down and mouse-up moves. The mouse-click is the "event" consisting of the mouse-down and mouse-up "incidents".

2.2.3 Irregularities

The "event" and "incident" definitions stated above are abstract concepts, which however are not mathematically useful. Hence, "irregularities" were defined as "*local deformations in the signal that represent a different state of the system, in the scale of the analysis resolution*". Incidents and events are global, in the sense that they are detected by taking into account all the data from the start of accumulation. Irregularities on the other hand, are detected by analyzing the signal locally (i.e. in a particular time interval,

which is based on the analysis resolution). Mathematically, these local variations can usually be detected by taking a derivative of the signal in the interval. Section 2.3 contains more on the mathematical aspects of irregularities.

The start and end (if it exists) of irregularities are marked by incidents, since an irregularity represents a new state of the system and incidents are transition points between states.

Some of the common types of irregularities encountered in signals are listed below. Figures 2 through 7 were taken from Mallat et al [5] to illustrate the concepts.

2.2.3.1 Cusps

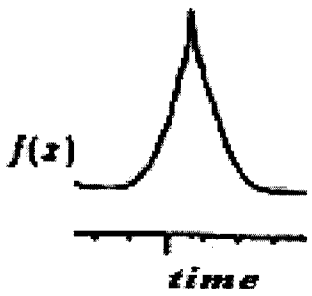


Fig. (2) Cusps

Cusps are sharp turning points in the data. For example, in EEG signals, cusps exhibit the accelerations and decelerations of heart beats. Mathematically, a cusp is defined as a point on a continuous curve, where the tangent vector reverses its sign. [13]

2.2.3.2 Diracs

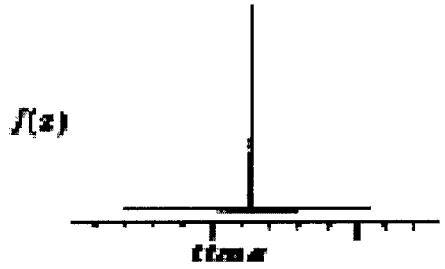


Fig. (3) Dirac Impulse

Dirac impulses are sharp variations in the signal that occur in an almost instantaneous (or very short) period of time. They frequently occur in electrical signals. [13]

2.2.3.3 Sharp Step Edges

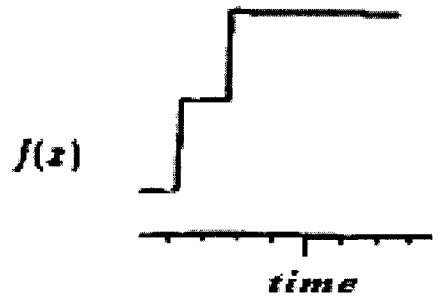


Fig. (4) Sharp Step Edges

In 1-D signals, step edges represent sudden jumps in the values of the data. In 2-D image data, edges refer to a rapid change in the image intensity. Step edge detection in 2-D data is heavily used in machine vision systems for object recognition and motion detection purposes. [5]

2.2.3.4 Smooth Step Edges

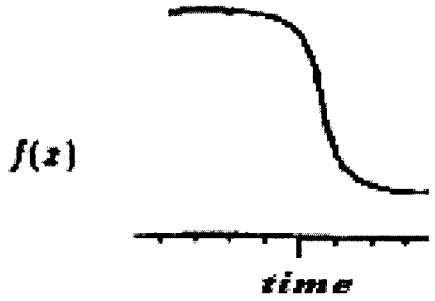


Fig. (5) Smooth Step Edges

These are usually generated when signals are passed through a smoothing filter, resulting in a smoothing of the sharp edges. They might also represent transitions (jumps) in the signal that take place over a period of time.

2.2.3.5 Fractal Type Edges

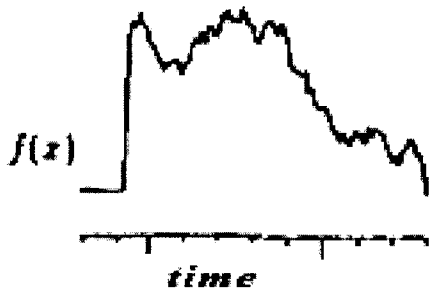


Fig. (6) Fractal Edges

The jagged edges in the data shown in Fig. 6 usually have fractal-like properties. Since sensor data in the real-world is discretely sampled, a lot of signals end up having these fractal-like edges (instead of continuous curves).

2.2.3.6 Singularities with Fast Oscillations

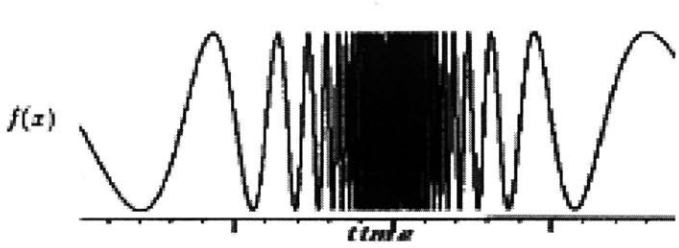


Fig. (7) Oscillating Singularity

These occur when there is a sudden change in the frequency content of the signals, such as in earthquake signals. In these cases, the entire period where there is a jump in the frequency of the signal, is considered to be the irregularity.

2.2.4 Edges

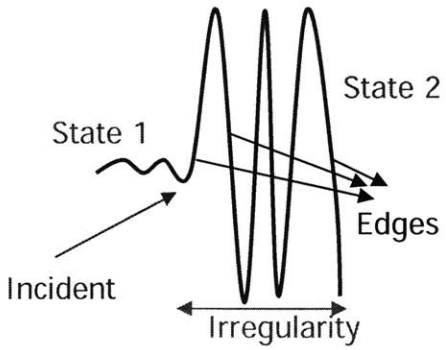


Fig. (8) Edges

For our purposes, edges are defined as "*the lines that mark the local change in behaviour of the signal*". These were required to algorithmically detect the incidents (i.e. the start and end points of irregularities), which can identified by traversing the edges that mark the boundaries of the irregularities. In Fig. 8, each of the lines in the oscillation represents an edge. An irregularity contains at least one (in the case of a jump), and usually multiple edges.

2.3 RELATION BETWEEN EVENTS, INCIDENTS, IRREGULARITES AND EDGES

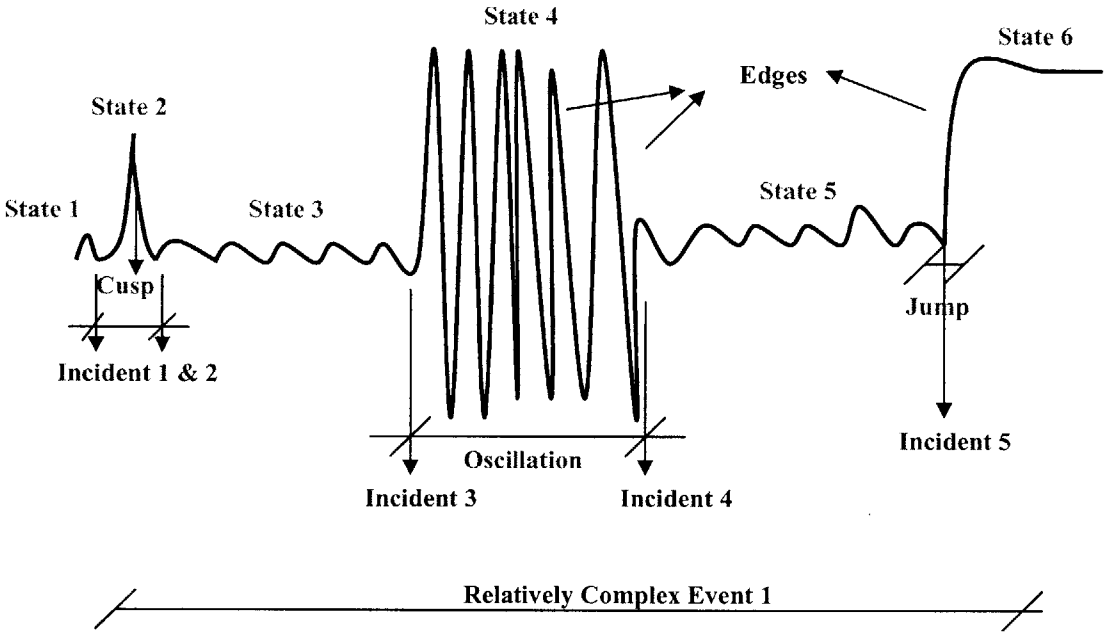


Fig. (9) Possible response signal of a hypothetical electronic circuit

Fig. 9 is an example of a possible response signal of a hypothetical electronic circuit, where an initial power spike is followed by an oscillation, finally culminating in a steady state response. There are three irregularities in the figure – a cusp, an oscillation and a jump. The cusp has two edges, while the oscillation has several. The start and end of both these irregularities are marked by incidents. The jump however contains only one edge, which is the irregularity and the incident in itself. In Fig. 9, incidents 1 through 5 mark a single event. However, it is possible if required, for another user to combine incidents 1 through 4 as a different event.

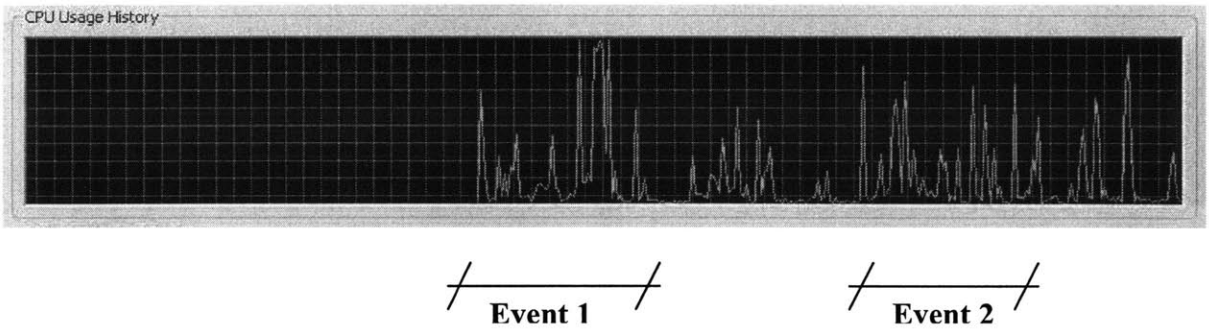


Fig. (10) CPU Usage History

Another real world example, to illustrate the relationship between these four concepts, is the graph of a computer's CPU usage. Fig. 10 contains several irregularities, and hence incidents and edges, but only a portion of those form events which indicate the times at which the machine was unusually busy.

The relationship can also be thought of as a partial hierarchy (depicted in Fig. 11).

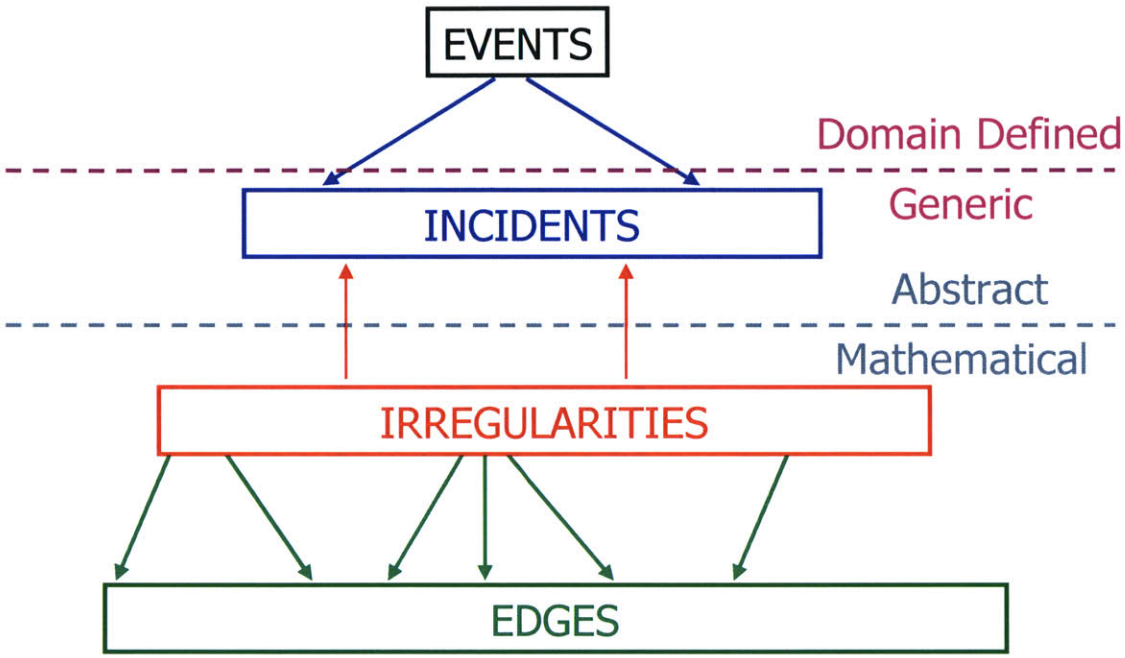


Fig. (11) Hierarchical Structure of concepts

Users ultimately want to detect domain-defined events, which consist of incidents, a definition independent of the domain. Incidents are indicated mathematically by irregularities, which consist of edges. In the rest of this project, the procedure that is followed in detecting events is to first find the edges in a signal, followed by the irregularities, then the incidents and finally the events.

2.4 MATHEMATICAL CHARACTERIZATION OF IRREGULARITES

In mathematics, the local regularity (or irregularity) of a function is measured with Lipschitz exponents, also referred to as Hölder exponents (which are used in multifractal analysis). [5]

Definition [5] :

- A function *f(x)* is said to be Lipschitz α at a point x_0, if and only if there exists two constants A and $h_0 > 0$, and a polynomial of order n (where n is a positive integer, $n \leq \alpha \leq n+1$), P_n such that for $h < h_0$

$$| f(x_o + h) - P_n(h) | \leq A | h |^{\alpha} \qquad (1)$$

- A function *f(x)* is uniformly Lipschitz α over the interval]a,b[, if and only if there exists a positive constant A and for any $x_0 \in$]a,b[there exists a polynomial of order n, $P_n(h)$, such that equation (1) is satisfied if $x_0+h \in$]a,b[
- The Lipschitz regularity of *f(x)* and x_0, is the superior bound of all values α such that *f(x)* is Lipschitz α at x_0
- A function is singular x_0, if is not Lipschitz 1 at x_0

Lipschitz exponents characterize the behaviour of *f(x)* around a point x_0, and are smoothness measures of the function. They are related to the order of differentiability of *f(x)* at x_0. The sharpness of an edge can be found by measuring its Lipschitz exponent. For example, a signal that is differentiable once has Lipschitz exponent 1. Hence, any function that is singular (non-differentiable) at x_0 is not Lipschitz 1 at x_0. A step function (jump) has Lipschitz exponent 0 and a dirac impulse has Lipschitz -1. Negative Lipschitz exponents correspond to sharp irregularities. If *f(x)* is Lipschitz α at x_0, $\alpha > n$, then *f(x)* is *n* times differentiable at x_0 and the polynomial $P_n(h)$ is the first *n+1* terms of the Taylor series of *f(x)* at x_0. Not only does the Lipschitz exponent give a measure of the function's differentiability at x_0, it also says that if a function is Lipschitz α at x_0, and if $n < \alpha < n+1$, the nth derivative of *f(x)* is singular at x_0 and α characterizes its singularity. [5]

CHAPTER 3 – SOFTWARE ARCHITECTURE

This chapter describes the software architecture that was designed for a generic event detection system. The following sections discuss at length the requirements analysis that was done, the possible design options that were available and consequent decisions that were made as part of this project.

3.1 PROCESS OVERVIEW

The typical sequence of activities that take place during the process of data transfer from the sensor sources to the subscribers, in an event detection system are as follows:

1. Data Acquisition from the sensors/base-stations
2. Data Preprocessing (such as de-noising and compression)
3. Event detection through filtering
4. Post processing of event data, which involves
    a. Archiving of events
    b. Event notification and consequent trigger action(s)
    c. Event prediction based on patterns discovered in the events

Any whole event detection system must allow for these processes in its design, though it is not necessary that an implementation of it must contain all these parts.

3.2 SYSTEM REQUIREMENTS ANALYSIS

The minimum requirements of a domain-independent event identification system that should be capable of handling any type of sensor data are:

1. Flexibility – Since it would be almost impossible to design an event detection algorithm that would work for all events in different domains, the architecture must have the provision for authorized users to install custom filters (or other components) which work for their domain. The system should also be flexible because some users might not require all the components, while some others might want to move around the order in which the processes are executed. Hence this is one of the most important requirements of a generic event-detector.
2. Expandability, scalability and portability - The system should be expandable in the sense that it should be able to handle any number of data sources (sensors and base stations) as well as support new sensor data types. Scalability refers to the system being capable of easily handling an increase in its subscribers. Portability means that it should be possible to make the system work anywhere and hence its design must be platform and operating system independent.
3. Distributed architecture – Users might want to have some components of the system on different machines (like the database, and sometimes the filtering service). Hence the architecture must support distributed elements of the system.
4. Different data formats – The system should be able to handle data from any type of sensor; and hence deal with the various data formats (such as 1-D numeric data, images etc.) the sensor signals might generate.
5. Multiple streams of incoming data - The system should be able to simultaneously process multiple, time-varying data sequences (that might be sampled at different rates), which represent measurements of various attributes of the monitored system. Time synchronization becomes a very important issue while dealing with multi-

stream data. These sequences need not necessarily be independent, and could depend on each other in two ways. The first is that an event for the system might have to be raised only if an event is detected in all (or a combination) of the input streams. For example, in a traffic incident detection problem, a bottle-neck should be detected only if the speed of vehicles decreases AND the density increases. The second manner in which sequences could depend on each other, is when they are correlated components of an attribute measure. For example acceleration components X, Y and Z might have to be analyzed in conjunction, since they together form the net acceleration. The problem with dependent sequences is that they lose their correlation, if analyzed independently. The results should also show up immediately (as each sequence is processed), even if each of these sequences are indefinitely long.

3.3 DESIGN OPTIONS AND DECISIONS

The following subsections describe the design options that were available for satisfying each of the requirements and the rationale for the final option that was chosen.

3.3.1 Flexible architecture

Requirement 1 was mostly satisfied by having a loosely coupled architecture with a set of autonomous but cooperating components. A set of generic interfaces (described in detail in Section 3.5) between these components allow them to be connected seamlessly.

This will enable users to fit their custom-designed components (such as filters) into the system, without much effort on their part. The design issues involved with custom filters are explained below.

3.3.2 Custom Filters

Custom filters bring about additional issues of security and authorization, since unauthorized users might use them as an opportunity to hack into the machine. Even authorized filters could place excessive demands on the resources of the computer and hence might either need to run on a separate machine, or on a low priority thread. Alternatively, custom filters could exist on the machine of the user who wishes to install it, which might be preferable if many external users are allowed to plug in their own filters. This would also circumvent the security issues to a large extent. The trade-off in separating out the filtering service is that the data would have to traverse an extra-roundtrip each time. This, however, would not be a big problem if the bandwidth is high.

For this version of the design, it has been assumed that the lab and the user will negotiate apriori the terms on which the custom filter is to be placed. A more efficient protocol for custom filter installation needs to be developed once more elaborate security procedures have been adopted into the design.

3.3.3 Expandable and Scalable Architecture

Requirement 2 can partially be tackled by having independent components that handle the input from multiple sources (data readers) and output to multiple clients (data publishers). If this system forms a part of the Alerts Module of the iLab sensor system that was described in Chapter 1, then this requirement is taken care of automatically.

3.3.4 Data Formats

Not only do sensors have different data types, but they also bundle the data differently. For example, sensors usually give out "n" readings in a batch followed by some sensor-dependent intermediate values (such as channel information, parity bits etc.), where n varies according to the sensors. XML could be used to bring this into a common format that the event system can interpret. There are two ways in which this could be done. The first is to encode the readings in XML and send them across the network. Some software like National Instruments' LabView have the provision of converting sensor streams into XML. However, there is a huge overhead involved in the parsing of XML data which makes this option undesirable. The second way is to send XML metadata about the sensor streams across, whenever a connection is made or when something in the event system is changed (for example, when the sampling rate of sensors is increased). This way, less expensive byte data can be exchanged between components, with the XML metadata allowing them to interpret the incoming data correctly. Appendix 2 has an example of an XML metadata file that could be used to describe sensor signals.

3.3.5 Multiple Streams

Multiple data streams might need to be analyzed simultaneously (dependently or independently) to raise an event. This is an important problem to solve, since most of the sensor applications in the real world have different types of sensors working together.

The main issue encountered here is that of time-synchronization, since sensor streams might be sampled at different rates. There are two ways of tackling this. The first is to

assume that the user will ensure that the clocks on each sensor source and the event-system are synchronized, which is a lot of work for the user. Alternatively, the time-stamps of the incoming data can be normalized with respect to each other, but that would require the user to feed in the clock-times of the various sources, which is again not preferable. To avoid this problem to a certain extent, an internal system time-stamp of the incoming data could be done, which allows for a uniform basis of comparison within the event-system. Section 3.4.3 describes the components of the filtering service that handle some of the issues of multiple streams.

The second major issue related to multiple streams is that of when an event exactly occurs. Some system events might need to be generated based on a combination of events in different streams. The problem then arises because of the different sampling rates of incoming streams. For example, temperature readings might be read every 1 sec, while pressure readings might be read every 1.5 sec, with a phase lag of 0.6 sec. The readings can then be compared either by interpolation or by matching them in a time-window. It has been assumed, for this version of the design, that the user knows and specifies the method they prefer and its parameters, such as the length of the time-window or the method of interpolation.

3.4 SOFTWARE ARCHITECTURE COMPONENTS

This section describes the components of the architecture that was developed as part of this project for the event detection system.

3.4.1 Data Acquisition Service

The data acquisition module serves two main purposes:

1. To read multiple streams of data from any type of sensor or base station.

2. To normalize the data into a format which the rest of the system can understand.

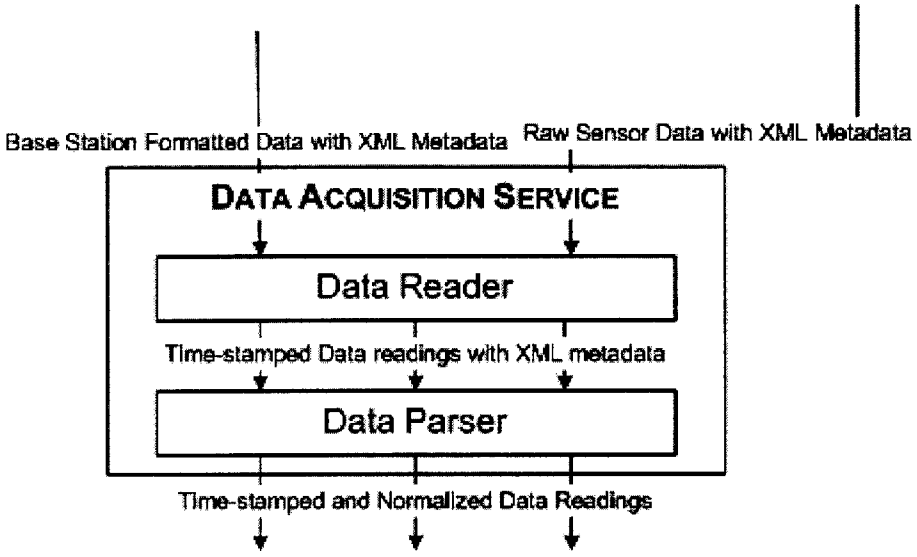


Fig. (12) Data Acquisition Module

It is assumed that the data source has a unit that can generate the XML metadata about the sensor stream, which is input into the system along with the data streams. The Data Acquisition module contains 2 components:

1. Data Reader – This gathers data from various sources and time-stamps the readings. The time-stamp needs to be done for two reasons. The first being that it can be used, to compare times for internal purposes, which will avoid issues of time-synchronization

between the different sources and the event-system. The second reason is that it cannot be assumed that all the incoming sensor data will be time-stamped by the source, which is why the system needs to time-stamp it. If the there is more than one Data Reader for the system, distributed over different machines, then it is assumed for this round of the design that the user will synchronize their clocks to generate similar time-stamps.

2. Data Parser – This is a translation service that uses the XML meta-data to convert readings into values that are of a common format that is independent of the sources, which the rest of the system can understand. Any other component of the architecture that needs the XML metadata to understand sensor readings must have a Data Parser.

3.4.2 Preprocessing Module

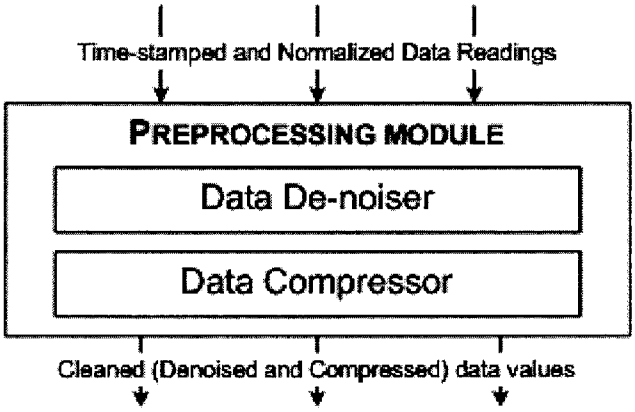


Fig. (13) Preprocessing module

The main purpose of this module is to clean up and compress the data and to perform any other operations that the user might want (such as basic pre-filtering of the data).

Prefiltering (with a low pass filter) would help to capture some of the smoother trends in the signal. These components are designed to be optional since some domains might not prefer to compress or de-noise the data, because critical information might be lost during these processes.

3.4.3 Filtering Service

The main purpose of the filtering service is to detect events that are of interest to the user. This service could on a separate machine (physically), if the computational needs of the algorithm exceed the capacity of the main machine.

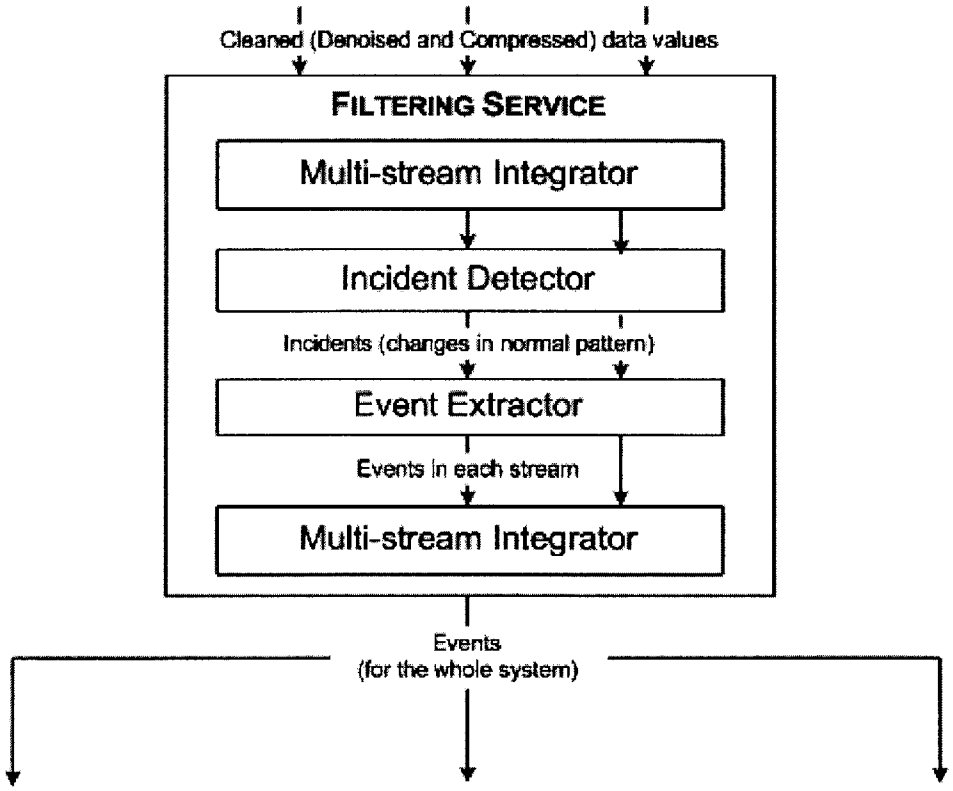


Fig. (14) Filtering Service

The filtering service has the following components:

1. Multi-stream integrator I – This is the first of the components that tackles the issue of multiple incoming streams. Its purpose to reduce the number of dependent sequences into a much smaller number of uncorrelated streams by combining them in a manner specified by the user. For example, acceleration components X, Y and Z can be combined to produce a single acceleration stream.
2. Incident Detector – This component does the main filtering job and detects incidents (as defined in Chapter 2) in each stream. Chapter 4 describes this project's implementation of this component using wavelets.
3. Event Extractor – This is a user -defined component, which prunes out the events (as defined in Chapter 2) that the user wants from the sequence of incidents generated by the Incident Detector.
4. Multi-stream integrator II – This is the second component that deals with the multiple streams issue. Its purpose is to generate "net system events" based on boolean logic (e.g. AND/OR) conditions, as specified by the user.

Alternatively, users could choose to integrate their custom filters instead of the one described above. Users could also have another, more accurate offline filter, for detailed event processing (shown in Fig. 16), which could use the archived data as its input.

3.4.4 Post Processor Module

The post processor module contains a collection of components that act on the events that have been detected by the filtering service.

3.4.4.1 Data Archive

The main purpose of the data archive is to store the events that occur in individual streams, as well as net system events. It also archives the start and end times of these events, as well as any other fields the user might want to store (such as event type).

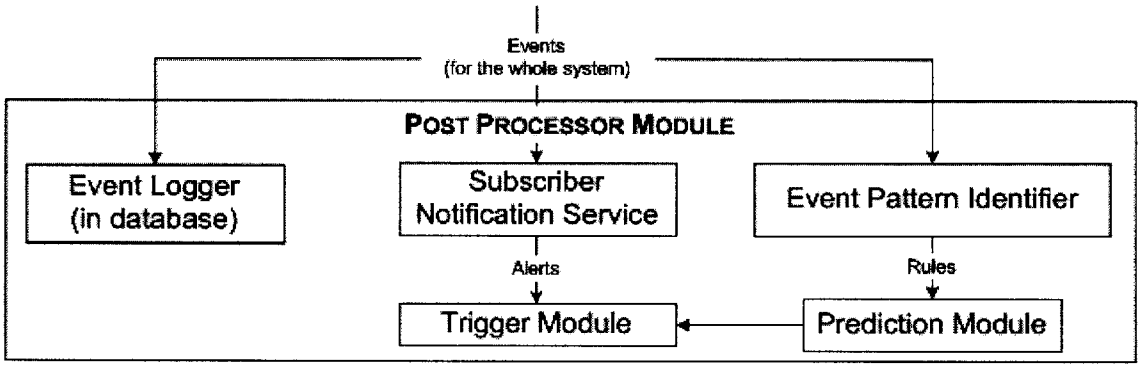


Fig. (15) Post Processor Module

There are two possible ways in which events could be recorded

1. Events by Reference – This method of archival assumes that the individual sensor data readings are being logged elsewhere in the system, for example by the iLab Sensor Architecture's Data Logger. So the event logger would only store the ID's of the sensor readings, which would then point to the Sensor Reading Log.
2. Events by Copy – This archive will store the attributes of the sensor readings (such as Sensor ID, time stamp, value etc.) that are part of an event in the event log. This may be needed in situations where the sensor generates so much data that the lab might not want to store individual sensor readings, but only the events of interest.

3.4.4.2 Event notification and Trigger action

Users usually want to be alerted when events occur, which is handled by the subscriber notification service. Authorized subscribers to the alert service should be notified by a means of their choice (email, pop-up windows etc.). The system should also allow alerts of different levels to be issued based on access permissions, and should be able to handle notification to individuals as well as user groups. The iLab Sensor Architecture's Data Publishers tackle of the issue of multiple subscribers. Custom filters also bring up the question of who can view the custom filtered data, since the filters' owners might sometimes want only certain subscribers to have access to the alerts.

Triggers, on the other hand, are different from alerts in that while alerts merely notify users about an events' occurrence, a trigger actually causes some follow-up action. An example of a trigger would be an executable program, which runs when an alert is raised, that consequently performs the desired action.

The trigger module takes as input the event stream that has been generated by the filtering service, as well the output stream of the prediction module (based on what the system thinks might happen). Triggers can alter the behavior of the system itself, as well as the source environment. For example, the sampling rate of the signal could be increased once an event occurs, to observe it more closely. Another example of a possible

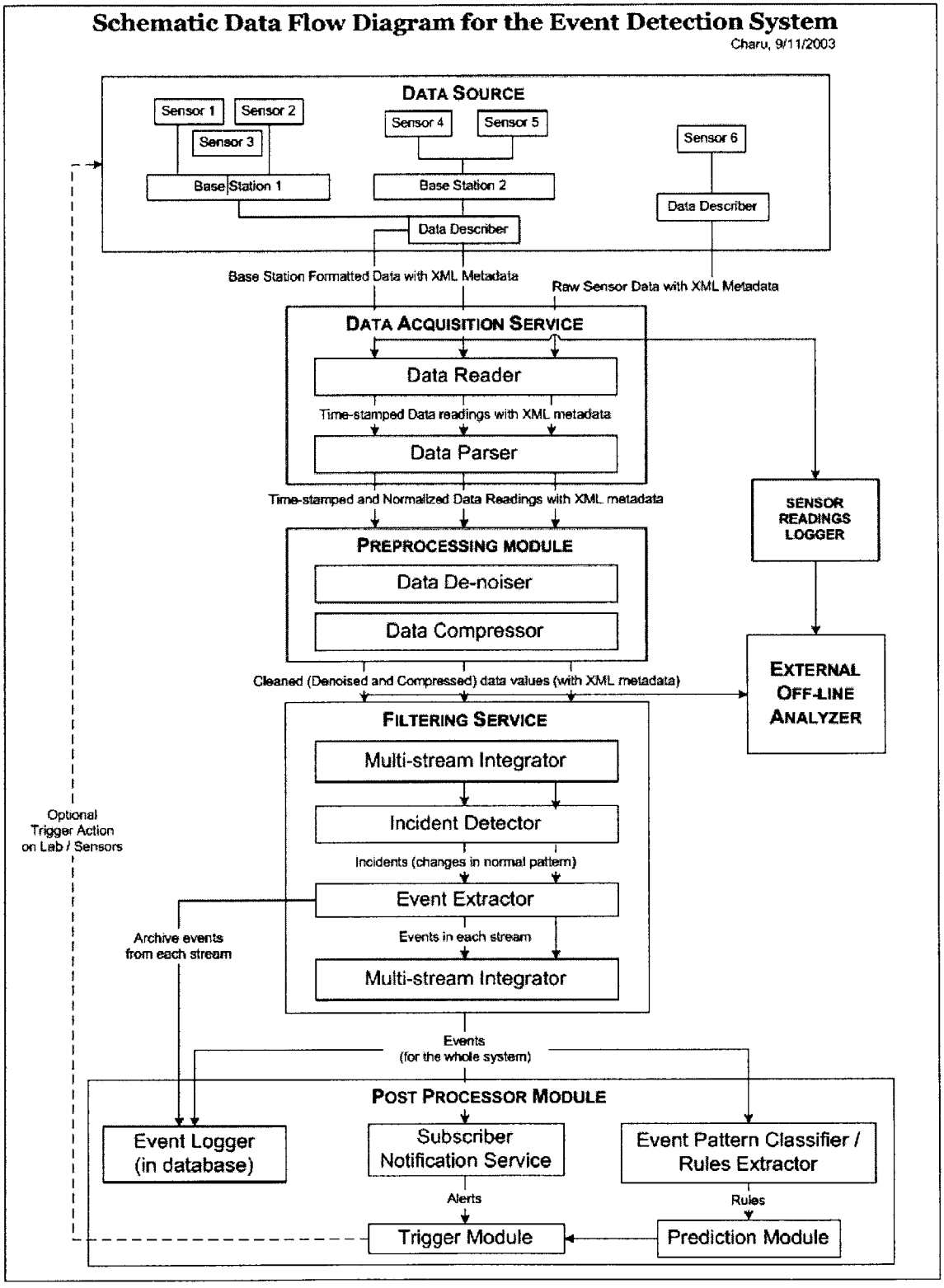


Fig. (16) Software Architecture of the Event Detection System

trigger action is when a device shuts down upon detecting unusual occurrences in the values. Analog output from the trigger module through actuators can cause hardware changes to the environment.

3.4.4.3 Event prediction based on patterns

The prediction module could identify future occurrences of events, based on the past values, in two possible ways:

1. To predict when a particular event will next occur
2. To predict what might occur at a particular time

An Event Pattern classifier, which identifies patterns in the sequence of events generated by the filtering service, feeds into the prediction module. These patterns could either be used directly for prediction, or as rules to determine what happens next.

Fig. 16 shows how all the components of the architecture tie in with each other.

3.5 LAYERED API STRUCTURE

Having a generic set of interfaces between components of the architecture allow them to talk to each other effortlessly. The original idea behind these interfaces comes from the Open Knowledge Initiative (OKI) at MIT [14]. OKI defines a set of fundamental services that focus on the modularity and interoperability of components in an educational environment, which would allow them to work well with each other, and with other enterprise applications.

Fig. 17 shows an initial version of the layered API structure that was developed for this event-detection system. The initial components communicate with each other using a Generic Sensor Reading API (a preliminary version of this API is attached in Appendix 3). OKI's DBC and SQL API's would then interface between the event-filter and the archive, while the OKI Event Notification API could layer between the event-filter and the notification service. Thus by specifying these interfaces, a generic protocol of communication can be established between components that do not have any previous knowledge about each other.

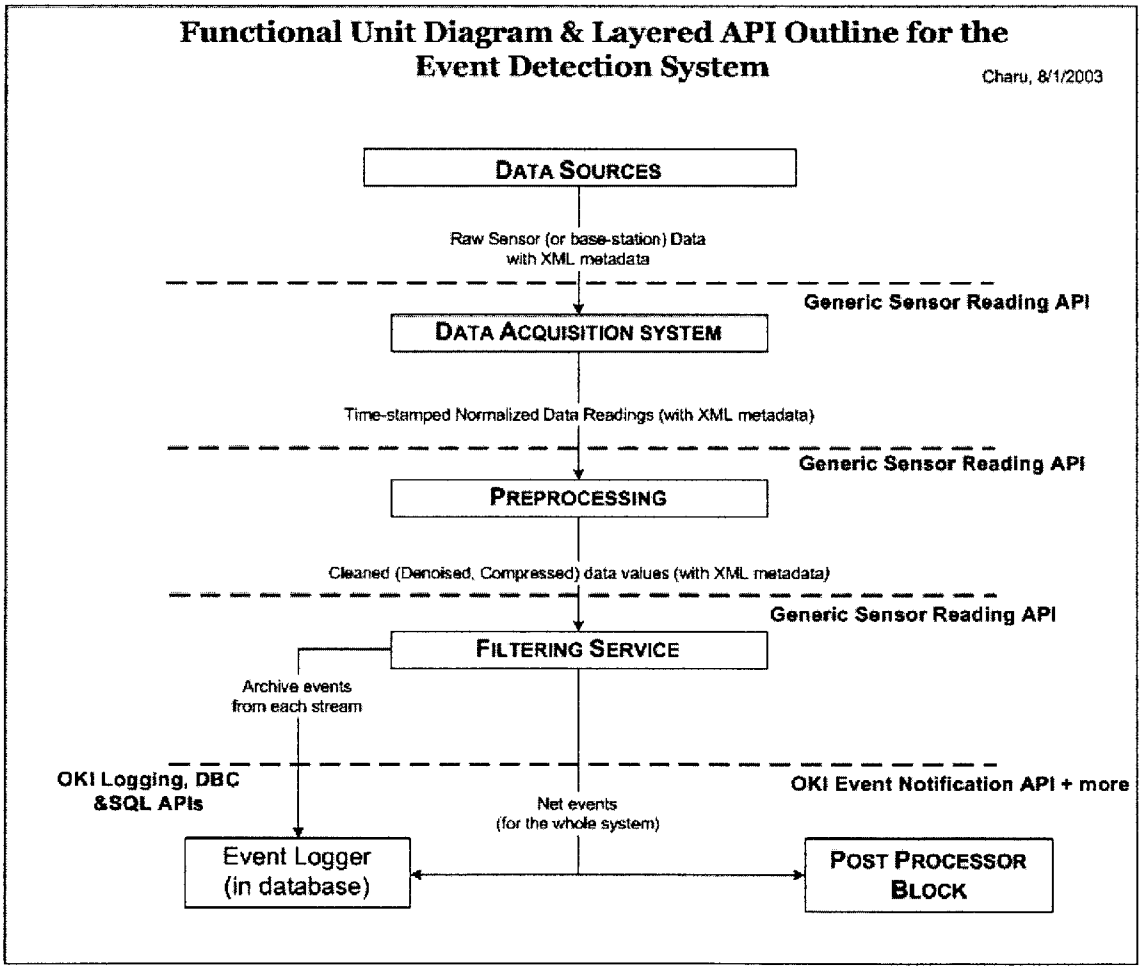


Fig. (17) Layered API Structure for the Event Detection System

3.6 COMPARISON WITH OTHER ARCHITECTURES

While no generic event-detection architecture was found in the literature survey, a few architectures had been defined for particular domains. The event-detection architecture developed in this project compared well with these domain architectures, matching them in functionality.

One of the architectures that was surveyed is the EMERALD network intrusion detection system [15], one of the pioneer works in the area of event monitoring in large networks. EMERALD takes a distributed and layered approach to network surveillance and consists of different types of engines that identify malicious attacks through different technique (statistical anomaly detection and signature-based detection). It also contains several third party modules such as an event logger, analysis engine and a results processor. A monitor API allows for the interoperability between the components. Though EMERALD may not match the architecture that was developed for this project component by component, it has the same functionalities that the latter possesses.

CHAPTER 4 – ALGORITHM FOR THE FILTERING SERVICE

This chapter outlines the wavelet-based algorithm for detecting irregularities that was developed for this project, as well as the reasons behind choosing the wavelet approach. This algorithm is a generic method for detecting events in real-time, independent of the domain, and assumes no prior knowledge of the data. If the users have information about the domain or data in advance, then they could choose to design a more efficient algorithm, and integrate it with the architecture. The algorithm is limited to work with one-dimensional numeric data only.

4.1 REQUIREMENTS ANALYSIS AND DESIGN DECISIONS

The filtering algorithm should have the following requirements:

1. It should not assume any prior knowledge of the data.
2. The input data is non-stationary, i.e. the model of the data or the parameters of the model could change with time.
3. It should have very low incremental computational complexity, so that events can be identified in a continuous and timely fashion.
4. The algorithm should be as accurate as possible. This is a domain-dependent requirement, which specifies how many false alarms vs. misses the algorithm can tolerate. In some cases, such as ECG monitoring, misses are very expensive, while false alarms could be tolerated. In domains such as cell phone fraud detection, generating false alarms would be very inconvenient for the subscribers.
5. It should be able to adaptively detect the trends and surprises in the data set at multiple levels of abstraction. For example, the events that are computed will vary

according to whether the time series is analyzed on a monthly basis or on a daily basis, even though the underlying data remains the same.

6. It should be able to distinguish outliers (highly abnormal events that should raise alerts) from trends (normal changes in the data sequence)

It is assumed that the algorithm will fit into the architecture defined in Chapter 3. Hence it is enough if the algorithm analyzes the streams independently, since the Multi-Stream Integrators in the Filtering Service take care of the correlation between streams.

4.2 POSSIBLE APPROACHES TO FILTERING EVENTS

The following sections discuss the four possible approaches to filtering events.

4.2.1 Heuristics and Statistical Techniques

These methods involve fitting the data with some probability model, or measuring some statistics of chosen features of the data, based on prior knowledge. Outliers or trends are then calculated based on deviations of the signal from the probability model or baseline statistics.

Network intrusion detectors use these methods in anomaly detection solutions. For example Thottan and Ji [16] use a Generalized Likelihood Ratio test to detect the change points in the behaviour of variables that represent the network traffic across various protocol layers. Abnormal changes are then detected by thresholding statistical parameters such as the mean and variance of the data. Guralnik et al. [17] developed a

generalized algorithm that detects change-points by iteratively curve-fitting the data based on statistical likelihood criteria.

The problem with these type of methods is that they usually assume the signals to be stationary or piecewise stationary, which cannot be done in the case of random sensor data. Furthermore, they require domain knowledge to determine the statistical parameters that need to be used in calculations, which again violates the requirements of a generic detector.

4.2.2 Pattern Recognition Techniques

A variety of pattern recognition techniques such as neural networks and clustering have been used to identify changes and irregularities in the signal. For example Suzuki [18] uses adaptive resonance theory based neural networks in a QRS-wave recognition system that can be used to identify cardiac arrhythmia; Coast et al [22] use Hidden Markov Models to classify ECG beats for arrhythmia detection. Yang et al [19] use text retrieval methods and clustering to detect unusual events occurring in news stories that are ordered sequentially in a time-series, both offline and online. Fawcett and Provost [12] use adaptive learning in the cell phone fraud detection domain to determine a set of rules that can be used to profile fraudulent calls. Duda, Hart and Stork [33] discuss a variety of techniques that can be used for pattern classification and analysis.

However, most pattern recognition techniques require some knowledge of the domain mainly for the purpose of feature selection and classifier training. While there are

techniques that use unsupervised learning, such as clustering [33], these become hard to implement in real-time and also need some domain information (for example to determine the number of clusters). On the other hand, if some prior information or data is available to the user, some of these techniques may be more accurate or quicker than the one ultimately chosen in this project.

4.2.3 Adaptive Filtering Algorithms

Adaptive filtering is the classic, generic approach for detecting events in unknown, non-stationary data. Gustafsson [20] describes several stochastic techniques, such as Least Mean Squares (LMS) and gradient minimization for detecting outliers and changes, and their applications in various domains. Some of these filters, like the Kalman filter, can also be used in prediction. [20]

The problem with these techniques is that they are mostly based on the Fourier transforms, which cannot exactly pinpoint the location of an event since the signal is converted to the frequency domain. Hence these techniques need to use moving windows that slide along the data, varying the width of which gives different analysis resolutions. However, the generality of the approach is partially lost by using a moving window approach, since the length of the window depends on the events that need to be detected. [26] Furthermore, not only do these windowing approaches introduce artificial edges into the signal but they also usually assume that the signal is stationary within the length of the window.

4.2.4 Wavelets

Wavelets can be used to determine irregularities in a signal, without domain knowledge. They overcome most of the problems faced in adaptive filtering, and are computationally more efficient than Fourier transforms. Also, they can be used for both de-noising and data compression in the pre-processor stage. Section 4.3 gives a brief introduction to wavelets and outlines the advantages of using them, based on the requirements. Section 4.4 describes how wavelet-based irregularity detection works.

4.2.5 Combining these techniques

A lot of the literature surveyed uses a combination of these techniques very effectively. For example, Adeli and Karim [21] use a wavelet preprocessor to de-noise traffic data, followed by fuzzy clustering for dimensionality reduction and for extracting significant information from the de-noised signal. Finally this processed data is fed into a radial basis function neural network for classification to detect incidents in the traffic data. KrishnaPrasad and Sahambi [23] use the discrete wavelet transform to extract the portions of an ECG signal that contain maximum information about an arrhythmia and then input the wavelet coefficients into a neural network for classification.

The algorithm outlined in this thesis uses wavelets to identify irregularities, followed by an adaptive threshold that is learnt from the data, which separates out the global irregularities, in the context of the analysis resolution.

4.3 WHAT ARE WAVELETS?

4.3.1 About Wavelets

Wavelets are small waves (pulses), which oscillate in a finite time interval in such a manner that the net integral over time is zero [25]. Fig. 18 shows a Morlet wavelet, which is symmetric, and oscillates from t=-4 to t=4. These small waves, which are called mother wavelets, are basis functions in continuous time. [27]

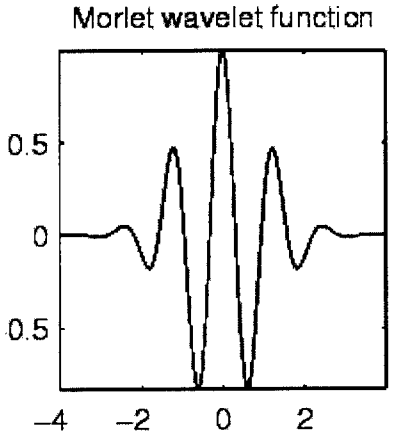


Fig. (18) A wavelet example

A basis is a set of linearly independent functions that can be used to produce all admissible functions $f(t)$. Hence,

$$f(t) = \text{combination of basis functions} = \Sigma\, d_{jk}\, w_{jk}(t).$$

where $w_{jk}(t)$ are the wavelet functions and d_{jk} are the wavelet coefficients. The wavelet functions $w_{jk}(t)$ can be constructed by scaling and shifting the mother wavelets.

$$w_{jk}(t) = w(\frac{t-k}{j})$$

where, w is the mother wavelet that is scaled j times and shifted k times [24]

The wavelet transform converts signals from the time domain to a time-scale domain (where scale can be considered to vary inversely with frequency). [27] The continuous wavelet transform is analogous to a windowed Fourier transform, where an inner product is done with wavelets instead of windowed exponentials.

$$Wf(t,j) = \frac{1}{\sqrt{j}} \int f(t) * w\left(\frac{t-k}{j}\right)$$

Here, the length of the "window" varies according the scale, which allows for the multi-resolution analysis of the signal. Changing the choice of the mother wavelet *w(t)* produces different types of wavelet transforms that can be used in a variety of applications. [24]

The Discrete Wavelet Transform (which is used when the data is not continuous but is sampled at discrete intervals, like in sensor signals), uses discrete values of the scale (in powers of 2), instead of a continuous scale. [25] Hence, the wavelets $w_{jk}(t)$ are then derived from the mother wavelet w(t) as,

$$w_{jk}(t) = w(2^j t - k)$$

where j is the scale and k is the shift.

The Discrete Wavelet Transform (DWT) can then be calculated as [24]

$$Wf(j,k) = 2^{j/2} \int f(t) w(2^j t - k)$$

Multi-resolution analysis can be achieved by iterating the signals over filter banks (over the low-pass or high-pass channels), where the filter coefficients depend on the wavelet chosen. In a filter bank the low-pass channel results in the smoothing of the signal, while

the high-pass channel gives the irregularities (high frequency content) in the signal. [27] Strang and Nguyen [24] explain wavelets, multi-resolution analysis, and filterbanks in detail in their book.

4.3.2 Why do they work here?

Key wavelet properties that make it suitable for the purpose of irregularity detection are

1. Wavelets are localized both in time and frequency, which allows them to pinpoint the location of irregularities without a moving window, unlike the Fourier transform. Hence the wavelet transform is ideally suited for processing non-stationary data. [26]
2. Wavelets have the multi-resolution property that allows users to look at different abstractions of the same data (for example data analysis based on a year or a month), according to the scale of analysis. [24]
3. Wavelets are computationally efficient and can provide timely results. For example, given a time sequence of length n where n is an integral power of 2, the complexity of the Haar transform is O(n), whereas the fast Fourier Transform is O(n log n). [29]
4. Wavelets provide an efficient representation of data; they code transient phenomena such as edges very efficiently, localizing them to a few coefficients. [27]
5. Wavelets can, to a certain extent, also be used to identify the type of irregularities discovered. [5] Section 4.4 describes how wavelets are used to calculate the Lipschitz exponents of data points, which are an indicator of the irregularity measure of a point.
6. Wavelets can be used in the preprocessing stage for de-noising and compression. [26]

4.4 CHARACTERIZATION OF LOCAL IRREGULARITIES WITH WAVELETS

4.4.1 Relation between the wavelet transform and Lipschitz exponents

While the Fourier Transform gives a global measure of the irregularity in the signal, it cannot exactly pin-point the locations of the irregularities. Wavelet transforms are hence well adapted for irregularity detection because they convert the signal into a time-scale domain, which allows us to identify when irregularities occur. [5]

It can be proved that the asymptotic decay of the wavelet transform at small scales is related to the local Lipschitz regularity of $f(x)$ [5]. It turns out that performing a wavelet transform over a function is equivalent to passing it through a smoothing filter and differentiating it. For example, if the smoothing function is a Gaussian $g(x)$, the result of passing $f(x)$ through the smoothing filter is $f(x)*g(x)$. Differentiating this, we get

$$\frac{d(f*g)}{dx}(x) = f(x)*\frac{dg(x)}{dx} = f(x)*w(x) = Wf(x,s)$$

where $w(x)$ is a wavelet and $Wf(s,x)$ is the wavelet transform of $f(x)$ at particular scale s of the Gaussian. The result of smoothing edges (or sharp variation points) and differentiating in the normal case, is to get peaks which can be detected as the local maxima of the absolute value of the first derivative (the local modulus minima give slow changes). Hence by identifying the local maxima of $|Wf(s,x)|$, i.e. the modulus of the wavelet coefficients, the irregularities in $f(x)$ can be identified.

To identify the degree of sharpness of an irregularity's edges, the wavelet modulus maxima can be used to measure the Lipschitz exponents of a function. Mallat et al, [5] prove a theorem which states that a function is uniformly Lipschitz α, α<n in a given interval, if the wavelet transform has no modulus maximum at small scales in that interval. An extension of this theorem would be to say that the function is not singular in an interval where its wavelet transform has no modulus maxima at small scales.

Mallat et al, also prove that *f(x)* is uniformly Lipschitz α at x_0 iff there exists a constant A such that

$$|Wf(s,x)| \leq As^{\alpha}$$

where s is the scale and $|Wf(s,x)|$ lies in in a cone defined by $|x-x_0| \leq Cs$ (C is a constant)

This is equivalent to

$$\log|Wf(s,x)| \leq \log(A) + \alpha \log(s)$$

which means that the Lipschitz regularity α is the slope of straight lines that remain above $\log|Wf(s,x)|$ on a log scale, and hence can be found by calculating the decay of $|Wf(s,x)|$ over a range of scales (In the discrete wavelet transform the smallest scale is 1).

4.4.2 Choosing the right Wavelet

It is important that the wavelet selected for analysis is able to identify all the irregularities that a user might be interested in, which means that it should be able to estimate the Lipschitz exponents α's that characterize these irregularities to a maximum value of n.

This maximum value n depends on the signal. However to identify irregularities of Lipschitz n, the chosen wavelet must have at least n vanishing moments. [5] Vanishing moments are a property of wavelets that indicates their differentiability. The more the number of vanishing moments in a wavelet, the more differentiable they are. A wavelet has p vanishing moments if *all* moments up to and including the (p-1) th moment are zero. Mathematically, for a wavelet with p vanishing moments [24]

$$\int_{-\infty}^{\infty} x^{p-1} w(x) dx = 0$$

where p is an integer.

However, if the chosen wavelet has too many vanishing moments, then a lot of local maxima are detected (since the number of maxima usually increases linearly with the number of vanishing moments). Hence the choice of the wavelet must be optimized such that it has as few vanishing moments as possible, but just enough to capture the irregularities the user wants. [5] For the purposes of this research, it was assumed that n=3 captures most of the irregularities in the test signals (based on the literature survey of signals in the domains studied).

4.5 A REAL-TIME ALGORITHM FOR GLOBAL IRREGULARITY DETECTION USING WAVELETS

4.5.1 Local real-time irregularity detection

The algorithm followed in this project, uses Mallat's [5] technique for irregularity detection. In standard wavelet 1-D edge detection theory, a noisy signal is passed repeatedly through a low pass channel, and the points where the peaks reinforce each

other across different subbands are identified as irregularities (based on the theory presented in Section 4.4). Fig. 19 illustrates an example, where a noisy step signal is passed through four levels of wavelet decomposition. At the first level, the wavelet coefficients are very noisy and the peaks can barely be made out. However, with further decomposition, the peaks in the different levels only stand out at the points where there are edges in the signal.

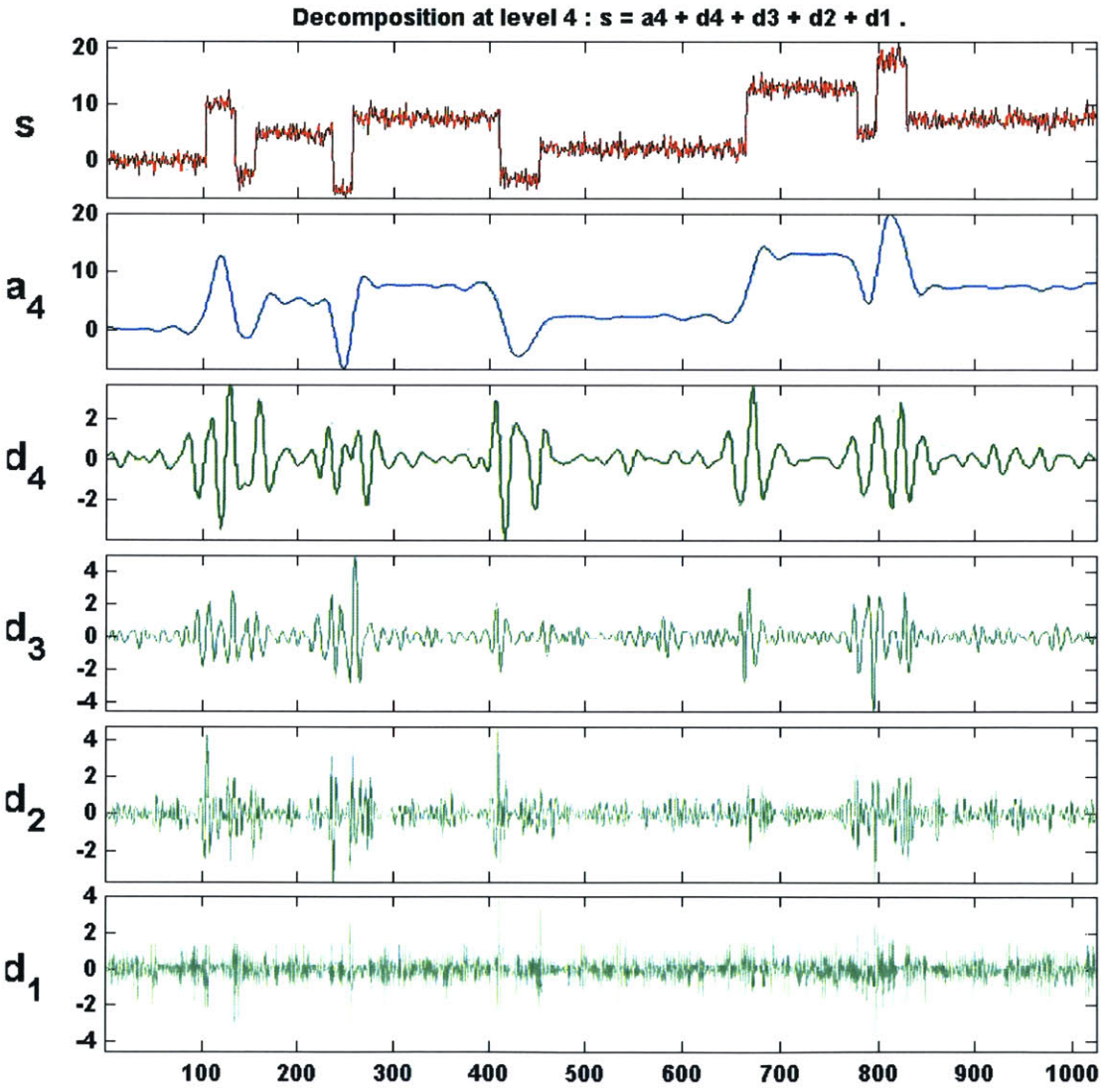


Fig. (19) Multi-Resolution Analysis of a noisy step signal

The procedure followed in real-time is very similar. First an appropriate wavelet must be chosen, according to the number of its vanishing moments. This does not depend all that much on prior knowledge of the data, since the number of vanishing moments depends more on the type of irregularities that need to be detected, and most of the wavelets that have the same number of vanishing moments, give more or less the same irregularities (observed from the implementation). Then the number of levels of decomposition must be chosen based on the resolution at which the user wishes to analyze the data. The more the levels of decomposition, the coarser the resolution, since the irregularities are smoothed out with each iteration. The high frequencies that appear in a lower resolution are treated as noise at higher resolutions, and hence not considered after the thresholding operation described in section 4.5.2. The algorithm then followed for real-time irregularity detection is as follows

For scale j = 1 to max_decomposition_level

- **Take a sliding buffer of data (based on filter length)**
- **Compute the DWT coefficients d_j's at level j**
- **Find the local modulus maxima of d_j's**
- **Compute a global maxima threshold based on scale j (discussed in Section 4.5.2)**
- **If the local modulus maxima at a point > global threshold then mark point as irregularity**

4.5.2 Using a global threshold

Mallat's methodology works while identifying irregularities offline, when the entire signal is available. However for real-time analysis, a threshold that separates out the global irregularities from the local ones at a particular scale of analysis was required.

Since the wavelet coefficients are treated as a noisy signal, while identifying the peaks, a standard denoising threshold can be applied, which is calculated based on the local modulus maxima of the wavelet coefficients. If the modulus maxima of the buffer consisting of the new data points exceed this threshold, then the points are taken as irregularities. The value of the threshold is then recomputed with the new values of the data.

4.6 IMPLEMENTATION DETAILS

The implementation of the wavelet algorithm was done in Matlab 6.0, which with its inbuilt wavelet analysis features, seemed to be the best environment to implement a prototype in. The data used for testing, were mostly Matlab examples of irregularities (or combinations of them), since comparisons could be made between an offline analysis and the real-time algorithm.

The wavelet used in the analysis was Daubechies 6, since it has 3 vanishing moments [28], which turns out to be good enough for identifying the types of irregularities that are present in the Matlab examples. The global threshold that was used in this was a standard noise detection threshold using the principle of Stein's Unbiased Risk Estimate (SURE) [30], where the noisy signal is considered to be detail coefficients obtained from the wavelet decomposition. Hence, while small variations in the detail coefficients are eliminated as noise, only the really significant changes are kept back after denoising.

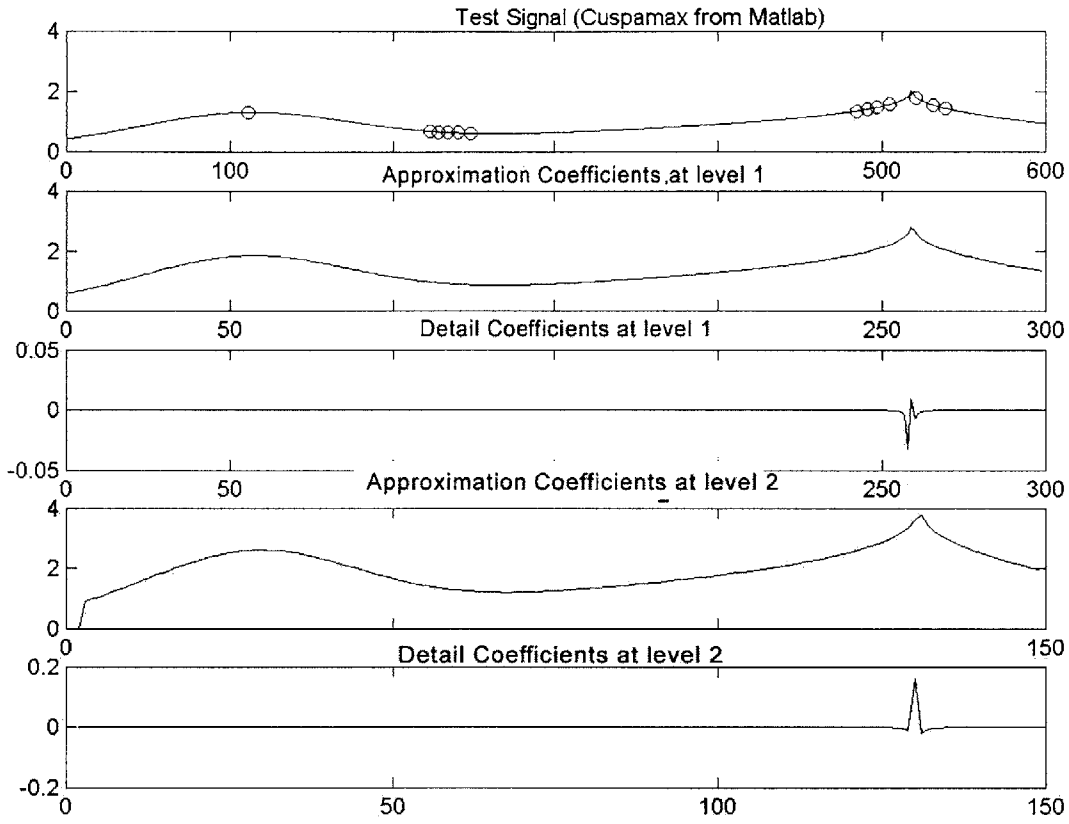


Fig. (20) Sample implementation

Fig. 20 is an example of a Matlab signal (cuspamax) run through the algorithm, analyzed to two levels of detail. The irregularities detected are marked by the circles. Even though the initial circles are not points of irregularity, the threshold computed initially identifies them so. This however does not happen for similar points in the future, once higher values of the threshold are computed.

CHAPTER 5 – CONCLUSION

5.1 SUMMARY

Detecting events in sensor data, and alerting users when they occur is a very crucial aspect of monitoring in any kind of environment. Though the idea for this research sprung mainly from the iLab project at MIT, this work can be used in non-laboratory applications such as ECG monitoring and network intrusion detection.

The research done in this project first involved developing a formal event theory that could be applied generically across several domains. Definitions of "events", "incidents", "irregularities" and "edges" were specified as a means of identifying the events ultimately required by users. Next, a flexible software architecture was defined for online real-time event detection, which users could either use directly (with the inbuilt filtering service), or easily integrate their custom components into. The architecture also provided the hooks for pre-processing the data (for example, denoising it), data archival, user notification and triggers, and for post-filtering pattern analysis. Finally a partial implementation of the filtering service was done, based on wavelets, which turned out to be the most ideal tool for analyzing random, unknown sensor signals for irregularities. Not only does the implementation work in real-time, but is also capable of supporting multiple levels (based on time-periods) of analysis, as required by the user, which was not possible in previous techniques for real-time event detection.

5.2 LIMITATIONS

The system is mainly limited by the generality required of it. It is very hard, and probably very inefficient, to develop a wholly general mechanism to detect events in any kind of domain. The following limitations are deficiencies of the approach and not of the implementation described in Chapter 4, since that is highly scaled down, and was done mainly to illustrate the concepts described in this thesis.

5.2.1 Limitations of event theory

The definitions specified in Chapter 2 only work for abnormal events, or events that represent sharp changes in the behaviour of the data. They are not applicable for events that represent smooth trend changes or no change at all. An example of the former situation is when a drift in the sensor values occurs. The deviations from the baseline occur very slowly over time, but are considered to be a significant event over a period of time. An example of the latter case is a flip flop that is used in a random number generator circuit. Here, the lack of change of data in the output of the flip flop would be considered as an event of interest. However, since most applications usually seek either outliers or sharp trend changes, the work in this research was mostly geared towards detecting those types of events.

5.2.2 Limitations of software architecture

While the architecture is flexible in allowing users to define their custom components, it is not possible to wholly implement the architecture without some domain knowledge. The main block of the filtering service is heavily dependent on the domain; both the

Multi-Stream Integrators and the event extractor need to be specified by the users. Again, this is a problem that arises because of the varying requirements of different users and domains. Also, while a block in the Postprocessor Module allows for triggers to be installed and activated, these require a very different type of architecture, based on agents that can take actions on the system.

The scope of the architecture is also limited in that it does not tackle any of the security and authorization issues. Finally, the system was intentionally designed without any user interaction. Hence, neither does the algorithm permit any user queries on the data, nor does the architecture incorporate user feedback. However, since the application itself is so user-driven, adapting the filtering algorithm to user feedback would've been a nice feature.

5.2.3 Limitations of wavelet algorithm

The wavelet algorithm in this project was designed to work with one dimensional numeric time-series data only. The area of detecting events and edges in 2-D image data overlaps with the goals of machine vision applications, which is a huge field in itself. The current version of the algorithm uses a predefined wavelet and maximum decomposition level (that can be user-specified). Also, the algorithm only iterates over the low pass filter currently, while some applications may require iterating over the high pass filter to capture the high-frequency components in the signals.

5.3 FUTURE WORK

5.3.1 Expanded domain implementation

Currently only a small portion of the filtering service was implemented for this project. A more complete implementation however, necessarily needs to be domain-specific. An expanded implementation would include creating a data archive to log events, performing user notification and predicting events through pattern analysis. It would be useful to develop a Microsoft .NET version of this implementation which could leverage web-services to perform these tasks easily across platforms.

The filtering algorithm can also be modified to be optimized for a domain, once it is chosen. Especially, if prior data is available, supervised learning techniques, where this data can be used to train the filtering service, will prove to be much more efficient than a generalized wavelet based algorithm.

5.3.2 Multi-Agent based architecture

Trigger actions can be carried out through "agents", which are program modules that can act on the environment. A multi-agent based architecture will allow the system to execute domain-specific actions efficiently upon detection of relevant events. The advantage of using an agent based system is that these agents are designed to be adaptive, and hence they learn to take the right actions based on their experiences and inbuilt knowledge. This is especially useful when the action that needs to be taken is not as clearly defined as a "turn on/off the switch" type of action, but is more goal-based, like "to stabilize the network, when a ping attack has been detected". Further more these agents usually work

independent of each other, and hence, can be added or removed from the system (according to the actions that need to be performed), without affecting the rest of the components. These agents can also be configured to work with actuators and actually perform hardware changes on the environment.

5.3.3 Improving the wavelet algorithm

The wavelet function chosen in the algorithm is pre-defined, whereas a truly adaptive method should be able to determine the appropriate wavelet that can analyze the data best. While this is hard to do in a completely general case, usually some information is known about the events that need to be detected. Thus an iterative procedure can be used to find out the wavelet that suits the data best.

Furthermore, the current algorithm only iterates over the low pass filter, whereas ideally, the best basis, over a combination of low and high pass filters can be determined, if information about the events to be detected is known. Finally, the method used for calculating the cut-off threshold in the algorithm is predefined. However, different time intervals might require different techniques for computing the threshold. For example, in traffic data, the thresholds that need to be computed for nights and days will vary hugely and hence a global threshold cannot be used. In such cases, interval thresholding, analogous to that used in denoising, can be used to overcome this problem.

APPENDIX 1 – EVENTS IN SOME DOMAINS

Field	Data	Feature Observed (vs. time)	Event (defined as)	Event (identified by)
Financial Data tracking	Multi-channel data (quotes from companies)	Stock prices	Sharp variation in price	Difference in prices over time exceeding preset user-defined thresholds
SETI [31]	Single channel output from a telescope broken down into multi-channels and distributed	Radio signal frequency	Detection of signals that suggest possible occurrence of extra-terrestrial life	Detection of narrow bandwidth signals whose Gaussians have a chi square fit less than 10, and power greater than 10.2 (in 12 sec)
Identifying new stories in continuous news streams [19]	Single channel data of news stories, with no prior knowledge of novel events. However, unlabelled news stories are available for contrast	Document text	Occurrence of a new story	Significant vocabulary shift and rapid changes in term frequency distribution
Sleep monitoring system [32]	Multi-channel sensor data	Heart rate, breathing rate, body orientation and restlessness index	Sleep disorder	Sudden change in restlessness index, heart rate or breathing rate

APPENDIX 2 – SAMPLE XML METADATA FILE OF SENSOR STREAMS

```
<?xml version='1.0' standalone='yes' ?>
<iLabData>
    <Version>1.0</Version>
    <Equipment>
        <Source>ThermSensor 1</Source>
        <SourceType>Thermometer</SourceType>
        <SamplingRateUnit>milliseconds</SamplingRateUnit>
        <SamplingRateValue>100</SamplingRateValue>
        <DataChannel> 1B </DataChannel>
    </Equipment>
    <Measurement>
        <MeasurementName>Temperature</MeasurementName>
        <MeasurementUnit>Celsius</MeasurementUnit>
    </Measurement>
    <Data>
        <BytesPerPacket>33</BytesPerPacket>
        <DataReadings>10</DataReadings>
        <DataReadingSize>2</DataReadingSize>
        <DataReadingUnit>Bytes</DataReadingUnit>
    </Data>
</iLabData>
```

APPENDIX 3 – GENERIC SENSOR READING API

Author: Charu, 8/5/2003

INTRODUCTION

The methods in this API define the functionality for reading data from a data source. It is assumed here that the data acquisition system is pulling data from the data source. The reason why pulling is done at this stage is to ensure that most of the sensor/base-station data gets to the iLab system, and hence can be archived.

The SensorReading object must contain a sensor id, time-stamp, data values and any other additional data that the lab may wish to use.

CALLS FROM DATA ACQUISTION SYSTEM (DAS) TO DATA SOURCE

GetSourceConfig:

Purpose: gets information from the source about how it is configured (for example, the number of channels / data streams in a sensor or the number of sensors connected to the base-station)

Parameters: int sourceID

//ID of sensor or basestation to which the connection has to be made

Returns: Object sourceConfig

//configuration of the source

CheckSignalStatus:

Purpose: checks if the data acquisition system can feel the presence of a data source. This should be done regularly (as required by the experiment) to check if the source is working.

Parameters: int threshold

// describes the minimum signal level required for connection to be established

int sourceID

// gives the ID of the sensor or base station

Returns: bool isSignalAlive

// yes if it is ok to make a connection / signal working

int signalLevel

// describes signal strength on a scale of 1-10

EstablishSourceConnection:

Purpose: creates a connection from the DAS to the source (using a pre-defined protocol).

Parameters: int sourceID

//ID of sensor or base station to which the connection has to be made

int connectionTimeout

// gives the time in ms until which the base station will try to connect with the sensor

Returns: bool isConnectionSuccessful

// yes if connected

Disconnect:

Purpose: drops the connection between the DAS and specified source

Parameters: int sourceID

//ID of sensor or base station whose connection with DAS has to be dropped

Returns: bool isDisconnectSuccessful

//yes if successfully disconnected

GetDataFormat:

Purpose: reads the XML metadata, which describes the format of data that is being transmitted from the source into the DAS. This should be called every time a new connection with the source is established.

Parameters: int sourceID

Returns: Object sensorReading

// the actual readings from the sensor – can be in any format, for example, an arraylist containing time-stamps and data values

ReadData:

Purpose: reads the data from the source into the DAS. The DAS polls the source for readings in regular intervals. (If the data needs to be pushed, then this method should not be called)

Parameters: int sourceID

int frequency (or time period)

// the number of times per second the DAS should poll the source

Returns: Object sensorReading

// the actual readings from the sensor – can be in any format, for example, an arraylist containing time-stamps and data values

CheckPacketDropped:

Purpose: To check if any packet has been dropped since the last reading

Parameters: static int prevCounterValue (static int)

Returns: int packetsDropped

// no. of packets that were dropped in the transmission

Calls from Source to DAS

PushDataReadings:

Purpose: This is an optional call that should be used if the data needs to be pushed from the source to the DAS.

Parameters: int sourceID

Object sensorReading

// the actual readings from the sensor – this can be in any format, (for example, an arraylist containing time stamps and data values)

Returns: bool isPushSuccessful

// yes if data has been transferred successfully to DAS

APPENDIX 4 - REFERENCES

[1] "StanfordStreamDataManager" website: http://www-db.stanford.edu/stream/

[2] "iCampus Student Proposals", Microsoft iCampus Website, http://swiss.csail.mit.edu/projects/icampus/

[3] Flagpole at MIT : http://flagpole.mit.edu/

[4] iLab Project White papers, CECI, MIT

[5] Stephane Mallat and Wen Liang Hwang, "Singularity Detection and Processing with Wavelets", *IEEE Transactions on Information Theory*, Vol. 38, pp. 617-643, March 1992.

[6] United States Geological Survey – Earthquake Hazards website: http://quake.usgs.gov/

[7] "National Oceanic and Atmospheric Administration" website: http://w3.noaa.gov/stormwatch/

[8] Andrew C. Revkin. Ozone Layer is Improving, According to Monitors. New York Times 2003 July 30.

[9] Hojjat Adeli, "Automatic detection of traffic incidents using data from sensors embedded in intelligent freeways", *Sensor Review,* Vol. 22, Number 2, pp. 145-149, 2002

[10] "Network Intrusion Detection FAQ" website: http://www.robertgraham.com/pubs/network-intrusion-detection.html#1.1

[11] http://www.medscape.com

[12] T. Fawcett and F. Provost. (1997),"Adaptive fraud detection", *Data Mining and Knowledge Discovery*, Vol. 1, Number 3, pp. 291-316, 1997

[13] "MathWorld" website : http://mathworld.wolfram.com/Cusp.html

[14] "Open Knowledge Initiative": http://web.mit.edu/oki/

[15] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," *Proc. 20th National Information Systems Security Conf.*, Baltimore, MD, October 7-10, 1997, pp. 353-365.

[16] Thottan, Marina and Ji Chuanyi, "Adaptive thresholding for proactive network problem detection", *Third IEEE International Workshop on Systems Management*, pp. 108-116, Newport, Rhode Island, April 1998

[17] V. Guralnik, J. Srivastava, "Event Detection from Time Series Data", *International Conference on Knowledge Discovery and Data Mining*, pp.33-42, San Diego, California 1999

[18] Suzuki, Yukinori, "Self-Organizing QRS-Wave Recognition in ECG using Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 6, No. 6, November 1995

[19] Y. Yang, T. Pierce, J.Carbonell, "A study of retrospective and on-line event detection", *Proceedings of the 21st annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 28-36, Melbourne, Australia, August 1998

[20] Fredrik Gustafsson., *Adaptive Filtering and Change Detection*, John Wiley & Sons Ltd., 2000

[21] Adeli, Hojjat and Karim, Asim, "Fuzzy-Wavelet RBFNN Model for Freeway Incident Detection", *Journal of Transportation Engineering*, pp. 464–471, November-December, 2000

[22] D. A. Coast, R. M. Stern, G. G. Cano, and S. A. Briller, "An approach to cardiac arrhythmia analysis using hidden markov models," *IEEE Trans. Biomedical Engineering*, Vol. 37, pp. 826–836, Sep 1990.

[23] G. Krishna Prasad and J.S. Sahambi, "Classification of ECG Arrythmias using Multi-Resolution Analysis and Neural Networks", *Proceedings of IEEE TENCON*, Bangalore, India, October 2003

[24] Gilbert Strang and Truong Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1997

[25] "Amara's Wavelet Page": http://www.amara.com/current/wavelet.html

[26] Amara Graps, "An Introduction to Wavelets", *IEEE Computational Sciences and Engineering,* Volume 2, Number 2, Summer 1995, pp 50-61

[27] The Mathworks : Wavelet Toolbox Overview

[28] I.Daubechies, *Ten Lectures on Wavelets,* CBMS – NFS Series in Applied Math, vol. 61, SIAM, Philadelphia, 1992

[29] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, AddisonWesley, 1992.

[30] Stein, C.M., "Estimation of the mean of a multivariate normal distribution", *Annals of Statistics* 9: 1135-1151, 1981.

[31] "SETI@home: Search for Extraterrestrial Intelligence at home" website : http://setiathome.ssl.berkeley.edu/

[32] Van der Loos HFM, Kobayashi H, Liu G, Tai YY, Ford JS, Norman J, Tabata T, Osada T., ":Unobtrusive vital signs monitoring from a multisensor bed sheet", *Proc RESNA Ann Conf*, Reno, Nevada, pp. 218-220, 2001.

[33] Richard O. Duda, Peter E. Hart, and David G. Stork," *Pattern Classification",* John Wiley and Sons, Inc., New York, 2001