INTERPRETOR: A Software Architecture for the Interpretation of Large and Noisy Data Sets

APKAR SALATIAN

School of Information Technology and Communications
American University of Nigeria
Lamido Zubairu Way, Yola By-Pass
PMB 2250, Adamawa State, Nigeria
apkar.salatian@aun.edu.ng

Abstract. In many domains there is a need to interpret the high volumes of noisy data. In this paper we propose and describe a new software architecture called INTERPRETOR for summarising and interpreting voluminous high frequency noisy data. INTERPRETOR consists of 3 modules: Filter which processes noise; Abstraction which abstracts features from the filtered data; and Interpretation which takes the abstractions and provides an interpretation of the data. In this seminal article we also show how INTERPRETOR has successfully been applied to 2 case studies.

Keywords: oftware architecture, filtering, abstraction, interpretation.

(Received February 11st, 2011 / Accepted May 2nd, 2011)

1 Introduction

In many domains there is a need to interpret high frequency noisy data. Interpretation of such data may typically involve pre-processing of the data to remove outliers, inconsistencies or noise. Rather than reasoning quantitatively on a point to point basis which is computationally expensive, this filtered data would be processed to derive abstractions which would be interpreted and the results reported. Such a common approach lends itself to the development of a software architecture.

Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns. In general, a particular system is defined in terms of a collection of components and interactions among these components. Such a system may in turn be used as a (composite) element in a larger system design. Indeed, a good software architecture will involve reuse of established engineering knowledge [19].

In this seminal paper we propose and describe the INTERPRETOR software architecture for interpreting and summarising high frequency noisy data sets. INTERPRETOR was inspired by the software architecture of ASSOCIATE [18] for interpreting Intensive Care Unit monitor data and ABSTRACTOR [16] for interpreting building sensor data - both systems have common features which facilitates a generic architecture. INTERPRETOR is based on the pipe and filter software architecture and consists of 3 consecutive processes: Filter which takes the original data and removes outliers, inconsistencies and noise; Abstraction which takes the filtered data and derives abstractions; and Interpretation which takes the abstractions and provides an interpretation and summarisation of the original data.

The structure of this paper is as follows. Section 2 discusses related work. Section 3 describes the INTERPRETOR software architecture to interpret and summarise high frequency noisy data Section 4 describes how the INTERPRETOR software architecture has been applied to 2 case studies. Final conclusions are given in section 5.

2 Related Work

A common architecture used to interpret high frequency data is the *blackboard* as used by ([1],[10], [14]). A blackboard system consists of a set of independent modules, called Knowledge Sources (KSs) that contain the domain-specific knowledge in the system, and a blackboard which is a shared data structure to which all the KSs have access. When a KS is activated it examines the current contents of the blackboard and applies its knowledge either to create a new hypothesis and write it on the blackboard, or to modify an existing one [13]. INTERPRETOR's architecture is similar to that of the blackboard in that individual tasks are performed by separate processes. However since the knowledge of blackboards are distributed, problems would arise with the co-ordination of knowledge sources which have competing obligations.

Another approach for interpreting high frequency and noisy data is the *service-oriented architecture (SOA)* used by ([4], [8], [20]). The SOA consists of components which can handle numerous distributed agents to allow the interpretation of data. Due to the architecture of SOA, extra functionality in the form of security for message passing between agents has had to be incorporated. Since INTERPRETOR is not a distributed system, there is no requirement for these extra services.

Another approach is a *multi-layered architecture* ([7], [21]). A multi-layered system is organized hierarchically where each layer provides service to the layer above it and serves as a client to the layer below it. The authors of [7] proposed a five layered generic and scalable architecture which uses components, middleware and agent technologies. Though a demonstrator was developed as a proof that the proposed conceptual software architecture is feasible in practice, there was no actual data or results to support their design. In contrast our proposed software architecture has been proven to work effectively.

Another approach is to have a *multi-agent architecture* ([2], [3], [12]). An agent is an autonomous computational process that inhabits an Agent Platform. An Agent platform provides the physical infrastructure in which agents are deployed and consists of the machines, operating systems, agent management components, the agents themselves and any additional support software. Agents typically offer one or more computational services that can be published as a service description. Agents typically communicate with each other to fulfill a task. Again, since INTERPRETOR is not a distributed system, there is no requirement for extra services which a multi-agent system would require.

A *non-hierarchical architecture* is SIMON [5]. SIMON (Signal Interpretation and MONitoring) is a system for monitoring neonates in the ICU. SIMON consists of a number of modules implemented as independent UNIX processes, communicating with each other through an inter-process communication (IPC) message protocol. The problem with such an architecture is the scheduling of the various modules. Care must be taken that the shared memory does not get corrupted by simultaneous writes. Another problem of SIMON is that it is solely an event driven architecture - it functions with discrete and infrequently determined input. INTERPRETOR deals with continuous and discrete data and does not have scheduling issues.

Another non-hierarchical approach is to use sequential processes. VIE-NET [11] a monitoring and therapy planning system for the artificial ventilation of newborn infants and resembles a em pipe and filter architecture where by a component reads streams of data on its inputs and produces streams of data on its outputs for another component to process. VIE-NET is made up of four sequential modules: *Data Selection* which filters out context-relevant data for further analysis; *Data Validation* which arrives at reliable measurements by detecting faulty data; *Data Abstraction* which transforms quantitative data of the observable system into qualitative values; and *Data Interpretation and Therapy Recommendation* which performs patient state assessments and associated therapy advise. INTERPRETOR strongly resembles VIE-NET except that INTERPRETOR does not perform data validation (though it could be performed as part of the Interpretation module) nor does it generate therapy recommendations because it is a generic architecture

3 Software Architecture

Figure 1 shows the Context Diagram of the INTERPRETOR system. The INTERPRETOR system takes high frequency noisy data and other relevant data to assist in interpretation from various input sources and presents to various output sources an interpretation of the original data.

Figure 2 shows the data flow in the INTERPRETOR system of Figure 1. Data is initially filtered to get rid of noise; rather than reasoning on a point to point basis, the resulting data stream is then converted by a second process into abstractions - this is a form of data compression. These abstractions are interpreted by a third process to provide an assessment of the original data.

We, therefore, derive the overall software architecture of the INTERPRETOR System in form of a Structure Chart as shown in Figure 3.

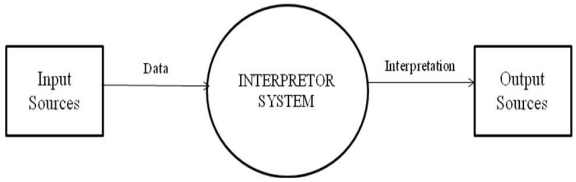

Figure 1: Context Diagram of the INTERPRETOR System

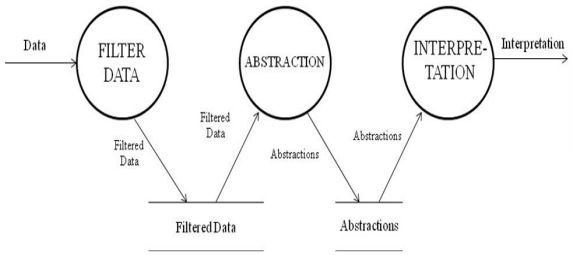

Figure 2: Data Flow Diagram of the INTERPRETOR System

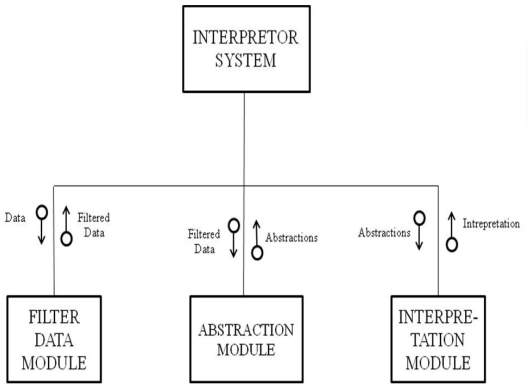

Figure 3: Overall Software Architecture of the INTERPRETOR System

It can be seen that INTERPRETOR is a data flow architecture. The architecture is decomposed into 3 processes which can be changed or replaced independently of the others - this makes INTERPRETOR a loosely coupled system. Indeed, each process of the INTERPRETOR performs one task or achieves a single objective - this makes the INTERPRETOR a highly cohesive system.

INTERPRETOR can be considered a pipe and filter architectural style because it provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component (process) which reads streams of data on input and produces streams of data on output. A local incremental transformation to input stream is made and the output begins before input is consumed. Data is passed through pipes between adjacent filters - they are the conduits for streams and transmit outputs from one filter to input of another. The advantage of this approach is the overall behavior is a simple composition of behavior of individual filters. The architecture facilitates reuse so any two filters can be connected if they agree on that data format that is transmitted. There is ease of maintenance because any filter can be changed or replaced depending on the application.

We hope to extend our INTERPRETOR design architecture, such that we have a generic design pattern for voluminous and high frequency noisy data, whereby, the data is passed through 3 consecutive processes: *Filter Data* which takes the original data and removes outliers, inconsistencies or noise; *Abstraction* which takes the filtered data and abstracts features from the filtered data; and *Interpretation* which uses the abstractions and generates an interpretation of the original data.

4 Applications of INTERPRETOR

We will demonstrate the application of the INTERPRETOR software architecture to 2 case studies: Interpreting Intensive Care Unit (ICU) monitor data and interpreting building monitor data.

4.1 Case Study 1 - Interpreting ICU Monitor Data

The ICU bedside monitors confront the medical staff with large amounts of continuous noisy data - this is emphasised when there are many cardiovascular parameters such as the heart rate and blood pressure being recorded simultaneously. The frequency of the data can be higher than 1 value every second which creates information overload for medical staff who need to interpret the data to evaluate the state of the patient.

A system called ASSOCIATE [15] has been developed using the the INTERPRETOR software architecture to interpret the ICU monitor data. We shall describe how ASSOCIATE implemented each of the modules of the the INTERPRETOR software architecture.

4.1.1 Filter Module

1. copy the first k values of the physiological data to be the first k values of the cleaned physiological data
2. **for** n = (k+1) to (number of points in the physiological data– k) **do**
3. create a window of points from (n-k) to (n+k).
4. sort the values in this window – this will force extreme values to the ends of the window
5. set the n^{th} value of the cleaned physiological data to be the median value of the sorted window
6. **end for**
7. copy the last k values of the physiological data to be the last k values of the cleaned physiological data

Figure 4: Algorithm for Filter Data Module

Filtering is the process of removing certain noise like clinically insignificant events from the physiological parameters. Clinically insignificant events which can not be removed at this stage will be dealt with by the Interpretation process.

After various investigations of filtering techniques, a median filter was chosen. The median filter involves a moving window which is centered on a point xn and if the window is of size 2k+1 the window contains the points x_{n-k} to x_{n+k}. By always choosing the median value in the window as the filtered value, it will remove transient features lasting shorter than k without distortion of the base line signal; features lasting more than that will remain. A summary of the algorithm for applying the median filter to our physiological data is shown in Figure 4.

4.1.2 Abstraction Module

Given continuous data (up to one value every second), it is computationally expensive to reason with each data value on a point to point basis - this data needs to be reduced by performing abstraction. Abstraction is the classification of filtered data generated by the filtering process into temporal intervals (trends) in which data is steady, increasing and decreasing. One is also interested in the rate of change e.g rapidly increasing, slowly decreasing etc. One must decide the beginning and end of an interval.

Our algorithm for identifying trends involves following two consecutive sub-processes called temporal interpolation and temporal inferencing. Temporal interpolation takes the cleaned data and generates simple intervals between consecutive data point. Temporal inferencing takes these simple intervals and tries to generate trends - this is achieved using 4 variables: *diff* which is the variance allowed to derive steady trends, g_1 and g_2 which are gradient values used to derive increasing and decreasing trends and *dur* which is used to merge 3 intervals based on the duration of the middle interval. Temporal Inferencing rules to merge 2 meeting intervals (Δ_{H2}) and 3 meeting intervals (Δ_{H3}) use the 4 variables to try to merge intervals into larger intervals until no more merging can take place. The algorithm for abstraction is summarised in 5. For further discussion of the algorithm the reader is advised to read [17].

1. Apply the inferences in Δ_{H2} which derive only increasing or decreasing trends by trying to combine the first two intervals; if this succeeds try to combine this new interval with the next and so on. If combination fails, then we take the interval which failed to be combined, and use it as a new starting point.
2. Apply inferences in Δ_{H2} which derive only steady trends.
3. Set flag *still-to-do* to **true**.
4. **while** *still-to-do* **do**
5. Set *previous* to the number of intervals generated so far.
6. Apply the inferences in Δ_{H3}
7. Apply the inferences in Δ_{H2}
8. Set *still-to-do* to *previous* = current number of intervals.
9. **endwhile**

Figure 5: Algorithm for Abstraction Module

4.1.3 Interpretation Module

Interpretation is based on defining a trend template for each type of event we wish to identify - examples of trend templates are shown in Figure 6. A given trend template will specify criteria which apply both within intervals and between intervals. The two relationships of interest between intervals are: meeting where the end time of one interval is the same as the start time of the other; and overlapping where there exists a time which is common to both intervals.

The algorithm for interpretation involves applying the templates to the temporal intervals. Clinically insignificant event and clinical condition templates initially have the status absent and therapy templates initially have the status working. The reasoning engine assesses the status of the templates (i.e hypothesised or confirmed) by evaluating the expressions located in the HypothesiseConditions and ConfirmConditions slots with the data. Actions to be performed when the templates are hypothesised or confirmed are provided in the HypothesiseActions and ConfirmActions slots. If

we have a template which has a hypothesised status over a number of adjacent segments which are subsequently confirmed then in retrospect we change these hypothesised states to confirmed. This is a way of confirming our initial beliefs. All segments with clinically significant templates that have confirmed states represent the interpretation.

Trend templates encompass three types of knowledge: temporal, differential and taxonomical. Temporal knowledge allows temporal reasoning; interval-based and point-based reasoning. Interval-based temporal reasoning is achieved using the still_developing and together functions. Given a clinical condition which is described in terms of overlapping intervals, the still_devel oping function operates on the uncertain period between the hypothesised state and the confirmed state of the clinical condition. Here the still_developing function is satisfied if there is the correct temporal progression from the hypothesised state to the confirmed state. Similarly the together function operates on overlapping temporal intervals which make up clinically insignificant events. Here the together function is satisfied if the overall changes in all the individual parameters that make up the event all share a common time interval. Though defined differently, the together and still_developing functions take into account the expected changes of the individual parameters that make up specific events do not occur at exactly the same time.

Point-based temporal reasoning is used to determine the outcome of therapy. It is known that clinicians expect changes in parameters to be achieved by a lower and upper temporal bound represented as time points in the future. ASSOCIATE expresses point based temporal reasoning within temporal intervals. When therapy is administered at a specific point in time, we compare a (future) interval which contains the therapy's temporal bound (lower and upper) with the interval which contained the time of administration. We are interested in whether parameters have increased, decreased or remained the same in the future after the time of administration.

Since several clinical conditions may be described by the same patterns, differential knowledge can be used to eliminate possibilities and hence prevent unnecessary reasoning. Information such as the patient record which contains the patient's history can be used as differential knowledge.

Also within the trend templates there is taxonomical knowledge - since several clinical conditions have similar attributes, this enables us to represent them as a hierarchy of classes and subclasses. Such a representation allows more abstract clinical conditions to be identified

```
Interpretation respiratory_problem
   Description ("general class of respiratory condition")
   Type_of (clinical_condition)
   Preconditions (NIL)
   HypothesiseConditions( AND (paO2,_,_,steady)
                              (paCO2,_,_,increasing))
   ConfirmConditions( AND (paO2,_,_,decreasing)
                          (paCO2,_,_,increasing))
   HypothesiseActions (ALARM_WARN)
   ConfirmActions (ALARM_TRUE)
end_template

Interpretation pneumothorax
    Description ("pneumothorax")
    Type_of (respiratory_problem)
    Preconditions (NIL)
    HypothesiseConditions(AND  (mean_bp,_,_,steady)
                               (heart_rate,_,_,increasing))
    ConfirmConditions (AND (mean_bp _,_,increasing)
                           (heart_rate _,_,increasing))
    HypothesiseActions (ALARM_WARN)
    ConfirmActions (ALARM_TRUE)
end_template

Interpretation digoxin
   Description ("digoxin")
   Type_of (therapy)
   Preconditions (digoxin)
   HypothesiseConditions (heart_rate increased 10)
   ConfirmConditions (heart_rate increased 20)
   HypothesiseActions (ALARM_ALERT)
   ConfirmActions (ALARM_TRUE)
end_template

Interpretation transcutenous_probe_off
  Description ("transcutenous probe coming off")
  Type_of (insignificant_event)
  Preconditions (NIL)
  HypothesiseConditions (NIL)
  ConfirmConditions(AND (meeting (paO2,_,>16,increasing)
                                 (paO2,>16,>16,steady)
                                 (paO2, _, >16, decreasing))
                        (meeting (paCO2,<3,_,decreasing)
                                 (paCO2,<3,<3,steady)
                                 (paCO2, <3, _, increasing)))
   HypothesiseActions (NIL)
   ConfirmActions (REMOVE)
end_template
```

Figure 6: Possible templates for clinical conditions, insignificant events and therapies

- if a specific instance of a clinical conditions cannot be identified then the more general class of clinical condition to which it belongs is more likely to describe the data. For further discussion of the algorithm the reader is advised to read [15].

4.1.4 Results

ASSOCIATE has been tested on three datasets from an adult ICU (here each set covers 24 hours and contains measurements of heart rate, mean blood pressure and central venous pressure) and six datasets from a neonatal ICU (here each set covers about 60 hours and contains measurements of heart rate, mean blood pressure, partial pressure of oxygen and partial pressure of carbon dioxide). The data sets were taken in 1995 as part of a research project and the results were validated by a consultant aneasthetist and a consultant neonatologist.

Overall, ASSOCIATE has a false-positive rate of 28.9% and a false-negative rate of 0.3% in identifying clinically insignificant events, a false-positive rate of 10.7% and a false-negative rate of 0.15% in identifying clinical conditions and a false-positive rate of 0% and a false-negative rate of 87.9% in determining the outcome of therapy. Since all have a true positive rate which is higher than its false positive rate, ASSOCIATE can be seen as a conservative system [9].

As an example, consider a three day data set taken from an ICU from from 00:01 on 22 April 1995 to 23:59 on 24 April 1995; the frequency of the signal is one data item per minute. No prior knowledge of events that occurred within this data set was known to the expert or the tester. Figure 7 depicts the physiological data from ICU patient monitors and Figure 8 depicts a graphical summary of the temporal intervals generated for each parameter by the Abstraction Module. Note that in the graphs HR represents the Heart Rate, BP represents the Blood Pressure, PO represents the Partial Pressure of Oxygen and TCO represents the Partial Pressure of Carbon Dioxide.

All clinically insignificant events were correctly identified and removed.

For the clinical condition interpretation, the expert agrees that ASSOCIATE identified all 11 episodes of respiratory problems in the data. Of 2 of these episodes, namely those identified from 11:44 on 23/04/95 to 12:04 on 23/04/95 and from 13:57 on 23/04/95 to 14:32 on 23/04/95 may have been pneumothoraxes. However, ASSOCIATE incorrectly identifies respiratory problems on 5 occasions. ASSOCIATE also incorrectly identifies a pulmonary haemorrhage and a pneumothorax at the same time, though the expert agrees that there is a respiratory problem at this time.

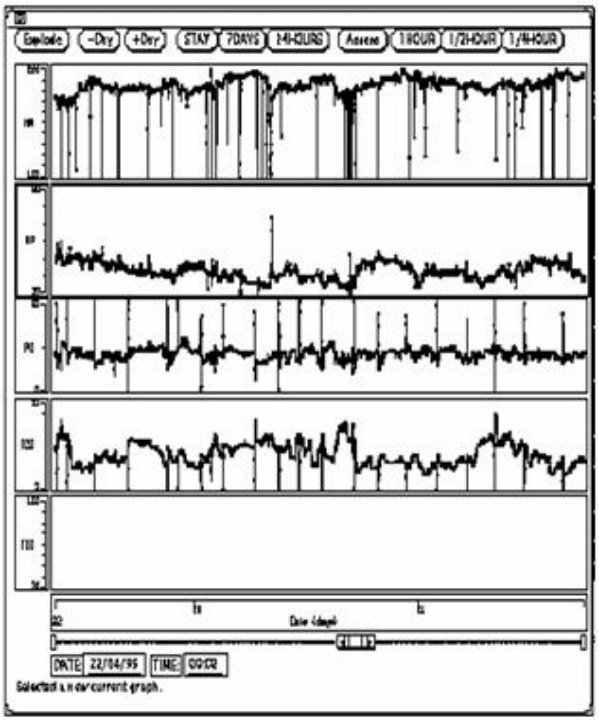

Figure 7: Original Physiological Data from ICU Patient Monitor

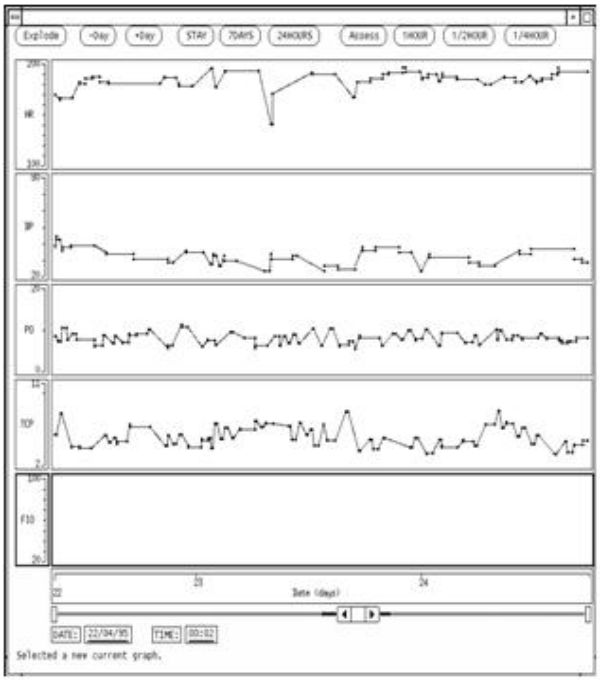

Figure 8: Graphical Summary generated by the Abstraction Module

ASSOCIATE identified 3 separate episodes of shock of which the expert agreed with 2 of them. The expert also agrees in ASSOCIATE's identifications of episodes of tachycardia and hypercarbia. However, a few of the episodes of hypoxaemia were incorrectly identified due to noisy data.

The expert agreed that ASSOCIATE recognised all clinical conditions in the data set i.e no clinical conditions were missed.

For the therapy interpretation, 6 therapies were administered. Of the 5 that worked ASSOCIATE correctly identifies 2 of them as working. ASSOCIATE correctly identifies the therapy that did not work. The incorrect results were because of noisy data and approximate times of administration.

4.2 Case Study 2 - Interpreting Building Sensor Data

Building operators are confronted with large volumes of continuous data from multiple environmental sensors which require interpretation. The ABSTRACTOR ([18], [16]) system used the INTERPRETOR software architecture to summarise historical building sensor data for interpretation and building performance assessment. We shall describe how ABSTRACTOR implemented each of the modules of the INTERPRETOR software architecture.

4.2.1 Filter Module

1. copy the first k values of the environmental data to be the first k values of the filtered environmental data
2. **for** n = (k+1) to (number of points in the environmental data- k) **do**
3. create a window of points from (n-k) to (n+k).
4. calculate the average of the values in this window
5. set the nth value of the filtered environmental data to be the average value of the window
6. **end for**
7. copy the last k values of the environmental data to be the last k values of the filtered environmental data

Figure 9: Algorithm for Filter Data Module

Initially data needs to be filtered to get rid of non-significant events in environmental monitoring data. Due to the nature and frequency of the data, an average filter was chosen - here all the very short duration spikes from the outdoor temperature data were removed whilst revealing the short duration trends hidden in the raw data. The algorithm for the filter module is given in Figure 9.

4.2.2 Abstraction Module

This module is exactly the same as the agglomerative approach used for case study 1 - for a discussion of this algorithm applied to building monitor data the reader is advised to read [20].

4.2.3 Interpretation Module

Given overlapping trends it is proposed, in the spirit of [6] they are split into global segments. A change in the direction of change (slope) of one (or more) channels or a change in the rate of change of one (or more) channels contributes to a split in the trends creating a global segment. A global segment can be considered as being a set of intervals - one for each channel.

```
if heat-flux increasing                 if heat-flux decreasing
   and ti-t0 decreasing then               and ti-t0 steady then
         fault detected                          fault detected
end if                                  end if

if heat-flux increasing                 if heat-flux steady
   and ti-t0 steady then                   and ti-t0 increasing then
         fault detected                          fault detected
end if                                  end if

if heat-flux decreasing                 if heat-flux steady
   and ti-t0 increasing then               and ti-t0 decreasing then
         fault detected                          fault detected
end if                                  end if
```

Figure 10: Example of rules to apply to global segments

The algorithm for interpretation involves applying rules to the global segments. Examples of rules for identifying faults are shown in Figure 10 - here a fault is declared when the heat-flux does not have the same trend as the difference in internal and external temperature (t1-t0). If rules are true over adjacent global segments then one can determine when the fault started and ended.

4.2.4 Results

ABSTRACTOR has been tested on over 8 days (121 79 minutes) worth of continuous data (see Figure 11a). The data was the heat-flux into a wall and the difference in internal and external temperature (ti-t0) measurements; the sampling frequency of the signals is one data item every 15 minutes. No prior knowledge of events that occurred within this data set was known to the expert or the tester. The application of the average filter (k=10 filter provides a running five and a quarter hour running average) is shown in the middle graph 11(b) and the intervals generated are shown in the bottom graph 11(c).

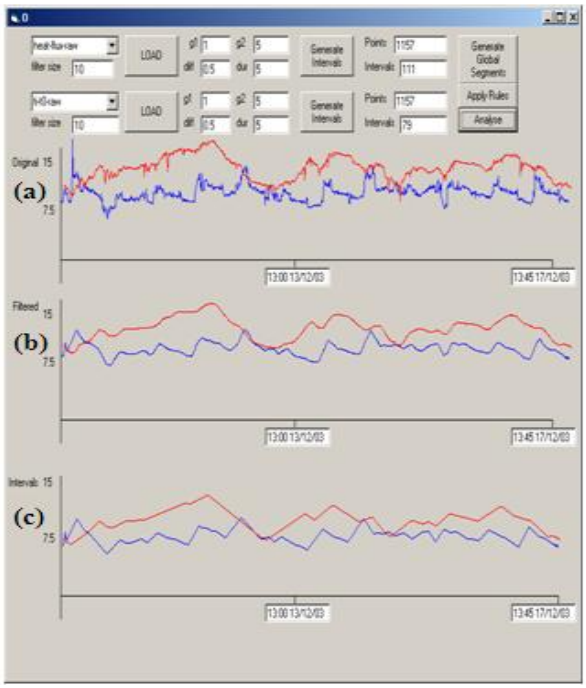

Figure 11: Output of ABSTRACTOR

Overall, ABSTRACTOR has a sensitivity of 56%, specificity of 64%, predictive value of 43%, a false positive rate of 57% and a false negative rate of 24%. These results mean that when a fault is present ABSTRACTOR is detecting it only 56% of the time but when there is no fault it will correctly identify this 64% of the time. Whilst it would seem that ABSTRACTOR is only slightly better than tossing a coin to decide the presence or absence of a fault it needs to be remembered that the actual fault conditions were derived from an expert's manual abstraction of the raw data which is dependent on the expert's attitude and experience. A direct comparison with the raw data is meaningless because the data is at intervals much shorter than the trends. If ABSTRACTOR were to be incorporated in its present state into a control system it would generate a high number of false alarms (57%) but would fail to detect a fault only 24% of the time. These results are indicating that ABSTRACTOR is a more liberal system than a random system [9].

5 Conclusions

The interpretation of high frequency and noisy data is non-trivial. The INTERPRETOR software architecture is designed in such a way to allow the interpretation of high frequency and noisy data and the results of INTERPRETOR are encouraging. We have shown that INTERPRETOR can be applied to different domains which have the same issues associated with the interpretation of voluminous and noisy data. Our future work will be to develop a tool for our software architecture which should lend itself for reuse and then validate it with further case studies.

In summary, INTERPRETOR reasons with multiple signals in an intuitive way. Although it is not perfect, it is a step forward in the development of systems for the interpretation of voluminous high frequency and noisy data.

References

[1] Beigl, M., Beuster, M., Rohr, D., Riedel, T., Decker, C., and Krohn, A. S2b2: Blackboard for transparent data and control access in heterogeneous sensing systems. pages 126–129, 2007.

[2] Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. Jade: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology*, 50(1-2):10–21, 2008.

[3] Carrascosa, C., Bajo, J., Julian, V., Corchado, J. M., and Botti, V. Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems with Applications*, 34(1):2–17, 2008.

[4] Crawford, C. H., Bate, G. P., Cherbakov, L., Holley, K., and Tsocanos, C. Toward an on demand service-oriented architecture. *IBM Systems Journal*, 44(1):81–107, 2005.

[5] Dawant, B. M., Uckun, S., Manders, E. J., and Lindstrom, D. P. Soda: Service oriented device architecturethe simon project - model-based signal acquisition, analysis, and interpretation in intelligent patient monitoring. *IEEE Engineering In Medicine and Biology*, 12(4):82–91, 1993.

[6] DeCoste, D. Dynamic across-time measurement interpretation. *Artificial Intelligence*, 51:273–341, 1991.

[7] Decruyenaere, J., DeTurck, F., Vanhastel, S., Vandermeulen, F., P Demeester, P., and deMoor, G. On the design of a generic and scalable multilayer software architecture for data flow management in the intensive care unit. *Journal of Methods of Information in Medicine*, 3:79–88, 2010.

[8] deDeugd, S., Carroll, R., Kelly, K. E., Millett, B., and Ricker, J. Soda: Service oriented device architecture. *IEEE Pervasive Computing*, 5(3):94–96, 2006.

[9] Fawcett, T. Roc graphs: Notes and practical considerations for data mining researchers. Technical Report HPL-2003-4, Intelligent Enterprise Technologies Laboratory, HP Labs Palo Alto, January 2003.

[10] Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., Collinot, A., Vina, A., and Seiver, A. Guardian: A prototype intelligent agent for intensive-care monitoring. Technical Report KSL 91-42, Knowledge Systems Lab Report, Department of Computer Science, Stanford University, June 1991.

[11] Miksch, S., Horn, W., Popow, C., and Paky, F. Context-sensitive data validation and data abstraction for knowledge-based monitoring. Technical Report TR-94-04, Austrian Research Institute for Artificial Intelligence, 1994.

[12] Mistry, M. and Shah, D. Implementation of multi-agents system in health care domain. pages 100–104, January 2011.

[13] Rich, E. *Artificial Intelligence*. 1988.

[14] Rudenko, D. and Borisov, A. An overview of blackboard architecture application for real tasks. volume 31, pages 50–57, 2007.

[15] Salatian, A. Interpreting historical icu data using associational and temporal reasoning. volume 4, pages 442–450, 2003.

[16] Salatian, A. A software architecture for decision support of building sensor data. *International Journal of Smart Home*, 4(4):27–34, 2010.

[17] Salatian, A. and Hunter, J. R. W. Deriving trends in historical and real-time continuously sampled medical data. *Journal of Intelligent Information Systems*, 13:47–74, 1999.

[18] Salatian, A. and Oriogun, P. A software architecture for summarising and interpreting icu monitor data. *International Journal of Software Engineering*, 4(1):3–14, 2011.

[19] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. New Jersey, 1996.

[20] VanHoecke, S., DeTurck, F., Danneels, C., DeProft, K., Taveirne, K., and Decruyenaere, J. Platform for intelligent agent subscription in icu. 2006.

[21] Yang, M., Wang, S., Abdelal, A., Jiang, Y., and Kim, Y. An improved multi-layered architecture and its rotational scheme for large-scale wireless sensor networks. *Las Vegas, NV, USA*, pages 855–859, Janaury 2007.