A Software Architecture for Decision Support of Building Sensor Data

Apkar Salatian
School of Information Technology and Communications
American University of Nigeria, Yola Bypass, PMB 2250, Yola, Nigeria
Email: apkar.salatian@aun.edu.ng

Abstract

Building operators need to interpret the high volumes of noisy data generated from the environmental sensors in their buildings. In this paper we propose a software architecture which processes the building monitor data to provide enhanced decision support in the form of assessments and accurate summaries to building operators in order to improve the quality of building performance. Our architecture has been tested on over 8 days worth of continuous data and the results are very encouraging.

Keywords: *Software architecture, signal processing, building, temporal reasoning.*

1. Introduction

Building operators are confronted with large volumes of continuous data from multiple environmental sensors which require interpretation. This data presents two major challenges.

The first challenge is to distinguish between significant and insignificant events. Insignificant events need to be processed for a number of reasons: retaining the data obtained during insignificant events and including them in further processing will give an inaccurate history of the building when, for example, requesting the mean temperature; and such data must be filtered out (ignored) by the building operator.

The second challenge is to interpret the large amounts of continuous data - this is emphasised when there are many environmental parameters being recorded simultaneously. Environmental parameters are usually interpreted in the context of other environmental parameters together with events that have happened in the past. It is far too time consuming for building operators to look at all the data to make informed decisions. Summaries are hence required. Building operators are interested in trends. Cross comparing between trends of different sensor signals allows staff to identify events.

In this paper we describe the *ABSTRACTOR* [1] software architecture which has been developed to provide enhanced decision support to building operators by removing environmentally insignificant events and performing summarisation and building state assessments. The software architecture was inspired by [2].

The structure of this paper is as follows. Section 2 describes ABSTRACTOR's software architecture to interpret high frequency noisy building data. Section 3 discusses the results of ABSTRACTOR. Section 4 discusses related work and final conclusions are given in section 5

2. Software Architecture

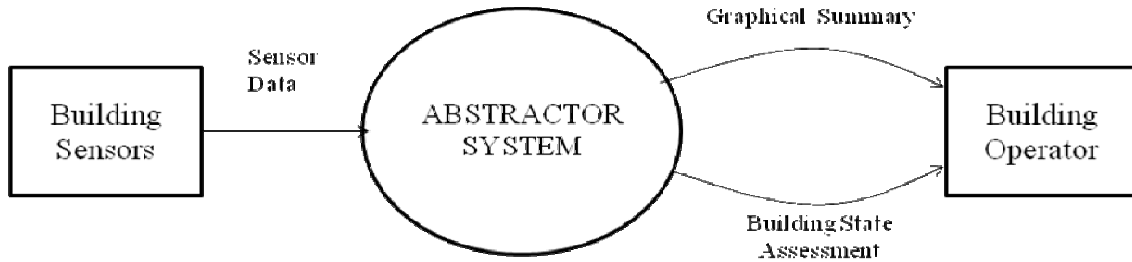

Figure 1 – Context Diagram of the ABSTRACTOR System

Figure 1 shows the Context Diagram of the ABSTRACTOR system. The ABSTRACTOR system takes sensor data from the environmental monitors attached to the building and presents to the Building operators a graphical summary of the sensor data in terms of trends to allow qualitative reasoning and building state assessments for decision support.

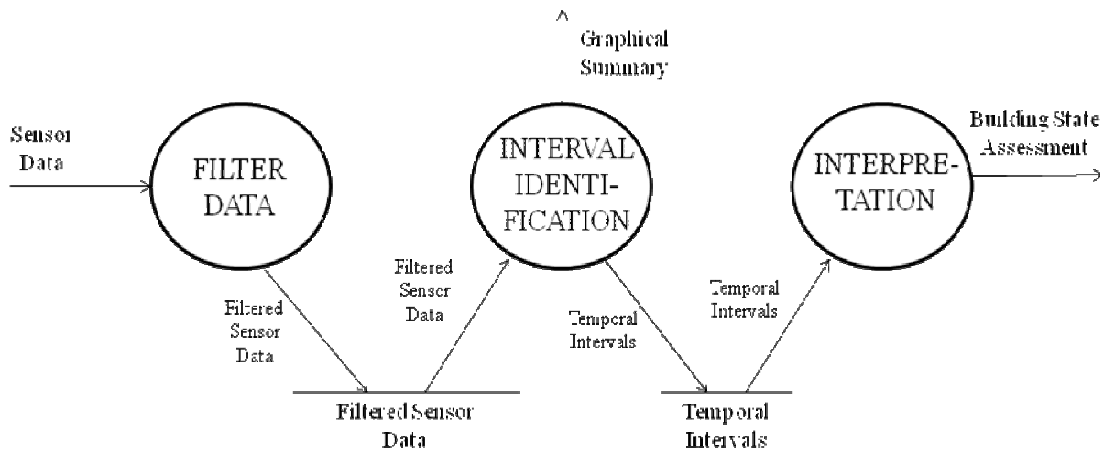

Figure 2 – Data Flow Diagram of the ABSTRACTOR System

Figure 2 shows the data flow in the ABSTRACTOR system of figure 1. Data is initially filtered to get rid of environmentally insignificant events; the resulting data stream is then converted by a second process into temporal intervals (trends) to provide a graphical summary of the data. These temporal intervals are interpreted by a third process which generates a building state assessment by identifying environmental events.

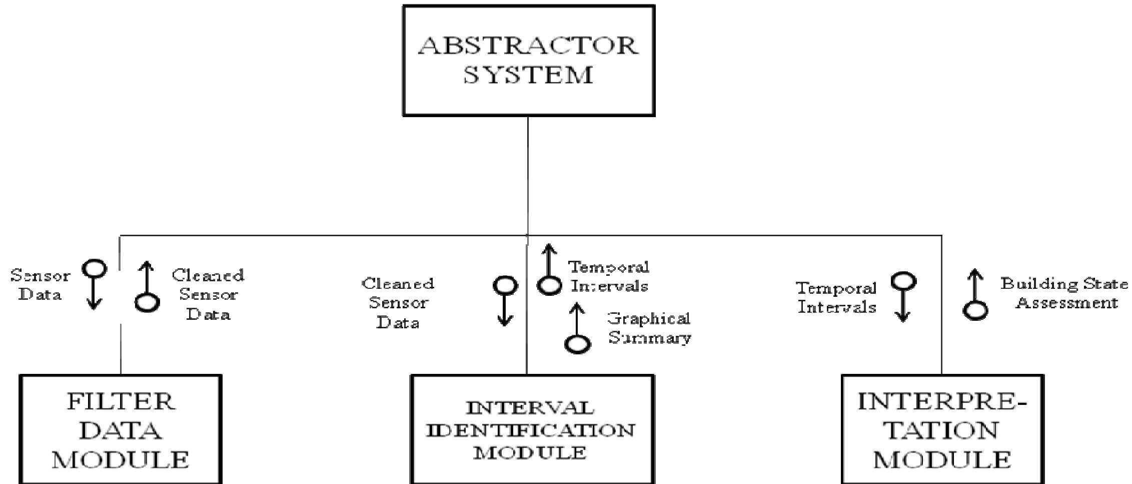

Figure 3: Overall Software Architecture of the ABSTRACTOR System

We, therefore, derive the overall software architecture of the ABSTRACTOR System in form of a Structure Chart as shown in Figure 3. Our software architecture defines the overall structure, the main modules and their connections of our system. By separating the system into 3 distinct independent modules we believe that ABSTRACTOR has low coupling and is highly cohesive. We shall describe each module in turn.

2.1. Filter Data Module

```
1.  copy the first k values of the sensor data to be the first k values of the cleaned sensor data
2.  for n = (k+1) to (number of points in the sensor data– k) do
3.          create a window of points from (n-k) to (n+k).
4.          take the average all the values in this window
5.          set the nth value of the cleaned sensor data to be
            the average value of the window
6.  end for
7.  copy the last k values of the sensor data to be the last k values of the cleaned sensor data
```

Figure 4: Algorithm for Filter Data Module

Initially data needs to be filtered to get rid of non-significant events in environmental monitoring data e.g air temperature spikes which occur during a cold spell when someone opens a window for a short period. If this happens infrequently then such events are insignificant and should be treated as noise and removed.

ABSTRACTOR uses an average filter with k=10 because it removes all the very short duration spikes from the outdoor temperature data whilst revealing the short duration trends hidden in the raw data. This is because a median filter removes transient features lasting shorter than half the width of the window hence events lasting more than half the width of the window will not be removed, a low-pass filter attenuates noise (noise may have some low-frequency components) and a high-pass filter eliminates low frequency variations and trends leaving only the higher frequency components.

2.2. Interval Identification Module

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 Interval Identification Module

Given continuous data it is computationally expensive to reason with each data value on a point to point basis. Interval Identification 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*. T*emporal interpolation* takes the cleaned data *a*nd 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, *g1* and *g2* 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 interval identification is summarised in figure 5. For further discussion of the algorithm the reader is advised to read [3].

2.3. Interpretation Module

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

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

Figure 6: Example of rules to apply to global segments

Given overlapping temporal intervals it is proposed, in the spirit of [4] 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 temporal

intervals creating a global segment. A global segment can be considered as being a set of intervals - one for each channel.

The algorithm for interpretation involves applying rules to the global segments to identify particular building events. If rules are true over adjacent global segments then one can determine when the environmental event started and ended. Examples of rules are shown in figure 6.

3. Results

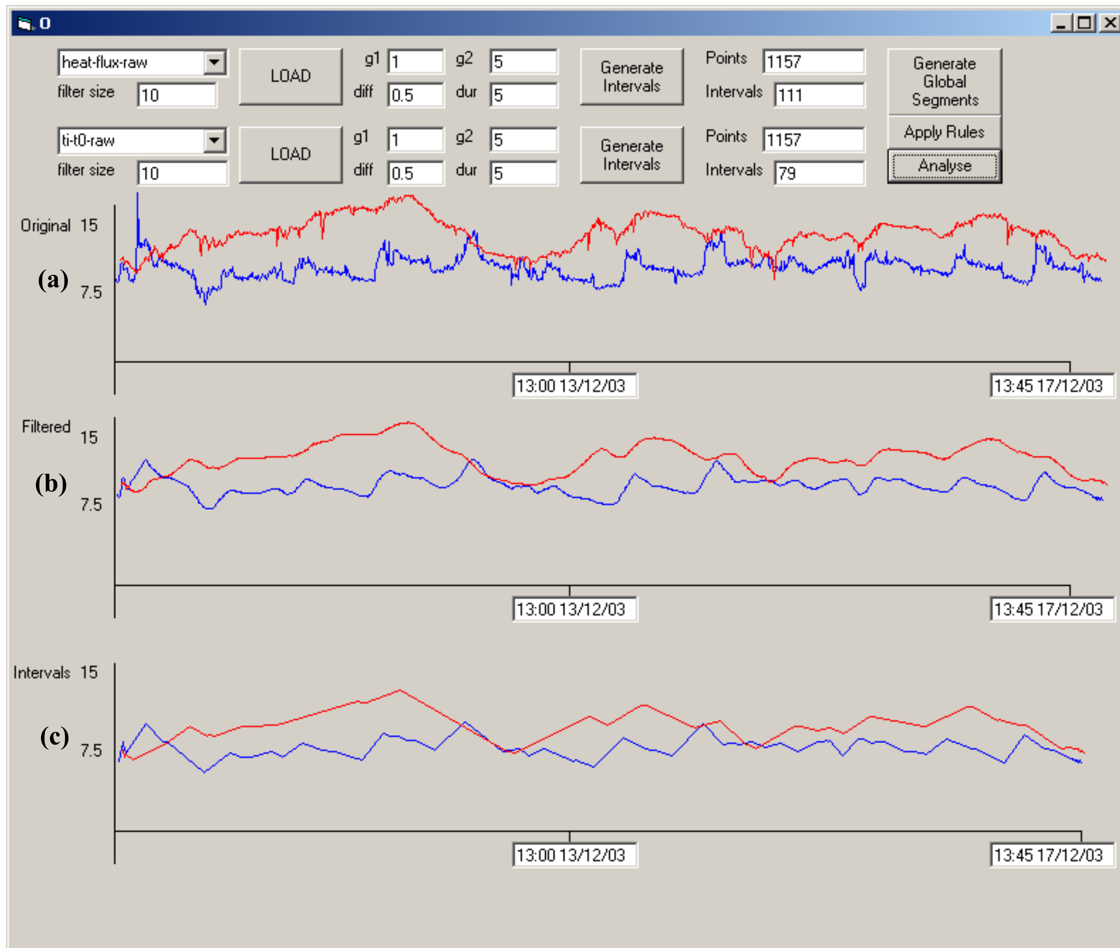

Figure 7: ABSTRACTOR applied to environmental data

ABSTRACTOR has been tested on over 8 days (12179 minutes) worth of continuous data (see figure 7a). 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 (b) and the intervals generated are shown in the bottom graph (c). 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 (b) and the intervals generated are shown in the bottom graph (c).

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 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 [5].

4. Related Work

A potential architecture for managing building monitor data is the *blackboard*. 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 [6].The blackboard is not suitable for our application because it is a way for the different modules to contribute to the blackboard to solve a particular problem – in our case our architecture has been developed to be a sequence of processes where the output of one process is the input to another.

Another potential approach is the *Service Oriented Architecture* (SOA). SOA defines how to integrate vastly disparate applications for a web-based environment and uses multiple implementation platforms. SOA allows interoperability between different systems and programming languages and provides the basis for integration between applications on different platforms through a communication protocol. SOA is achieved by using middleware which is computer software that connects the different software components and allows them to interact [7]. The SOA is too complex for our application – for our system we only need a single platform and though the modules could be written in different languages, the output of one process would be the input to the next so a simple communication protocol would be required.

Another approach to managing large volumes of data could be a *distributed architecture*. A distributed architecture consists of multiple autonomous processes that communicate through a computer network. The processes interact with each other in order to achieve a common goal. Distributed architecture eliminates the costs of middleware by allowing individual systems to participate in shared processes without needing to host matching middleware and without having to go through a centralized server. Like the SOA, a distributed architecture is too complex for our application – the only interaction between processes required to obtain the goal of our system is for the output of one process would be the input to the next. Moreover, our architecture can run on a single processor and does not require a network.

Another potential approach to interpreting building monitor data is to use the 3-tier architecture. The 3-tier architecture is similar to ABSTRACTOR - however, topologically they are different. The 3-tier architecture is a client-server architecture which consists of 3 tiers: a *presentation* tier which is the user interface for a client; a *logic* tier which controls the application's functionality by performing related processing; and a *data* tier which consists of database servers to store and maintain the data. A general rule in 3-tier architectures is the presentation (client) tier never communicates directly with the data tier - all communication

must pass through the logic tier. Like ABSTRACTOR, the 3-tier architecture is linear. In a 3-tiered system, it is expected to have different hardware for each tier whereas with ABSTRACTOR everything will be on one system – therefore, with the ABSTRACTOR architecture we save costs making it an inexpensive system.

5. Summary and Conclusions

The interpretation of building data is non-trivial. There is a need for i*nsignificant event* removal to give accurate summaries to building staff. Due to the vast amounts of data there is a need to derive trends in the data and reason qualitatively. A software architecture is required to capture these design decisions.

The ABSTRACTOR software architecture is designed in such a way to allow the interpretation of building data and the results of ABSTRACTOR are encouraging. The software architecture is simply a data flow system where the structure of the design is determined by the orderly linear motion of data from component to component.

In summary, ABSTRACTOR 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 building data.

.

References

[1] Salatian, A., & Taylor, B. (2008) 'ABSTRACTOR: An Agglomerative Approach to Interpreting Building Monitoring Data', Journal of Information Technology in Construction, Vol. 13, pages 193-211, ISSN 1874-4753, May 2008.

[2] Salatian, A., & Oriogun, P. (2011) 'A Software Architecture for Summarising and Interpreting ICU Monitor Data', International Journal of Software Engineering, ISSN: 1687-6954 (print version), ISSN: 2090-1801 (electronic version). In Press.

[3] Salatian, A., & Taylor, B. (2004) 'An Agglomerative Approach to Creating Models of Building Monitoring Data', Proceedings of the eighth IASTED International Conference on Artificial Intelligence and Soft Computing, pages 167-172, ISBN 0-88986-458-6, ISSN 1482-7913, Marbella, Spain, 2004.

[4] D.DeCoste. Dynamic Across-Time Measurement Interpretation, Artificial Intelligence 51, pages 273-341, 1991.

[5] T. Fawcett, ROC Graphs: Notes and Practical Considerations for Data Mining Researchers, Intelligent enterprise Technologies Laboratory, HP Labs Palo Alto, HPL-2003-4, Jan 7 2003.

[6] E. Rich, Artificial Intelligence, page 278, McGraw-Hill International editions, 1988.

[7] Huhns, M.N., Singh, M.P., Service-oriented computing: key concepts and principles, Internet Computing, IEEE, Volume 9, Issue 1, pages 75 – 81, ISSN: 1089-7801, 2005