Chapter 7
Validating the INTERPRETOR Software Architecture for the Interpretation of Large and Noisy Data Sets

Apkar Salatian
American University of Nigeria, Nigeria

ABSTRACT

In this chapter, the authors validate INTERPRETOR software architecture as a dataflow model of computation for filtering, abstracting, and interpreting large and noisy datasets with two detailed empirical studies from the authors' former research endeavours. Also discussed are five further recent and distinct systems that can be tailored or adapted to use the software architecture. The detailed case studies presented are from two disparate domains that include intensive care unit data and building sensor data. By performing pattern mining on five further systems in the way the authors have suggested herein, they argue that INTERPRETOR software architecture has been validated.

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 noise. Rather than reasoning 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.

Software architectures involve 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

DOI: 10.4018/978-1-4666-2208-1.ch007

may in turn be used as a (composite) element in a larger system design. Software architectures can act as a model of computation for data flows in a system. Indeed, a good software architecture will involve reuse of established engineering knowledge (Shaw & Garlan, 1996).

In this paper we will describe and validate the *INTERPRETOR* software architecture for interpreting large and noisy data sets. INTEREPRETOR was inspired by the software architecture of ASSOCIATE (Salatian & Oriogun, 2011) for interpreting Intensive Care Unit monitor data and ABSTRACTOR (Salatian, 2010) for interpreting building sensor data - both systems have common features which facilitates a generic architecture. INTERPRETOR consists of 3 consecutive processes: *Filter* which takes the original data and removes noise; Abstraction, which derives abstractions from the filtered data; and Interpretation, which takes the abstractions and provides an interpretation of the original data.

THE INTERPRETOR 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. A third process to provide an assessment of the original data interprets these abstractions.

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

It can be seen that INTEREPRETOR is a data flow architecture and model of computation. The architecture is decomposed into three 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 also be considered a pipe and filter architectural style because it provides a structure for systems that process a stream of data.

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 three 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.

APPLICATIONS OF THE INTERPRETOR SOFTWARE ARCHITECTURE

We will demonstrate the application of the INTERPRETOR software architecture to two case studies from the author's research endeavours: interpreting Intensive Care Unit (ICU) Monitor Data and interpreting building monitor data.

Figure 1. Context diagram of the INTERPRETOR system

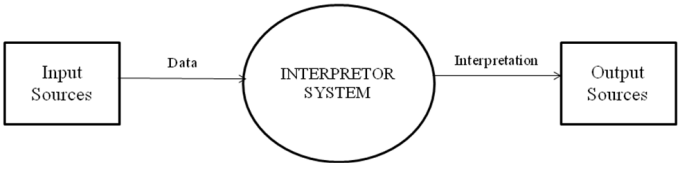
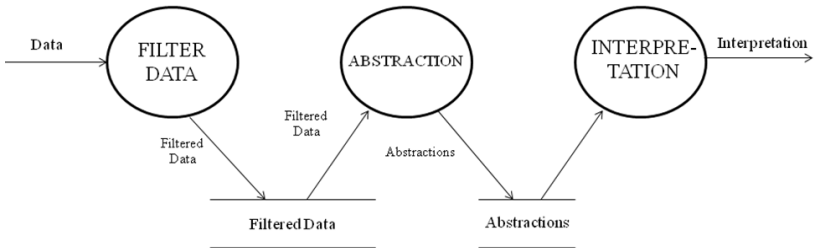


Figure 2. Data flow diagram of the INTERPRETOR system



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 (Salatian, 2003) has been developed using the INTERPRETOR software architecture to interpret the ICU monitor data. We shall describe how ASSOCIATE implemented each of the modules of the INTERPRETOR software architecture.

Filter Module

Filtering is the process of removing certain noise like clinically insignificant events from the physiological parameters. Clinically insignificant events which cannot 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 centred on a point x_n 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.

Figure 3. Overall software architecture of the INTERPRETOR system

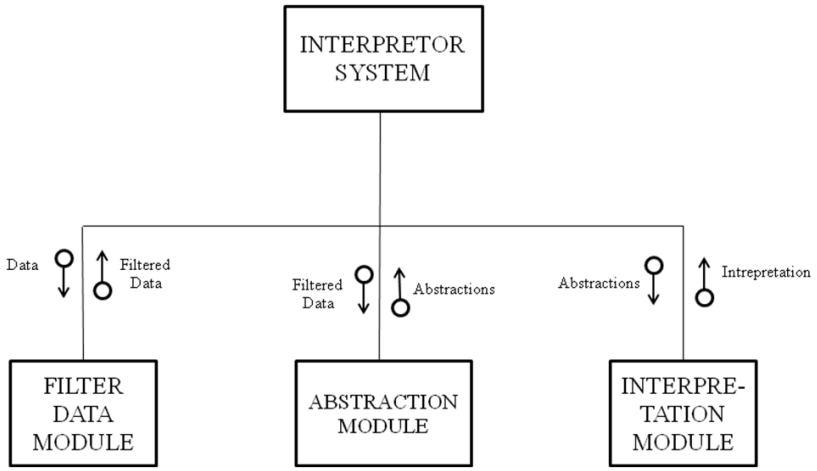


Figure 5. Algorithm for abstraction module

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 6. Possible templates for clinical conditions, insignificant events, and therapies

```
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 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

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
```

cal. 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_developing* 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)

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 must decide the beginning and end of an interval since they are not known in advance.

Our algorithm for identifying trends involves following two consecutive sub processes called temporal interpolation and *temporal inferencing*. Temporal 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 abstraction is summarised in Figure 5. For further discussion of the algorithm the reader is advised to read (Salatian & Hunter, 1999).

Interpretation Module

Interpretation is based on defining a *trend template* for each event we wish to identify - examples of trend templates are shown in Figure 6. A 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 matches the start time of the other; and *overlapping* where there exists a time that 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 *taxonomi-*

Figure 4. Algorithm for filter data 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

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 – 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 (Salatian, 2003).

Results

ASSOCIATE has been tested on three datasets from an adult ICU and six datasets from a neonatal ICU each set covering about 60 hours of data. 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 (Fawcett, 2003).

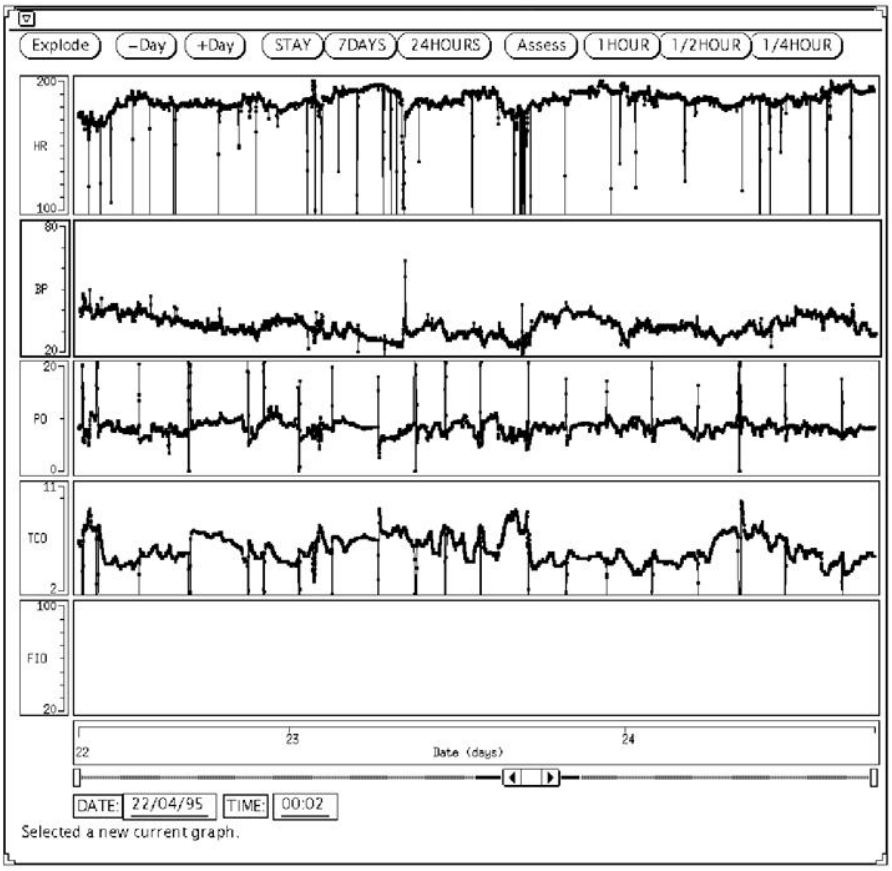
As an example, consider a three day data set taken from an ICU from from 00:01 on 22 April 1995 to 23:59 on 24 April 1995; the frequency of the signal is one data item per minute. The expert or the tester had no prior knowledge of events that occurred within this data set. Figure 7 depicts the physiological 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. ASSOCIATE also 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. Indeed, 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.

Figure 7. Original physiological data from ICU patient monitor



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 (Salatian & Taylor, 2008, Salatian & Taylor, 2011) 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.

Filter 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. The algorithm for the filter module is given in Figure 9

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 (Salatian & Taylor, 2004)

planning and *Realization*. Document planning is responsible for selecting the 'important' data points from the input data and to organize them into a paragraph - this is a form of filtering. Micro planning is responsible for lexical selection and ellipsis - this is a form of abstraction of the filtered data. Realization is essentially responsible for ordering of the phrases in the output and also to perform punctuation tasks - this is analogous to the Interpretation module of the INTERPRETOR software architecture.

A similar approach is taken by (Turner et al, 2008) to generate textual summaries of geo-referenced data based on spatial reference frames. From the initial data basic events are generated (filtered out) by a data analysis process which is then abstracted into higher-level concepts. The final stage is to interpret these messages in sentence form for textual summarization.

(Sun et al, 2005) extract extra knowledge from click-through data of a Web search engine to improve web-page summarization. Among the 3,074,678 Web pages crawled, the authors removed those which belong to 'World' and 'Regional' categories, as many of them are not in English - this filtering resulted in 1,125,207 Web pages, 260,763 of which are clicked by Web users using 1,586,472 different queries. Three human evaluators were employed to summarize (abstract) these pages. Each evaluator was requested to extract the sentences which he/she deemed to be the most important ones for a Web page. An interpretation of the precision of the query terms was finally reported.

(Knox et al, 2010) presented a case-based reasoning approach to activity recognition in a smart home setting. An analysis was performed on scalability with respect to case storage, and an ontology-based approach was proposed for case base maintenance - this could also lend itself to the INTERPRETOR software architecture. Firstly to create a cut-down (filtered) case base a reduction was made by firstly using a simple statistical technique, and then by semantically linking the case solutions with corresponding case features - this could be considered a form of abstraction. The case solutions were analysed and some had their accuracy reduced while others had theirs increased - this is considered a form of interpretation. The analysis was then reported in the form of graph.

CONCLUSION

The interpretation of large and noisy data is non-trivial - one approach is to have a software architecture which can be tailored and applied to different domains which have the same issues associated with the interpretation of data.

We have shown that research into trying to interpret large and noisy datasets do not actually follow any particular software architecture or framework - they just tell us about the 'tactics' they have employed in order to process such data. By conducting a detailed empirical study of the author's former research endeavours and pattern mining five further systems, we believe that we have successfully argued that our INTERPRETOR software architecture allows systems to be adapted at a much higher level of abstraction to facilitate the interpretation of large and noisy data leaving the tactics to the individual modules.

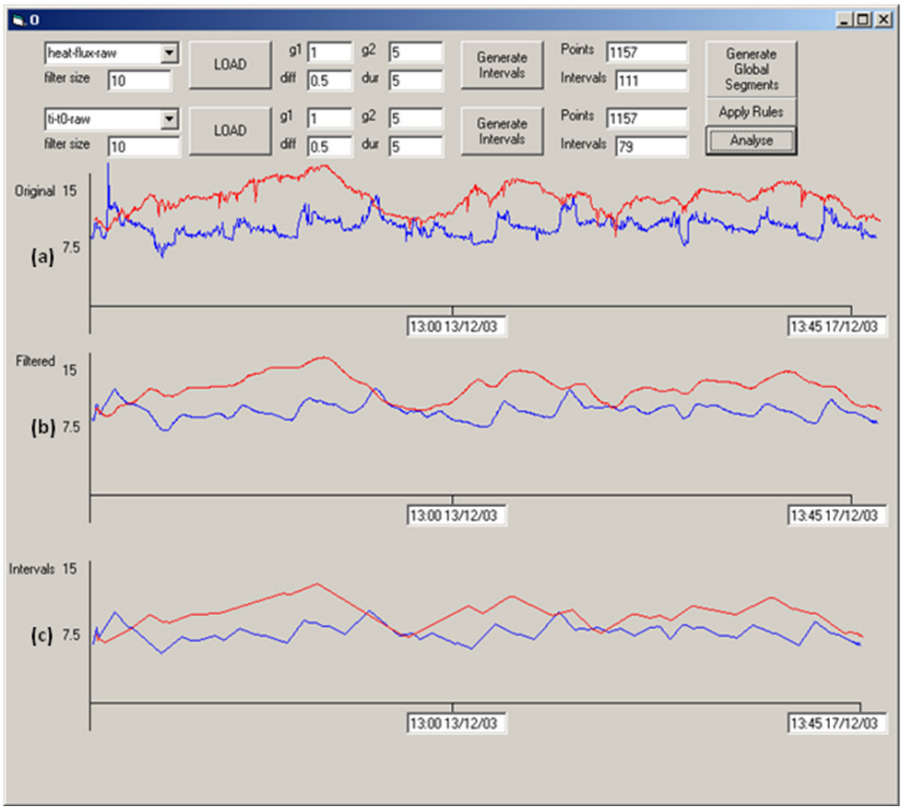
Our future work will be towards developing a generic software tool for this software architecture, which should lend itself for reuse.

REFERENCES

DeCoste, D. (1991). Dynamic across-time measurement interpretation. *Artificial Intelligence*, *51*, 273–341. doi:10.1016/0004-3702(91)90113-X.

Fawcett, T. (2003). *ROC graphs: Notes and practical considerations for data mining researchers*. Palo Alto, CA: HP Labs.

Figure 11. Output of ABSTRACTOR



VALIDATION OF THE INTERPRETOR SOFTWARE ARCHITECTURE

By performing pattern mining in the form of tailoring and adapting different systems to perform filtering, abstraction and interpretation we will now further validate the INTERPRETOR software architecture.

BT-45 (Portet et al, 2009) generates natural language textual summaries of continuous physiological signals and discrete events from a Neonatal Intensive Care Unit. BT-45 could be adapted to use the INTERPRETOR software architecture. The first stage of BT-45 is *Signal Analysis, which* extracts the main features of the physiological time series - this fulfils the role of the Filter module of INTERPRETOR. BT-45 then performs *Data* Interpretation, *which* performs some temporal and logical reasoning to infer more abstract medical observations and relations from the signal features which can considered the Abstraction module of INTERPRETOR. The next stages of BT-45 are *Document Planning* which selects the most important events from earlier stages and groups them into a tree of linked events then *Microplanning and Realisation* which translates this tree into coherent text for reporting - collectively they could be considered the Interpretation module of INTERPRETOR.

Sumtime-Mousam (Sripada et al, 2003) is a text generator that produces textual marine weather forecasts for offshore oilrig applications. It uses a subset of the processes of BT-45 and also follows the INTERPRETOR software architecture. The architecture of SUMTIME-MOUSAM follows 3 processes: *Document planning, Micro*

Interpretation Module

Given overlapping trends it is proposed, in the spirit of (DeCoste, 1991) they are split into *global segments*. A change in the direction of change 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.

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.

Results

ABSTRACTOR has been tested on over eight days (12179 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. The expert or the tester had no prior knowledge of events that occurred within this data set. 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%, and 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 that 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 (Fawcett, 2003).

Figure 10. Example of rules to apply to global segments

```
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 8. Graphical summary generated by the abstraction module

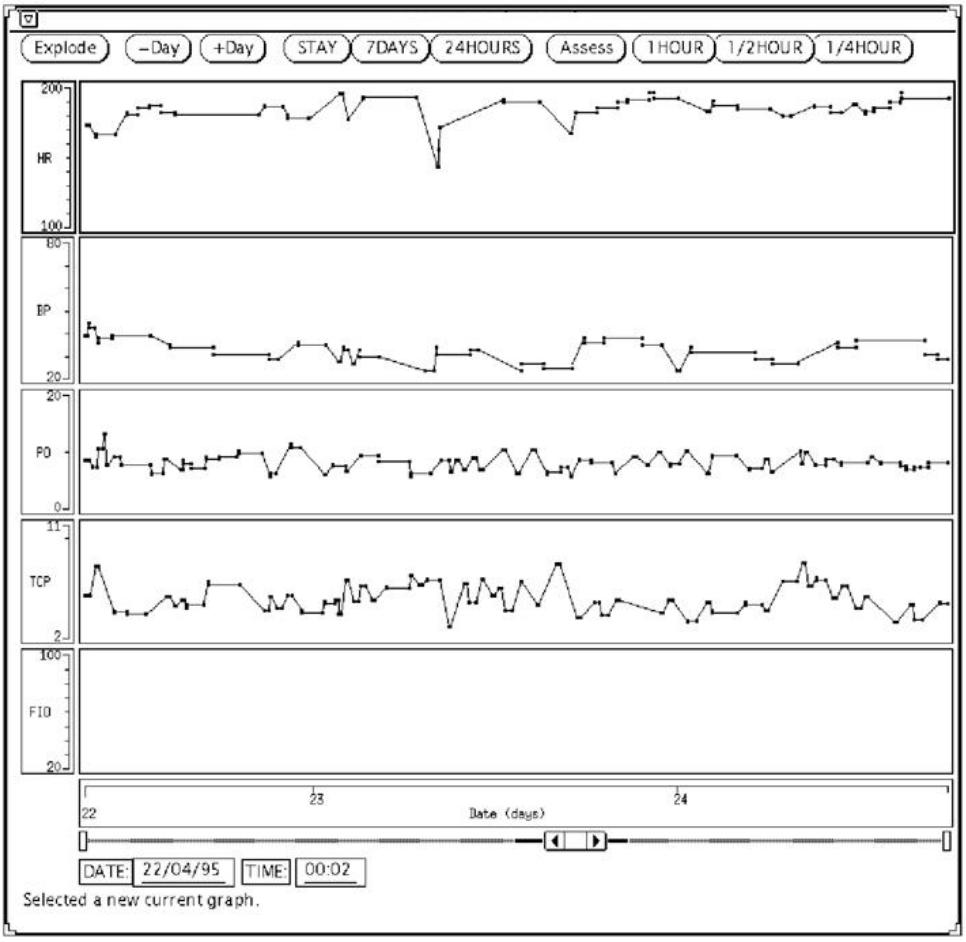


Figure 9. Algorithm for filter data module

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 n^{th} 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

Knox, S., Coyle, L., & Dobson, S. (2010). Using ontologies in case-based activity recognition. In *Proceedings of 23rd Florida Artificial Intelligence Research Society Conference*. St. Pete, FL: AIRSC.

Portet, F., Reiter, E., Gatt, A., Hunter, J. R. W., Sripada, S., Freer, Y., & Sykes, C. (2009). Automatic generation of textual summaries from neonatal intensive care data. *Artificial Intelligence*, *173*, 789–816. doi:10.1016/j.artint.2008.12.002.

Salatian, A. (2003). Interpreting historical ICU data using associational and temporal reasoning. In *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence*. Sacramento, CA: IEEE.

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

Salatian, A., & Hunter, J. R. W. (1999). Deriving trends in historical and real-time continuously sampled medical data. *Journal of Intelligent Information Systems*, *13*, 47–74. doi:10.1023/A:1008706905683.

Salatian, A., & Oriogun, P. (2011b). A software architecture for summarising and interpreting ICU monitor data. *International Journal of Software Engineering*, *4*(1), 3–14.

Salatian, A., & Taylor, B. (2004). An agglomerative approach to creating models of building monitoring data. In *Proceedings of 8th IASTED International Conference on Artificial Intelligence and Soft Computing*. Marbella, Spain: IASTED.

Salatian, A., & Taylor, B. (2008). ABSTRACTOR: An agglomerative approach to interpreting building monitoring data. *Journal of Information Technology in Construction*, *13*, 193–211.

Salatian, A., & Taylor, B. (2011). ABSTRACTOR: An expert system for fault detection in buildings. In *Proceedings of 1st International Conference on Intelligent Systems & Data Processing*. Vallabh Vidya Nagar, India: IEEE.

Shaw, M., & Garlan, D. (1996). *Software architecture: Perspectives on an emerging discipline*. Englewood Cliffs, NJ: Prentice Hall.

Sripada, S., Reiter, E., & Davy, I. (2003). SumTime-mousam: Configurable marine weather forecast generator. *Expert Update*, *6*(3), 4–10.

Sun, J.-T., Shen, D., Zeng, H.-J., Yang, Q., Lu, Y., & Chen, Z. (2005). Web-page summarization using clickthrough data. In *Proceedings of 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM.

Turner, R., Sripada, S., Reiter, E., & Davy, I. (2008). Using spatial reference frames to generate grounded textual summaries of georeferenced data. In *Proceedings of 5th International Natural Language Generation Conference*. Salt Fork, OH: IEEE.

ADDITIONAL READING

Anthony, T., Babar, M. A., Gorton, I., & Han, J. (2006). A survey of architecture design rationale. *Journal of Systems and Software*, *79*(12), 1792–1804. doi:10.1016/j.jss.2006.04.029.

Anuj, S., Singhal, M., Gibson, T., Sivaramakrishnan, C., Waters, K., & Gorton, I. (2008). An extensible, scalable architecture for managing bioinformatics data and analyses. In *Proceedings of IEEE Fourth International Conference on eScience '08*. Indianapolis, IN: IEEE.

Babar, M. A., & Gorton, I. (2009). Software architecture reviews: The state of the practice. *IEEE Computer*, *42*(7), 26–32. doi:10.1109/MC.2009.233.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Reading, MA: Addison-Wesley Professional.

Bosch, J. (2000). *Design and use of software architecture adopting and evolving a product-line approach*. Reading, MA: Addison-Wesley Professional.

Bosch, J. (2004). Software architecture: The next step. *Lecture Notes in Computer Science*, *3047*, 194–199. doi:10.1007/978-3-540-24769-2_14.

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *Pattern-oriented software architecture: On patterns and pattern languages*. New York: John Wiley and Sons.

Buschmann, F., Meunier, R., Rohnert, H., & Sommerlad, P. (1996). Pattern-oriented software architecture: *Vol. 1. A system of patterns*. New York: Wiley.

Clements, P., Garlan, D., Bass, L., Stafford, J., Ivers, J., & Little, R. (2002). *Documenting software architectures: Views and beyond*. Upper Saddle River, NJ: Pearson Education.

Clements, P., Garlan, D., Little, R., Nord, R., & Stafford, J. (2003). Documenting software architectures: Views and beyond. In *Proceedings of 25th International Conference on Software Engineering*. Portland, OR: IEEE.

Clements, P., Kazman, R., & Klein, M. (2001). *Evaluating software architectures: Methods and case studies*. Reading, MA: Addison-Wesley Professional.

Dobrica, L., & Niemelä, E. (2002). A survey on sftware architecture analysis methods. *IEEE Transactions on Software Engineering*, *29*(7), 638–653. doi:10.1109/TSE.2002.1019479.

Eeles, P., & Cripps, P. (2009). *The process of software architecting*. Reading, MA: Addison-Wesley Professional.

Fairbanks, G. H. (2010). *Just enough software architecture: A risk-driven approach*. New York: Marshall & Brainerd.

Gorton, I. (2008). Software architecture challenges for data intensive computing. In *Proceedings of 7th Working IEEE / IFIP Conference on Software Architecture*. Vancouver, Canada: IEEE.

Gorton, I. (2011). *Essential software architecture*. Berlin: Springer-Verlag. doi:10.1007/978-3-642-19176-3.

Gorton, I., Cuesta, C. E., & Babar, M. A. (Eds.). (2010). *Proceedings of software architecture, 4th European conference, ECSA 2010.* Copenhagen, Denmark: ECSA.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., & America, P. (2005). Generalizing a model of software architecture design from five industrial approaches. In *Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture*. Pittsburgh, PA: IEEE.

Hofmeister, C., Nord, R. L., & Soni, D. (1999). *Applied software architecture*. Reading, MA: Addison Wesley.

Jansen, A., & Bosch, J. (2005). Software architecture as a set of architectural design decisions. In *Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture.* Pittsburgh, PA: IEEE.

Kamal, A. W., & Avgeriou, P. (2010). Mining relationships between the participants of architectural patterns. In *Proceedings of 4th European Conference on Software.* Copenhagen, Denmark: IEEE.

Knodel, J., Lindvall, M., & Muthig, D. (2005). Static evaluation of software architectures-A short summary. In *Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture.* Pittsburgh, PA: IEEE.

Qian, K., Fu, X., Tao, L., & Xu, C. W. (2009). *Software architecture and design illuminated.* New York: Jones and Bartlett Publishers.

Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2000). Pattern-oriented software architecture: *Vol. 2*. *Patterns for concurrent and networked objects*. New York: Wiley.

Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2009). *Software architecture: Foundations, theory, and practice*. New York: Wiley.

KEY TERMS AND DEFINITIONS

Abstraction: This is the process of identifying features such as trends in the data.

Filter: This is the process identifying and retaining or removing outliers, inconsistencies or noise from the data.

Interpretation: An explanation of the data.

Pattern Mining: This is the process of finding or matching systems to a particular architecture or framework.

Software Architecture: This is the structure of a system, which comprises software elements and the relationships among them.

Temporal Inferencing: The process of using rules to merge consecutive intervals into larger intervals.

Temporal Interpolation: The process of creating an interval between 2 consecutive points.