Towards an Integrated Approach to Monitor and Analyse Health Care Data using Relational Databases

Philip Schmiegelt[1], Jingquan Xie[2], Gereon Schüller[1] and Andreas Behrend[3]
[1] *Fraunhofer FKIE, Fraunhoferstr. 20, 53343 Wachtberg, Germany*
[2] *Fraunhofer IAIS, Schloss Birlinghoven, 53754 Sankt Augustin, Germany*
[3] *University of Bonn, Institute of CS, Römerstr. 164, 53117 Bonn, Germany*

Keywords: Patient Monitoring, Phase Analysis, Declarative Programming, Domain-specific Architecture, Data Management, Decision Support Systems.

Abstract: In modern patient monitoring systems a tremendous amount of data is gathered, stored, and analysed to support doctors in making important decisions in a timely manner. To this end, different types of data from different sources have to be processed such as sensor readings of patients vitals, meta-data like the age and weight of a patient, and historical data like performed treatments or therapies. Most of the data is low-level and has an intrinsically temporal nature which need to be preprocessed for doctors to find high-level information in an efficient way. In monitoring scenarios however, aside from the detection of critical situations of patients, medics are often interested in the phases in which their patients are most probably in. In this paper, we show how phase analysis can considerably reduce the syntactic complexity of continuous queries as provided by the Continuous Query Language (CQL). Such phases provide an advanced and higher level of abstraction enabling effective and intuitive formulation of queries comparing to classic CQL. This can greatly improve the development efficiency and reduce the maintenance complexity of patient monitoring system.

1 INTRODUCTION

In the clinical context, especially at the intensive care unit, various sensor data are continuously collected, e.g. heartbeat, temperature, blood pressure, oxygen saturation, and more. These values have to be carefully supervised and analysed by doctors for correct diagnosis. They also indicate whether further examinations have to be carried out to diagnose or exclude a particular illness. On the other side, due to the technological advances, sensor types and their data rate and precision have greatly increased in recent years. The amount of data a doctor has to observe becomes overwhelming. Thus decision support systems which can *effectively* and *efficiently* preprocess low-level data to provide high-level information and even useful knowledge to doctors have been gaining much attention.

A first attempt to assist doctors was the MYCIN (Shortliffe, 1976) system, developed in 1972. It was, however, not ready to be fully integrated into daily hospital routine. MYCIN was designed as a decision support system. It generates a list of the most possible illnesses and proposes appropriate therapies based on several simple (mostly yes/no) questions. Nowadays, the (automatic) capturing of various patient data as EMRs (Electronic Medical Records) has been widely used. The potential of applying EMRs for automatic diagnosis and therapy support is however far from being exhausted. One important reason is that the employed database systems provide only limited analysis functionality for data streams. In particular, SQL lacks a comprehensive support for the temporal aspect, which is one of the most important requirements in the clinical context. For example, the orders of certain symptoms or the occurrences of certain vital recordings within a given time interval are relevant for reaching a "correct" diagnosis. Besides of that doctors usually want to be informed about phases *with temporal constraints* such as "patient has higher temperature for more than three minutes". Because of the streaming nature of EMR-based systems, real-time analysis is the prerequisite for time-critical continuous queries. In fact, queries for (fast) changing vitals or laboratory data have to be timely re-evaluated based on the current and the historical medical records.

For practical applications like (Schüller et al.,

Schmiegelt P., Xie J., Schüller G. and Behrend A..
Towards an Integrated Approach to Monitor and Analyse Health Care Data using Relational Databases.
DOI: 10.5220/0004326204070412
In *Proceedings of the International Conference on Health Informatics* (HEALTHINF-2013), pages 407-412
ISBN: 978-989-8565-37-2

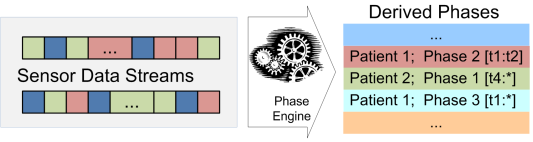

Figure 1: From Events to Phases.

2010), a huge number of continuous queries have to be evaluated on top of a highly dynamic data volume. This leads to a considerable amount of changes or updates to the derived information. Doctors, on the other hand, cannot continuously observe the entire list of dynamic query results. Instead, they are more interested in seeing the latest changes of the most relevant phases the patients are currently in. To this end, observed data and detected phases have to be classified using some domain-specific labels (e.g. normal, guarded, serious, or critical). Labelled phases provide a high-level means to support doctors to find the most relevant information more efficiently. If a patient is in a critical phase, various vital signs like blood pressure, heart rate, temperature and oxygenation etc. may exceed pre-defined threshold values at the same time. For a doctor however, it is usually clear from the current phase, which symptom has to be addressed at first by an appropriate treatment. A labelling of phase transitions (e.g. getting better, worsening, etc.) would be helpful for characterising the change of the patient's condition over time. For instance, doctors usually want to know whether a patient is deteriorating overtime or how fast the patient's vitals have deteriorated from a *guarded* to a *critical* phase. The way from events to phases, as shown in Fig. 1, has already been outlined in the KIDS model as a kind of high-level information (Liu et al., 2012). The sensor data streams on the left are analysed continuously by the phase engine. If a pre-defined significant change about patients situations is detected, a new phase will be derived and the current phase will be terminated.

2 MOTIVATING EXAMPLE

In this section, we give an example about the determination of the risk of a cardiac arrest and the patient's state of *Hemodynamic Instability*. An indication for a possible cardiac arrest is that the heart rate is greater than 120 for more than one hour and in the meantime the blood pressure is higher than 100 (Guerra et al., 2011). This could be specified in standard SQL as follows:

```
CREATE VIEW Cardiac_Arrest_Sign1 AS
SELECT patientId, heartRate,
       bloodPressure, timepoint
FROM   patientData pd
WHERE  bloodPressure > 100
       AND heartRate > 120
       AND NOT EXISTS (
       SELECT * FROM patientData
       WHERE patientID = pd.patientID AND
         bloodPressure <= 120 AND
         timestamp > NOW() - 1 hour)
```

If the blood pressure decreases by 20% to a value below 60 mmHg within a six-hour window and the heart rate increases by more than 50%, a *Hemodynamic Instability* might be entered by a patient (Lehman et al., 2007). We could formally define this using the following SQL statement:

```
CREATE VIEW Hemodynamic_Instability AS
SELECT patientId, heartRate,
  bloodPressure, timepoint
FROM   patientData pd
WHERE  bloodPressure < 0.8*(
  SELECT max(bloodPressure)
  FROM patientData
  WHERE patientId = pd.patientID
    AND timestamp > NOW() - 6 hours )
  AND heartRate > 1.5*(
    SELECT min(heartRate)
    FROM patientData
    WHERE patientId = pd.patientID
     AND timestamp > NOW() - 6 hours )
```

Due to the declarative nature of SQL, the resulting queries remain quite simple and readable. In the stream context, however, the risk of a cardiac arrest is redundantly derived by the stream engine. The stream processing system does not consider the current state of a patient. The doctor, on the other hand, usually wants to be alerted on the first occurrence of this condition. After that, the doctor should immediately start with an appropriate therapy and ought to be informed as soon as the risk for a cardiac arrest ends. The continuous notification of the same condition however should be avoided and the determination of the patient's phase (no notifications for the same phase) would be much more appropriate.

Another problem arises if we want to find out situations where an *Hemodynamic Instability* is immediately followed by a *Cardiac Arrest*. The timestamps encoded in the timepoint attribute have to be compared to determine whether the two critical situations follow each other. The complexity of these queries increases dramatically as soon as more temporal relationships have to be considered.

Note that there are many conditions which probably indicate a cardiac arrest. The query above just represents a particular constellation. If there are several database views that contribute to the detection of a possible cardiac arrest (such as Sign2, Sign3, etc.),

it is necessary to rank the corresponding derived indications based on their priorities. This priority depends on the degree of urgency with which a critical condition has to be treated. Therefore, we propose to employ a priority value for each phase allowing to order phases according to their relative importance. In addition, provenance is a key functionality in this scenario meaning that all derivations and their interconnections are stored (see Section 4).

3 PHASES

The concept of phases has been formally defined in (Schüller et al., 2012). In this section, we will briefly recall the core ideas of phases and more details about the formal definitions and semantics can be found there.

3.1 Semantics

A phase is used to describe the general, abstract state a patient has in a certain interval of time, e.g. "having fever". This interval does not necessarily have a fixed end, instead its end can be continuously evaluated, e.g. the "fever" phase ends when a normal temperature has been observed. Formally, a phase p is defined as a n-tuple $p = \langle o, n, b, e, a_1, \ldots, a_n \rangle$, with

$$\begin{array}{ll} o & \text{the object to which the phase belongs} \\ n & \text{the name of the phase} \\ b & \text{the time of begin of the phase} \\ e & \text{the time of end of the phase} \\ a_i & \text{phase-related attributes.} \end{array} \quad (1)$$

where $b < e$. The a_i are attributes attached to a phase. In case of "fever" this could be the temperature of the patient.

With this definition, the following queries become easily possible:

- Which phases did/does a patient have?
- Which patient was/is in a certain phase?
- At which point in time did/does a certain phase of a patient begin or end?

3.2 Phase Properties

Transitions. One of the key functionalities is the definition of possible *transitions* between two phases. This enables the treatment of all other transitions as "forbidden" indicating that these transitions are abnormal and should not happen. For example the changing from the tachycardia phase to the bradycardia phase should not be allowed. Instead, if the sensor readings of a patient indicate that such a phase transition is occurred, an alarm should be triggered. All observed phases and phase transitions form a phase transition graph.

Exclusivity. Phases can be either *exclusive* or *non-exclusive*. For example, a patient can either be in the state "Bradycardia" or "Tachycardia". It is not possible to have both phases at the same time. However, a patient can have a phase "Bradycardia" along with a phase "Fever", as these two phases are not exclusive. This leads to the following formal definition:

Exclusivity. *Two types of phases $\mathbb{P}_1, \mathbb{P}_2$ are called exclusive if and only if the condition*

$$\begin{aligned} \forall o : \neg \exists p_i, \in \mathbb{P}_1, p_j \in \mathbb{P}_2 : \\ (p_j.b \leq p_i.b \wedge p_i.e \leq p_j.e) \vee \\ (p_i.b < p_j.b \wedge p_i.e > p_j.e) \end{aligned} \quad (2)$$

holds.

Ranking of Phases. Another very important concept is the possibility to *rank phases*, e.g. based on their importance. If a patient is in the phase "Fever" and "Hemodynamic Instability", the Fever has low importance. Of course, the assessment can change dynamically. One of the attributes assigned to each phase can be used to store a numerical value, representing its importance. It is important to note that the importance of a phase cannot be evaluated itself, instead all other currently active phases have to be taken into account. If, for example, Fever and Tachycardia are observed at the same time, the Tachycardia might have a higher importance. If, however, signs of a virus infection arise, the importance of the Fever might rise above the importance of the Tachycardia. The assessment of a state also depends on other phase attributes . A Fever phase with a temperature of 38 degrees Celsius is of little importance, whereas a temperature of 42.3 degrees Celsius indicates a very critical situation resulting an higher importance.

Non-events. Another problem difficult to express with standard SQL is the *non-occurrence of events* within a given time interval. Consider e.g. the application of an antipyretic drug, where the temperature is expected to decline over a certain period of time. If the desired effect does not occur (non-event), then appropriate measures have to be taken. An automatic mechanism to support the detection is especially useful in the clinical context, where doctors work in shifts and the reaction to a medication needs not be visible during a single shift.

3.3 Functions on Phases

At this point, it is useful to define some *functions on phases*. They will serve for a later discussion on querying phases.

isInPhase. This boolean function returns *true* if and only if the object o is within a phase named n at a certain point in time t:

$$\text{isInPhase}(o,n,t) = \begin{cases} \text{true}, & \text{if } \exists p \in P: \\ & p.b \leq t \leq p.e \wedge \\ & p.o = o \wedge p.n = n \\ \text{false}, & \text{else} \end{cases} \quad (3)$$

Duration. The duration of a phase p is given by

$$duration(p) = p.e - p.b \quad (4)$$

A *temporal order* for exclusive phases p_1 and p_2 on the same object o can be defined by:

$$p_1 \leq_t p_2 \Leftrightarrow p_1.b \leq p_2.b \; (\Leftrightarrow_{\text{def 3.2}} p_1.e \leq p_2.e) \quad (5)$$

Sequence. This boolean function returns *true* if and only if two phases were active on an object in temporal sequence:

$$\text{sequence}(o, p_1, p_2, \ldots, p_n) = \begin{cases} \text{true}, & \text{if } p_1 \leq_t p_2 \\ & \leq_t \ldots \leq_t p_n \\ \text{false}, & \text{else} \end{cases} \quad (6)$$

The strict sequence can be used to ensures that no third (or more) phase was active between two phases:

$$\text{strictSequence}(o, p_1, p_2, \ldots, p_n) = \begin{cases} \text{true}, & \text{if } p_1 \leq_t p_2 \leq_t \ldots \leq_t p_n \\ & \wedge \nexists p_j \notin \{p_i, p_{i+1}\}, i \in \mathbb{N} : 1 \leq i < n : \\ & p_i \leq_t p_j \leq_t p_{i+1} \\ \text{false}, & \text{else} \end{cases} \quad (7)$$

Of special interest are methods which allow for an advanced pattern matching, possibly on unlimited regular expressions. A detailed discussion is, however, out of the scope of this paper. The interested readers can find deeper insight in (Cadonna et al., 2011). The functions defined there can be applied analogously for the handling of phases.

Previous/Next. Both functions return the previous and the next phases that are recorded in the history for a given object at a certain point in time:

$$\text{prev}(o,t) = \max(\{P | p.e < t \wedge p.o = o\}) \quad (8)$$

with the max function and the temporal order defined in (5). Obviously previous and next can only be used on exclusive functions. The next function can be defined analogously:

$$\text{next}(o,t) = \min(\{P | p.b > t \wedge p.o = o\}) \quad (9)$$

4 ADVANTAGES OF PHASES

Provenance. In all places where difficult, possibly life-threatening decisions have to be made, provenance is of a high importance. The concept of phases can help to identify the chain of causes and effects which lead to a phase transition. Simply storing all sensor readings, medications, or meta-data is not sufficient. Due to the vast amount of data, accumulating even over a short period of time, a comprehensive analysis becomes not feasible. Instead, by the application of phases the amount of data to search and to interpret can be dramatically reduced. Therefore the higher-level analysis becomes more applicable and scalable. This is where common Complex Event Processing (CEP) (Luckham, 2002) solutions fall short. They are designed to have a very high throughput, but in the clinical context the delay of writing data persistently to disks is too high. On the other hand, storing all sensor readings, as favoured by systems based on relational database, is also not a good solution: as a single sensor reading is not interesting any more after a few minutes. This is where our concept of phases can be used: filtering the whole data stream, discarding the low-level sensor readings while keeping the relevant "phases" and their transitions persistent.

Consider the example of a patient suffering from a hemodynamic instability as a consequence of stenting. Looking at the raw sensor data will not give any clues why the stenting was done. However, if a phase of "artery stenosis" is stored, provenances can be easily established on based on this high level information.

Historic patient data and their relationships or causality can be analysed in an easier way. Some illnesses, such as various kinds of flu encountered in the past years, are unpredictable and usually only occur within a short period of time. When a new mutation of the virus arises, the symptoms might be wrongly interpreted due to the lack of knowledge about the new mutation. As this knowledge evolves over time, it becomes interesting to re-read the data with a "what-if" analysis, e.g. *what* would have changed *if* we had known the virus before the first outbreak.

Prediction. It is of great value to have a better overview about the current situations of a patient. A

prediction about the future trends about a patient's phase is in many cases more interesting for doctors. Phases especially phase transition graphs provide a foundation for a high-level prediction. While it is difficult, if not impossible, to forecast sensor readings, the prediction (with certain probability) of the next phase is possible. This can be achieved based on the analysis of the phase transition graph. In most cases, it is even sufficient to predict a general trend as a forecast, e.g. will the fever become lower or higher in the next hours, or will the heart rate continue to decrease or stay at a low level.

Focusing on Important Values. For applications where software products are used as decision support systems, the effective displaying of the most prominent parameters plays a central role. At the layer of Graphical User Interface (GUI), information which should appear on the screen is ultimately decided. This is sometimes not a good choice. Visual appearances are mostly hard-coded within the application and are not flexible enough to be adjusted directly by the user at runtime. Having the abstract concept of phases available solves this problem. For example, when a doctor is called to come to a patient's bed, phases with the highest importance are shown at first. The attributes belonging to those phases have to be displayed, while all others, even if they might be abnormal, can be ignored at first. For example, in the case of a *Hemodynamic Instability*, these attributes are heart rate and blood pressure. All other sensor readings, like oxygenation, can also be viewed on request, but the most important values have to be highlighted automatically. Transferring this knowledge from the GUI to the underlying system is a great advantage, since it can be adapted in a more intuitive and less error-prone way.

5 MOTIVATING EXAMPLE REVISITED

After having outlined the definitions and advantages of phases in the clinical context, we come back to the example given in the beginning of this paper. The solution is provided by using phases instead of the classical SQL.

A sample phase tuple is depicted in Figure 2. *s_ID* is the id of the given phase. It must be unique. *p_ID* is the id of a patient which is denoted as an object in the phase definition. A list of attributes *attr1* to *attrN* is modelled as payloads providing extra context information for the given phase. The attributes *start*

Figure 2: The tuple structure of a typical phase.

and *end* are timestamps denoting the temporal constraints. The start timestamp must always be given while the end is optional, i.e. the end timestamp can be "undefined". It is only defined when a new phase to the same object has been detected.

The detection of the "hemodynamic instability" which might be followed by a cardiac arrest can be rewritten now as:

```
SELECT patientId,
FROM patientData
WHERE sequence(
  HeartPhase.name = 'hemodynamic instability',
  HeartPhase.name = 'cardiac arrest sign 1')
```

It is clear that in the sample code above, no timestamps are given explicitly. The temporal relationships which are important for clinical context are embedded within the phase definition itself.

The next sample code shows alarms for patients whose condition gets worse over time. It is assumed that the importance is a comparable value with partial or full ordering, e.g. a natural number. To ensure that only patients in a certain state are returned, the `isInPhase`-function is used.

```
SELECT patientID, 'deteriorating' AS
FROM patientData
WHERE
isInPhase(patientID, PatPhase2, NOW)
  AND strictSequence(PatPhase1,PatiPhase2)
  AND PatPhase1.importance <
      PatPhase2.importance
```

As shown above, after the application of phases the complexity of queries can be reduced quite a few. For a realistic application with hundreds of or thousands of queries, the correctness, robustness and maintainability of the whole system can be greatly improved.

6 RELATED WORK

There are some extensions of SQL which provide the syntactic extensions to handle events. One of them is SARI-SQL (Rozsnyai et al., 2009), which introduces events with a time interval, where start and end timestamps can be queried separately. TSQL2 (Snodgrass, 1995) introduces the concept of states, it does however, differ from the approach proposed in this paper: in TSQL2 neither identifiers for states are provided nor methods on the transitions between states are introduced. As phases are started and ended by events,

they are comparable to regular expressions on these events. Pattern-based event matching has been discussed in (Cadonna et al., 2011). Likewise, modern stream processing engines are able to handle a large number of events. However, these events usually only have a small duration (Abadi et al., 2003; Chen et al., 2000) or a fixed time interval (Krämer and Seeger, 2004). In contrast, phases are typically long lasting and the end point of a phase maybe *undefined* until other phases become active.

Phases play a prominent role in modern health care monitoring system (Gawlick et al., 2011). The assessment of phases and phase transitions are of high importance for doctors. While in the clinical context this kind of abstract high-level information is usually described by the word "state" or "status", we choose to use the term "phase" instead. The reason is that the term *state* is extensively used in computer science, having various defined semantics in different contexts. To avoid any confusion, a overloaded usage should be avoided. Furthermore, the concept of phases have been used, e.g. in airspace monitoring (Schüller et al., 2010) and has also been discussed in (Schüller et al., 2012) in more details.

Phases can be interpreted as certain event patterns with extended time interval specifications. The matching of such pattern without time interval has been already discussed in some literatures (Cadonna et al., 2011). Algorithms for efficiently detecting different event patterns could also be used for phase detection. For CEP engines, events usually have only a single time stamp without a duration (Abadi et al., 2003; Chen et al., 2000). There are however approaches where each event has a fixed interval in which it is considered to be valid (Krämer and Seeger, 2004). In contrast to phases this kind of interval has to be determined at detection time of the respective event.

7 CONCLUSIONS

In this paper we introduced the concept of phases in the medical context. The application of the phase concept has numerous advantages over the plain SQL statements. It still has to be fully investigated, implemented and evaluated to unleash its full potential. A comprehensive discussion on the syntax and semantics of phases still needs to be done. The underlying phase processing engine is to be developed using for example database stored procedures or other high-level programming languages. The concept of phases is motivated within the medical domain but it can also be useful for other monitoring scenarios.

REFERENCES

Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. (2003). AURORA: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139.

Cadonna, B., Gamper, J., and Böhlen, M. H. (2011). Sequenced Event Set Pattern Matching. pages 33–44, New York, NY, USA. ACM.

Chen, J., DeWitt, D. J., Tian, F., and Wang, Y. (2000). NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *SIGMOD*, pages 379–390, New York, NY, USA. ACM.

Gawlick, D., Ghoneimy, A., and Liu, Z. H. (2011). How to build a modern patient care application. In *HEALTHINF*, pages 427–432.

Guerra, D., Gawlick, U., Bizarro, P., and Gawlick, D. (2011). An Integrated Data Management Approach to Manage Health Care Data. In *BTW*, pages 596–605.

Krämer, J. and Seeger, B. (2004). PIPES - A Public Infrastructure for Processing and Exploring Streams. In *SIGMOD*, pages 925–926.

Lehman, L., Kyaw, T., Clifford, G., and Mark, R. (2007). A Temporal Search Engine for a Massive Multi-Parameter Clinical Information Database. In *Computers in Cardiology, 2007*, pages 637 –640.

Liu, Z. H., Behrend, A., Chan, E., Gawlick, D., and Ghoneimy, A. (2012). Kids - a model for developing evolutionary database applications. In *DATA*, pages 129–134.

Luckham, D. (2002). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 1st edition.

Rozsnyai, S., Schiefer, J., and Roth, H. (2009). SARI-SQL: Event Query Language for Event Analysis. In *CEC*, pages 24–32.

Schüller, G., Behrend, A., and Manthey, R. (2010). AIMS: An SQL-based System for Airspace Monitoring. In *IWGS*, pages 31–38, New York, NY, USA. ACM.

Schüller, G., Schmiegelt, P., and Behrend, A. (2012). Supporting Phase Management in Stream Applications. In *ADBIS*, pages 332–345.

Shortliffe, E. H. (1976). Computer-Based Medical Consultations: MYCIN. *Annals of Internal Medicine*, 85(6):831–831.

Snodgrass, R. T., editor (1995). *The TSQL2 Temporal Query Language*. Kluwer.