Database Functionalities for Evolving Monitoring Applications

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: Monitoring, Data Streams, Event Processing, Temporal Databases, Provenance, Knowledge Management, Declarative Programming.

Abstract: Databases are able to store, manage, and retrieve large amounts and a broad variety of data. However, the task of understanding and reacting to the data is often left to tools or user applications outside the database. As a consequence, monitoring applications are often relying on problem-specific imperative code for data analysis, scattering the application logic. This usually leads to island solutions which are hard to maintain, give raise to security and performance problems due to the separation of data storage and analysis. In this paper, we identify missing database functionalities which overcome these problems by allowing data processing on a higher level of abstraction. Such functionalities would allow to employ a database system even for the complex analysis tasks required in evolving monitoring scenarios.

1 INTRODUCTION

Database applications enable users to deal with the ever increasing amount and complexity of data and knowledge. However, the process of problem solving, which requires understanding and tracking the current status and evolution of data, knowledge, and events, is still handled mostly by humans and not by databases and their applications (Wieringa, 2003). Therefore, the KIDS database model has been proposed as a blueprint to extend database technologies to manage data, knowledge, directives (processes), and events in a coherent way (Liu et al., 2012; Chan et al., 2012). The acronym KIDS stands for the most important elements of this model by means of Knowledge, Information, factual Data, Directives and Social interactions. KIDS distinguishes among three classes of data (facts, information, and directives) and three classes of knowledge (classification, assessment, and enactment). Solving problems entails the capturing and the reduction of emerging and historical facts into information by applying classification knowledge. Then such information is used to assess the situation and prescribe/describe the directives for dealing with the situation. Finally, the directives have to be executed by applying enactment knowledge. As directives are enacted, newly emerging facts will again be captured and classified; this determines whether a situation has been resolved or not.

As an example, consider a health care scenario where patient's data are continuously captured as EMRs (Electronic Medical Records). The review and the interpretation of medical data is becoming increasingly time consuming and controversial. Therefore, modern patient care applications have to provide significant help to handle such challenge; i.e., doctors need a system that transforms EMRs into compact information, applying the codified medical knowledge, and providing the most likely interpretations and their probabilities. This must be done on demand as well as proactively in real time to alert doctors and nurses about adverse and time critical situations. Once the doctor is alerted of the situation and supplied with the information summarizing the patient condition along with the relevant facts, s/he can assess the situation and decide on the course of action. The support should also help doctors selecting the most appropriate protocol of care, e.g. by indicating which medicine or combination of medicines has been successful with patients in a similar situation and also which tests are most advisable to reduce the level of uncertainty of the diagnosis. Once the orders are submitted, the system needs to help in the supervision and documentation of the execution. In

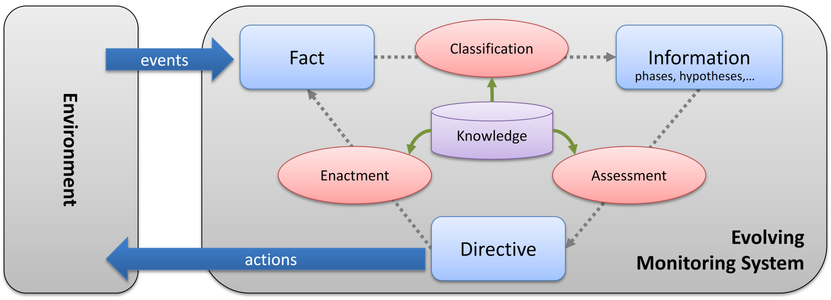

Figure 1: The overall architecture of complex reactive systems designed with the KIDS model inside.

essence, KIDS provides doctors comprehensive support in all phases of the treatment including: the capturing of the facts (EMRs), the extraction of information from these facts, the assessment of the relevant information, the determination of the course of action (directives - orders), and the enactment of such directives. A comprehensive support for managing such data processing is beneficial in various monitoring scenarios, e.g. air traffic control, network management or load balancing in cloud processing. In all these applications, KIDS allows for distinguishing the different forms of data processing steps and indicates the cyclic dependencies among them.

The KIDS model provides an innovative database-centric methodology to design large-scale knowledge-intensive applications in a systematic way. It is however still too abstract to be implemented into existing DBMS right away. In this paper, we therefore apply this model to analyze a practical patient-care use case and identify functional requirements needed for KIDS. As a result, we indicate the way how relational database technology should be further extended in order to be well-suited for realizing evolving monitoring applications.

The remainder of this paper is structured as follows: Section 2 gives a brief introduction about the KIDS model. In Section 3 a patient care use case is described and analyzed using the KIDS model. Functional requirements to extend DBMS to support KIDS are proposed in Section 4 with the concept of phases as a feasible approach in Section 5. Section 6 shows related work and Section 7 concludes this paper and includes future work.

2 KIDS MODEL REVISITED

Traditional relational databases are designed to store facts about the real world in an effective and efficient manner. Facts represent uninterpreted quantitative information about the world. However, facts are often only a small part of a larger data model, usually incorporated in the application logic where the raw facts have to be analyzed, conclusions drawn, and actions have to be initiated. This discrimination between database and application however, is not a natural one. It arises from the inherent inability of traditional database systems to store and process anything but factual data. This is where the KIDS model kicks in. It introduces the necessities to leverage a relational database to include more then just factual data.

The acronym KIDS represents the four most important concepts in this data model: Knowledge, Information, Data and Social interactions (Chan et al., 2012). The dependencies among the KIDS' concepts are illustrated in Figure 1 with an emphasis on the Fact-Information-Directive (FID) loop. The two top-level concepts in KIDS are *data* and *knowledge*. There are three different types of data: fact, information and directive which are represented in Figure 1 as blue filled rectangles. Facts are raw data like "temperature is 39°C". Information is the interpretation of facts, e.g. "temperature of 39°C possibly means fever". Directives are actions which need to be performed to check or affect the environment, e.g. "apply drug X to treat the fever of patient Y". As illustrated in Figure 1, knowledge is used to support three different processes: classification, assessment and enactment which are represented as red filled ellipses. The classification process utilizes knowledge to convert facts (raw data) into information (interpretations). Similarly, the assessment process generates actions based on the information and available knowledge. To close the loop, the enactment process tries to track the execution of directives and gather further related events from the environment.

3 USE CASE

In this section, a practical use case in the clinical context is presented which shows common workflow patterns for a patient monitoring system. This use case is typical for patients where the diagnosis is not obvious

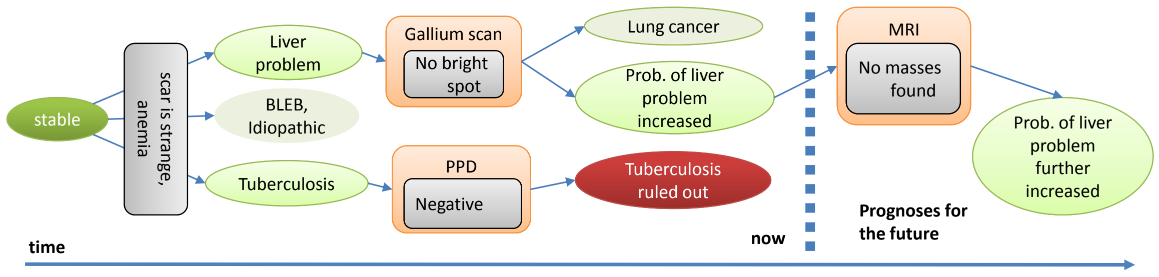

Figure 2: Work flow of the use case consisting of the past treatments and the future prognoses.

at first sight. Instead, different hypotheses with probabilities are made and corresponding tests have to be conducted to exclude or confirm certain hypotheses, or even generate new hypotheses. Often, a couple of iterations are needed to achieve the final diagnosis. During this process physicians have to consider various information about the patient (e.g. the entire treatment history, current status, etc.) and apply their professional knowledge to make the final decision.

Our use case is extracted from an episode of the TV series "Dr. House"[1]. A patient suffers from anemia and a scar looks unusual. The first hypotheses of the patients illness are either tuberculosis or BLEB (large blister filled with serous fluid, in this case in the lung) or an unknown liver problem. The tuberculosis is quickly ruled out by a PPD (purified protein derivative) test. On the contrary, the probability of the hypothesis about "unknown liver problem" is increased through a Gallium scan. Based on the experiences of physicians, the BLEB hypothesis remains with a low probability and is therefore not followed any more. To further strengthen the hypothesis about the unknown liver problem, an MRI is scheduled. The result of MRI is expected to be obtained in an hour and it will be used as the main evidence to reconsider the "unknown liver problem" hypothesis. A graphical summary of this part of the entire work flow is depicted in Figure 2.

Let us now apply the KIDS model to differentiate the different types of entities within the use case. As indicated by the KIDS model, a database should not only store factual data but incorporate a circular representation of facts, information, and directives. This FID loop can easily be found in this example.

At first, *factual data*, in this case that the patient is suffering from anemia and a scar is looking in a abnormal way, is observed. This qualitative data has to be entered into the database is a standardized way, enabling an automatic processing of the data. In the medical context, often sensor readings like temperature or blood pressure are gathered. These are of course easier to analyze in a database system, as they are simple numerical values with a fixed domain and well defined meanings.

These facts are then *classified*, that is hypotheses for matching diagnoses are searched in the system. In the medical context, this classification is done by doctors, not fully automatic. However, it is important that sufficient decision support is given to the medical personnel in order to take all facts into account. Here, three hypotheses are found: Tuberculosis, BLEB, and liver problem. Of course, in this step, medical *knowledge* about various diseases stored in the database is incorporated.

The resulting hypothesis are stored in the database as *information*. The key here is to have both all information of a patient safely stored and on the other hand be able to quickly present the most important pieces of information to a querying doctor. This could e.g. be the most severe illness the patient is suffering, the most abnormal sensor reading, or even, at a very abstract level, the current state (e.g. 'critical' or 'guarded'). In most scenarios, each piece of information will have a probability value attached to it. In the medical context, each diagnosis has a degree of uncertainty, which has to be reflected in the system.

These hypotheses are then presented to a doctor, who *assesses* this information. S/he is assisted by the database, which proposes tests or medications which have successfully been used before on patients in a similar state. In other scenarios, a fully automatic assessment will be feasible. In this use case, a PPD test for tuberculosis could be suggested. The decision is supported by the probability values for each of the hypothesis, and the information gain of the PPD test, as it has a yes/no result and therefore has a great influence on all of the hypothesis for this patient. To have such a decision support system alone is already valuable, since it could reduce costs and ensure a faster cure, because the right diagnosis can be given faster.

After that, *directives* to cure the illnesses or ensure a certain diagnosis are determined. These directives should also be stored in the database. In the use case, the doctor agrees with the automatic suggestion and decided that the tuberculosis hypothesis should

[1]http://house.wikia.com/wiki/Abigail_Ralphean

be followed and a test be made.

The effect of this *enactment* then deliveres new facts, which close the FID loop. Note that the enactment takes place in the real world, however the database has to store the back-link between new facts and a directive. This ensures that in a retrospective analysis the changing vitals can be associated with the initiating directive. In this use case, the test for tuberculosis was negative, thus the hypothesis can be deleted.

4 REQUIRED DBMS FUNCTIONALITIES

Ideally, the underlying database would automatically support doctors in all phases of the treatment including: the capturing of the facts (EMRs), the extraction of information from these facts, the assessment of the relevant information, the determination of the course of action, and the enactment of such directives. Obviously, a complex set of user-defined functions (UDFs), triggers, materialized SQL views, stream processing techniques, etc. could be used to model the desired behavior within a database. However, this would be very problem specific and not leverage one of the key concepts of a database: having an universal, declarative querying language. In particular, we need database support in order to query the entire chain of cause and effects, to propose suitable directives and disease hypotheses, and to automatically check the expected results for an issued directive. If the database would maintain auxiliary information about the chain of cause and effects, a corresponding query could be much easier formulated. Therefore, some extensions to existing database technologies are needed to fully provide all of the aspects the KIDS model offers in a user-friendly way.

Support Temporal Reasoning. Obviously, all entities of the KIDS model require a direct support of a time concept by the DMBS. Due to the many different ways facts are transferred from their observation to storing in the database, e.g. directly (a sensor with build-in network capabilites) or by manually entering data into the database (e.g. the interpretation of an MRI scan), it is necessary to support both application and system time (Snodgrass, 1995). It should be noted that time in this case is not a simple attribute which is added to each tuple. Instead, it is used as a basis to enable the interconnection of all elements within the database. This means that almost every operator has to be augmented to take the timing of the tuples processed into account.

Provenance. Especially in the medical context, provenance is of high importance. There are two factors which are of interest. The first one is to have the chain of cause and effect fully available within the database system. For example, it must be possible to determine the reason why a certain treatment (such as an MRI) has been performed. In our use case, the reason was to increase the confidence on the existence of a liver problem and this information has to be made queryable.

On the other hand, it must be possible to retrospectively investigate the knowledge that was present at a specific point in time. For example when a drug given to support the functioning of the liver leads to a sudden deterioration of the patient due to a infection with tuberculosis, it is important to have the knowledge at the time of applying the drug available. Having the concept of phases, a quick overview can be given, which shows that a liver problem was the most probable hypothesis of the patient's state, and that the tuberculosis hypothesis had been followed but was abandoned for sound reasons. This quickly shows that the deterioration of health for the patient was unforseeable.

Data provenance in databases is an active research area (Simmhan et al., 2005). Efficient, intuitive and scalable approaches to computing provenance in databases on a fine-grained level is still a challenging task (Karvounarakis et al., 2010). To our best knowledge, there is still no practical methods provided by commercial DBMS to efficiently support complex and fine-grained provenance. Application developers have to implement their own ad-hoc algorithms to deal with provenance in specific domains. In order to fully unleash the power of KIDS however, the built-in support of fine-grained provenance tracking with an intuitive interface is essential. It can provide a systematic and robust platform for complex reactive system designers and can significantly simplify the development cycle.

Evolving Knowledge Management. Knowledge plays a central role in the KIDS model as illustrated in Figure 1. Systematically representing knowledge in a computable form has a long research history, in particularly in Artificial Intelligence (Davis et al., 1993). It is however not the focus of KIDS to develop a new and innovative knowledge representation formalism suitable for DBMS. Existing approaches like inference rules with deductive reasoning have been successfully integrated into modern DBMS since a long time. The management of knowledge evolvement is however still pretty weak in contemporary DBMS. For example, it is still difficult to *semantically* query the whole evolving history of certain knowledge defined as views in today's DBMS.

Knowledge in complex reactive systems is normally changing dynamically. For example, the rules and experiences of physicians to make diagnoses are not fixed. They are evolving dynamically either through education, learning or social interactions. In modern patient monitoring systems, these kinds of knowledge are modeled by system developers through careful and thorough communication with physicians. This kind of "knowledge transfer" is rather complicated and the correctness can not be guaranteed. If the knowledge from physicians has evolved, the corresponding formal representations have to be synchronized. Existing solutions handle all these by themselves in application logics. For large systems it is very time-consuming and error-prone.

To support KIDS, existing DBMS should be extended to support sophisticated and efficient knowledge management and treat knowledge as a first-class citizen as data. This includes a declarative means to formally represent knowledge, an efficient mechanism to store and query knowledge, a scalable way to handle the evolving of knowledge. Besides of that the ability to manage personalized knowledge is essential since knowledge is an individual asset [2].

Classification. In this context, classification can both be a problem which can be solved by the database alone, or a human operator also has to add its knowledge and expertise. In automated systems, the classification of incoming facts, e.g. sensor readings, is a typical stream processing problem. Several solutions already exist, either as stand-alone software (e.g. Esper) or already integrate into the database management system (e.g. Oracle). That is, the basis for an automated classification exists, however for a fully functional implementation of the KIDS model it has to be tightly integrated into the DBMS. It must be possible to define the stream processing rules from within an SQL interface. As stated in the beginning of this paragraph, integration of external classification by e.g. doctors has to be processed as well. As example, the classification that a certain patient has tachycardia, derived from a stream of sensor readings, could be:

```
CREATE CONTINUOUS QUERY as
SELECT patientID, count(patientID) as c
FROM ICU_Stream
WHERE heart_rate>110
PARTITION BY patientID
RANGE 10 minutes
HAVING c > 10
```

[2]Though a set of common knowledge exists for different individuals, for complex reactive systems however the personalisation is important since the application of different knowledge can result in completely different interpretations of data.

A problem which is inherently difficult to model in current querying languages is the absence of certain facts or a sequence of facts. When e.g. the heart beat of a patient rises in a non-critical way, that event should not be reported. The cause could be a simple movement of the patient. However, if the heart rate does not return to its previous value after a short amount of time, a doctor should be notified to further investigate this abnormal behaviour.

Information. As illustrated in Figure 1, complex reactive systems are used to continuously monitor their environments and react to situations-of-interest (Wieringa, 2003). In general the reactive system does not know *exactly* what is happening in the environment. What the reactive system can do is trying to *approximate* the situations in the environment based on the sequence of captured events.

In KIDS the approximation of the environment is modelled as a set of hypotheses. Each hypothesis is associated with a probability. The size of the whole hypothesis space varies in different application domains. Contemporary DBMS should be extended to provide efficient and scalable built-in support for hypothesis management in a declarative manner. This is a challenging task and to our best knowledge existing DBMS still miss a systematic means to handle hypotheses in a declarative manner.

Decision support (Eom et al., 1998) on the other side is a fundamental component in knowledge-intensive reactive systems. It has been extensively applied in the clinical context (Kawamoto et al., 2005) to assist physicians to make diagnosis. Ideally, for a reactive system, most required knowledge and data are stored in a database as required by the KIDS model. This provides an excellent and feasible foundation to enable complex decision support in timely fashion. Different approaches and formalisms are introduced in (Eom et al., 1998) for decision support. In order to provide a full KIDS stack, DBMS should be extended to integrate them and provide a declarative interface to simplify decision support development in complex reactive systems.

All of these requirements can be dealt with by using phases. They comprise an abstract overview of complex situations, like an illness of a patient. An in depth discussion on the concept of phases is given in the next section (5).

Assessment. Like classification, assessment is rather a process which might take place outside of the database than a fact that could easily be stored as a relational tuple. However, this process might be supported by an automated analysis of the available information. This is supported by the phase concept. It allows a labeling of phases, e.g. ‘critical’, ‘guarded’ and ‘stable’ in the medical domain. This helps doctors to immediately identify which patient needs her/his attention most.

An important aspect here is that the labeling cannot be determined by viewing a single phase alone, it is always influenced by other phases which are active in parallel. A typical question of a doctor is, which of the patients is in the most severe condition and need attention next.

Directives. When directives are stored, they are expected to have a result after a certain period of time. It is therefore crucial that a time-based trigger (Behrend et al., 2009) or a DB job is started and the outcome is checked after its firing. In its most primitive form, a simple string along with a trigger expression could be used.

```
INSERT INTO
  patientDirectives(patientID, directive)
VALUES
  (id3, "PPD test for tuberculosis"@IN(3h))
```

This string would then be presented to the primary doctor after a three hour interval has elapsed, such that s/he can then assess whether the examination has taken place. A more sophisticated method would be to take the domain knowledge stored in the database into account.

It is important to note here that using such a mechanism does not only store the temporal relationship between entities. It also enables a causal relationship between the directive, its execution and the results. This is essential for fully implementing the KIDS model, establishing a causal chain between the items stored in the database.

Enactment. Enactment refers to the physical world, meaning that a directive is executed. From a database point of view, it is not obvious why this should be stored. In most cases, the database will not even be notified (and does not need to be notified). Also, the process can take a long time, e.g. a long running test for a certain type of bacteria. The database will, however, implicitly be notified when a directive has been carried out by new or changing facts, e.g. the result of a test or changing vitals of a patient. It is still very important to keep track of these enactments, to be able to provide a full provenance. If a change in the sensor readings is detected, a link to the directive has to be stored, to allow to retrospectively analyze the chain of cause and effect.

In general, this is a complex task, even for a human. There are many factors which can be the cause of e.g. a drop of the heart rate. However, it is impossible to model all of the aspects, therefore we will assume that each directive, e.g. applying a drug, has a limited number of effects. This domain knowledge has to be programmed into the database. Also, the knowledge is enhanced with a time window, in which the effect is expected to occur. This means that when a directive is deployed, several CEP queries have to be started to watch for the changes. Once an expected behavior is observed, it has to be stored as an enactment for further reference and the CEP query can be terminated. After the time has elapsed, the remaining queries can be terminated, and if none of them fired a doctor has to be notified that a directive did not have the desired effect, as explained above.

5 PHASE SUPPORT IN DETAIL

The concept of *phase* has been proposed in our previous work (Schüller et al., 2012; Schmiegelt et al., 2013). It provides a high-level and feasible database-centric approach to design complex monitoring systems (air traffic, patient, etc.). In this section the phase concept is further developed to support KIDS-like complex reactive system design with database technologies in mind.

5.1 Introduction of Phases

A *phase* is used to describe the general abstract state of an entity (an airplane, a patient, etc.) within a time interval. For example, if a person has fever starting from January 1st, 2000 and it lasts for one week, then this can be modelled as a phase denoting the status of the person during that period. This time interval does not necessarily have a fixed end. Its end can be continuously evaluated, e.g. the “fever” phase ends when a normal temperature has been observed, and until then the phase has the ending timepoint “unknown”. Formally, a phase p is defined as a tuple $p = \langle o, n, b, e, a_1, \ldots, a_n \rangle$, with

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

where $b < e$. The a_i are attributes attached to a phase. For example, in case of the “fever” phase, this could be the temperature of the patient. With this definition, the following questions can be answered:

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

5.2 Derivation of Phases

Phases are derived either directly from factual data or from other phases. Phases can be mapped in the KIDS model as the *information* and *hypothesis*. In Figure 2 phases are represented as filled ellipses. For example, "stable", "liver problem", "tuberculosis" etc. are all phases with certain probabilities. These phases are generated based on the observed symptoms of the patient. The probability for each phase should be computed automatically, based on the known facts and the entire set of (partially) matching hypotheses.

In realistic database-centric reactive systems, phases can be defined with the CREATE PHASE clause. For example, the phase "liver problem" in Figure 2 can be defined as follows:

```
CREATE PHASE liver_problem
SELECT patientId
FROM   PatientData
WHERE  symptom = 'scaris strange, anemia';
```

At runtime, the "liver problem" phase with probability 0.6 is derived automatically when a record is inserted into the database with the specified symptom. The probability, start and end time points are assigned automatically by DBMS. Subsequent symptom descriptions with the same value does not cause the system to derive new "liver problem" phase, but just change the end time point to the new "valid" time [3]. Similarly two other "liver problem" phases with different probabilities can be defined as follows:

```
CREATE PHASE liver_problem
SELECT patientId
FROM   PatientData pd, liver_problem lp
WHERE  lp.prob=0.6
       AND lp.patientId=pd.patientId
       AND pd.gallium_scan='No bright spot';
```

```
CREATE PHASE liver_problem
SELECT patientId
FROM   PatientData pd, liver_problem lp
WHERE  lp.prob=0.7
       AND lp.patientId=pd.patientId
       AND pd.MRI='No masses found';
```

All these three phases can be considered as hypotheses in KIDS. The provenance of the refinement of hypotheses in the use case introduced in section 3 can be queried as follows:

```
SELECT PROVENANCE OF liver_problem
WITH PROBABILITY 0.8
WHERE patientId=1;
```

This query returns the direct provenance of the phase, i.e. the MRI test with the result "No masses found". To query the whole provenance as a transitive closure, the "ALL" keyword can be used:

```
SELECT ALL PROVENANCE OF liver_problem
WITH PROBABILITY 0.8
WHERE patientId=1;
```

This will retrieve the whole evolving history as provenance for the given phase. Besides of that the phase definitions which have contributed to the evolving of phases are also returned. Since the phase definitions are mappings of domain knowledge which can change over the time, it leads to one of the core functionalities in the phase concept to support evolving knowledge management as discussed in section 4.

5.3 Version Control of Phase Definitions

The definition of phases represents the knowledge in the KIDS model for the classification, assessment, and enactment processes. Knowledge is not only a personalised asset but also intrinsically dynamic. This makes the provenance management in KIDS a challenging task since the knowledge elements in provenance can change.

In the phase concept the ability to store different versions of phase definition is supported as an internal mechanism. The *transaction time* of phase definitions is used to retrieve a specific version of the phase definition at a certain time point or during a given time period. For example, the following query can be used to retrieve the phase definition of "Fever" at January 1st, 2010:

```
SELECT PHASE DEFINITION OF Fever
WHERE time_contains('Jan 1st, 2010');
```

This query returns exactly one result or NULL. It is also possible to retrieve a set of different versions of phase definition during a time period:

```
SELECT PHASE DEFINITION OF Fever
WHERE time_between(
      'Jan 1st, 1999', 'Jan 1st, 2010');
```

Depends on the evolving history of the phase definition of "Fever", this query can return several different versions of the definition of "Fever". All these features are deeply embedded in the DBMS and can be accessed declaratively. Comparing to the ad-hoc solutions implemented in the application logics, this provides a more effective and robust approach to manage evolving knowledge.

[3]The transactional time can also be used depending on system requirements.

5.4 Phase Properties

Exclusivity. Phases can be either *exclusive* or *non-exclusive*. For example, a patient can either be in the phase "Bradycardia" or "Tachycardia". It is however not possible to have both phases at the same time. On the other side, a patient can have a phase "Bradycardia" along with a phase "Fever" since these two statuses for a patient can co-exist in real life. This leads to the following formal definition:

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

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

holds.

Phase Functions. Phase provides an essential set of functions to enable high-level temporal and functional phase management in a declarative manner. These functions serve as a fundamental basis for *bitemporal reasoning* and *fine-grained provenance generation* as discussed in section 4.

The boolean function **Is an entity in a phase** returns *true* if and only if the entity o is within a phase named n at a certain point in time t:

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

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

$$p_1 \leq_t p_2 \Leftrightarrow p_1.b \leq p_2.b \; (\Leftrightarrow_{\text{def } 2} p_1.e \leq p_2.e) \quad (4)$$

This allows to define the boolean function **Sequence**, which returns *true* if and only if two phases were active on an object in temporal sequence. The **Strict Sequence** can be used to ensures that no third (or more) phase was active between two phases.

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.

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

$$\text{prev}(o,t) = \max(\{P | p.e < t \land p.o = o\}) \quad (5)$$

$$\text{next}(o,t) = \min(\{P | p.b > t \land p.o = o\}) \quad (6)$$

with the temporal order defined in (4). Obviously previous and next can only be used on exclusive functions.

Transition Graph. One of the key functionalities in the phase concept is the ability to define possible *transitions* between two phases. All phase definitions and phase transitions form a *phase transition graph*. The transition graph implicitly enables the treatment of all other transitions which do not exist in the graph as "forbidden" transitions, i.e. they are abnormal and should not happen at runtime. For example, based on the experiences of physicians, normally the status changing of a patient from the "tachycardia" to the "bradycardia" should not happen. If the sensor readings of a patient indicate that such a phase transition has actually occurred, an alarm should be triggered to alert physicians that something abnormal is happening. This is a advantage, as it is usually much easier to specify which transitions are allowed, instead of trying to explicitly specify transitions that correspond to abnormal behavior. A provenance query can be issued after an alert to find out the reasons for this illegal phase transition.

Ranking of Phases. Another essential feature in the phase concept is the ability to *rank* phases based on, e.g. their relative importance given by domain experts. For instance, if a patient is in the phase "Fever" and "Hemodynamic Instability", then the Fever phase has lower rank based on the rules given by physicians. Of course, the rank can change dynamically. One of the attributes assigned to each phase can be used to store a numerical value, representing its importance. The assessment of a phase rank also depends on a phases' attributes. For example, a Fever phase with a temperature of 38 degrees Celsius is of little importance, whereas a temperature of 41.2 degrees Celsius indicates a very critical situation resulting an higher rank.

Non-occurrence of Events. Another problem which is 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 complex reactive system. For example, in patient monitoring systems, where physicians work in shifts and the reaction to a medication might not be visible during a single shift.

6 RELATED WORK

Reactive system design has a long history. Different kinds of design methods have been proposed to facilitate the development of reactive systems (Wieringa, 2003). The database-centric approach for complex reactive systems is still quite new and various extensions for contemporary DBMS are need. There are already some extensions of standard SQL providing the syntactic instruments to handle phase-like concepts. 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. Also related to this work are the achievements made by the stream processing community (Krämer and Seeger, 2004). Precise semantics are defined and concrete syntactic extensions to standard SQL are proposed; a systematic means to manage evolving knowledge and explicit provenance support is however still missing.

Knowledge representation (Davis et al., 1993) is a fundamental research topic in computer science. Expert systems try to use production rules to form a computable knowledge base have gained successful applications (Shortliffe, 1976). These technologies however do not scale well for large datasets. Supporting the management of ontological dataset as knowledge in DBMS is gaining more attractions from both academia and industry (Das and Srinivasan, 2009). Based on the relational database, the storage and query of these graph datasets are rather efficient, however a systematic approach to explicitly utilise the knowledge to analyse the captured data is still missing.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we applied both the KIDS model to a typical use case from the medical domain. We then analyzed the functional requirements to a traditional relational database system to be able to fully support KIDS. The concept of phases integrates as a means to model uncertainty in the derived information in KIDS. In our future work we plan to further analyze and reveal the potential of phases, in particular the prediction model and decision support. Besides of that SQL extensions are going to be implemented along with a prototype embedded in a commercial relational database management system.

REFERENCES

Behrend, A., Dorau, C., and Manthey, R. (2009). Sql triggers reacting on time events: An extension proposal. *ADBIS*.

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

Chan, E. S., Behrend, A., Gawlick, D., Ghoneimy, A., and Liu, Z. H. (2012). Towards a synergistic model for managing data, knowledge, processes, and social interaction. *SDPS*.

Das, S. and Srinivasan, J. (2009). Database technologies for rdf. *Reasoning Web. Semantic Tech. for Inf. Systems*.

Davis, R., Shrobe, H., and Szolovits, P. (1993). What is a knowledge representation? *AI magazine*, 14(1):17.

Eom, S. B., Lee, S. M., Kim, E., and Somarajan, C. (1998). A survey of decision support system applications (1988-1994). *Journal of the Operational Research Society*.

Karvounarakis, G., Ives, Z. G., and Tannen, V. (2010). Querying data provenance. *Proceedings of the 2010 ACM SIGMOD*.

Kawamoto, K., Houlihan, C. A., Balas, E. A., and Lobach, D. F. (2005). Improving clinical practice using clinical decision support systems: a systematic review of trials to identify features critical to success. *Bmj*, 330(7494):765.

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

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

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

Schmiegelt, P., Xie, J., Schüller, G., and Behrend, A. (2013). Towards an integrated approach to monitor and analyse health care data using relational databases. *HealthInf*.

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*. Artificial intelligence series. America Elsevier Publishing Company, Inc.

Simmhan, Y. L., Plale, B., and Gannon, D. (2005). A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36.

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

Wieringa, R. J. (2003). *Design Methods for Reactive Systems: Yourdon, Statemate and the UML*. Morgan Kaufmann Publishers.