Model-Based Query Language for Analyzing Clinical Processes

Janis Barzdins*, Juris Barzdins**, Edgars Rencis*, Agris Sostaks*

* Institute of Mathematics and Computer Science, University of Latvia, Riga, Latvia
** Faculty of Medicine, University of Latvia, Riga, Latvia

Abstract

Nowadays large databases of clinical process data exist in hospitals. However, these data are rarely used in full scope. In order to perform queries on hospital processes, one must either choose from the predefined queries or develop queries using MS Excel-type software system, which is not always a trivial task. In this paper we propose a new query language for analyzing clinical processes that is easily perceptible also by non-IT professionals. We develop this language based on a process modeling language which is also described in this paper. Prototypes of both languages have already been verified using real examples from hospitals.

Keywords:
Hospital, Management, Modeling languages, Query languages

Introduction

As mentioned in [1], hospitals are expert organizations, and one of problems is how many different expert groups (managers, doctors) are able to collaborate. Since medical professionals but not the managers carry the ultimate responsibility for the patient’s outcomes, the management has a limited control over the doctors’ individual bedside decisions. Therefore, a more profound involvement of the doctors in transforming the processes within their health care organizations has been widely regarded as a factor that is critical for their success [2],[3],[4],[5]. In contrast to the professional managers who have received an appropriate training and who control the administrative resources (e.g., specially dedicated business analysts for extracting process knowledge from the increasing amount of digitally stored data), doctors so far have benefited to a much lesser degree from the advantages of health information technologies (HIT) for better understanding of clinical patterns. The goal of our research is to develop a business model-based method for hospital use which would allow doctors to retrieve directly the ad-hoc information from various hospital databases which is needed in building their process-oriented knowledge for their managerial roles.

Modeling Language for Hospitals

If we take some existing hospital information system, it is not a big problem to supplement it with new hardcoded queries of hospital processes. The problem, however, arises when we want to create new queries on the fly that are not included in the set of predefined ones. In this case we have to know the ontology of the system in order to be able to develop reasonable queries. Traditional ontology languages are not very helpful, when we need to deal with processes describing dynamic behavior. To create an ontology language which is able to cope with this situation, firstly we need to address a separate problem — we need a language for describing processes in a hospital.

The traditional process modeling languages have found a limited use in the hospital settings (see, e.g., [6], [7]). One of the reasons behind this delay has been the lack of clear definition of the sequence of activities that are carried out in clinical processes.

In recent years business processes in hospitals have been studied for the applicability of modeling methods used in other industries. For example, there are published reports of successful usage of BPMN for describing the clinical process for strictly selected group of patients with a specific diagnosis in oncology [8] and the process in a selected department for pathology investigations [9]. However, there are also reports suggesting that application of BPMN is difficult in the specific domain of health care, since the nature of health care processes in a multidisciplinary hospital is inherently complex [6], and that is the reason for the usage of domain-specific modeling languages and tools [7].

To overcome these problems, we propose a new domain-specific modeling language for hospital modeling (called MEDMOD), which borrows the most useful features from UML Class, Activity and Use-case diagrams. Since UML Class Diagrams are very close to ontologies [10], we can look at the MEDMOD language as at a special ontology language designed for describing processes in a hospital.

Let us now describe the elements of MEDMOD in more detail (see Figure 1).

Activity. Activity is the central element of the MEDMOD language and denotes a task in time having a start and end moments. Semantically it is related to the Action element of UML Activity diagrams. Examples of Activity are seen in Figure 1 depicted as yellow boxes with rounded corners.

From the linguistic point of view, we divide Activities in three different categories based on how the Activity name is formed. The first type of Activities is the most common one and conforms to the simple present linguistic form — “Doctor sets diagnosis”. The second type of Activities is formed in passive voice and used in cases, when there can be different consequences to some previous Activity leading to execute one of different outgoing flows from it — “Patient admitted to hospital ward”. The third type of Activities refers to a greater process with some given name, which then serves as a name for the Activity — “Clinical process in ward”. These naming conven-
Activities can also have attributes of five primitive data types – Integer, Real, String, Boolean and DateTime. These attributes can be specified for every concrete Activity at diagram creation, and different values can be assigned to these attributes of concrete instances of the Activity at run-time. Since there are very detailed codifiers in the medical world for coding every procedure, diagnosis or other attributes (see Health Level Seven International, the global authority on standards for interoperability of health information technology [19]), we also allow using enumerations as data types. For instance, the Activity “Doctor assigns procedure” has an attribute “procedure_code”, whose values come from the enumeration “pCode” (see Figure 1).

Exactly one of the Activities of every MedMod diagram is denoted as the Start Activity meaning that the execution of the diagram starts with this Activity (there can be no ingoing arrows to this Activity, e.g., see Activity “Patient enters the hospital” in Figure 1).

Follows. This type of oriented relation can be established between two Activities A and B meaning that Activity B can only start after Activity A has ended (the same semantics as the Control Flow of UML Activity diagrams). It is allowed for several Activities to follow the same Activity – the XOR semantics is implied in this case meaning that only one of those outgoing flows can be executed. We denote this situation by introducing a new diamond-shaped graphical element seen in Figure 1. It is also allowed to have several ingoing flows into an Activity implying the OR semantics, i.e., the following Activity can start executing when at least one flow has executed, and several instances of that Activity can arise, if several incoming flows executes at different times. It is, however, not allowed to introduce several parallel outgoing flows from the same Activity. We substitute the parallel branching of UML Activities with a more general feature, the composition, by introducing so called Aggregate Activities and their parts – Component Activities that can be executed simultaneously.

Composition. A composition between two Activities can be established, if one Activity (called the Aggregate) semantically consists of one or more other Activities (called the Components). It has an analogy with the relation “includes” of UML Use-case diagrams. We have borrowed the notation for the Aggregate part of Composition (the filled diamond) from the UML Class diagrams. Also, a composition fork graphical element can be introduced to collect the Components of the same Aggregate Activity (seen in Figure 1). Each Component Activity can appear several times within the Aggregate, therefore we also allow cardinalities to be attached to the Component end of a Composition (the default cardinality is 1).

Interruption. An interesting phenomenon relates the composition – what is the semantics of a Follows-type relation going out from the Aggregate Activity? It was stated before that the Follows flow can execute when the Activity A has ended. But the Aggregate Activity can actually never end, if it has at least one Component having a cardinality, e.g., * (many). In this case the Aggregate is constantly waiting for new and new Component instances to born, and only some force from outside can decide, when to stop the waiting process. We must therefore introduce a new type of control flow – an Interruption – stating that if there is an outgoing Interruption flow from the Aggregate Activity A to some Activity B, it means that the Activity A is suspended, when the flow is executed (i.e., when the Activity B needs to be started) meaning that it can no more create new Component instances (already created Component instances continues to execute normally). For instance, in Figure 1 the Activity “Clinical process in ward” is suspended when the doctor decides to either transfer the patient to another ward, or to discharge the patient. The Interruption flow is adorned with a jagged “lightning bolt” arrow. Simple Activities can also be interrupted in similar manner.

Extension. Extension is an oriented relation between two Activities A and B meaning that Activity B can be called at some time during the execution of Activity A. This feature allowing us to extend the Activity is also borrowed from UML Use-case diagrams. The call is triggered, when some predefined condition occurs. The condition is described as an Extension point name and attached to the Extension. For instance, a doctor can decide that a “second opinion is necessary” (the Extension point name) during the evaluation of patient’s medical needs. In that case another Activity “Patient consulted by second doctor” is called (see Figure 1).

If we look at the MedMod diagram from the process point of view, we can notice that every instance of the Start Activity defines a separate transaction consisting of those instances of Activities that can be reached from the instance of the Start Activity (these are called run-time instances in the process modeling world). The basic assumption we make here is that no two transactions can ever share any common instances. It must be notices that certain Activities (so called Multiple Activities) can have several instances within a transaction (because of loops and cardinalities of type “many”). We use a

![Figure 1 - An example of a MEDMOD process](image-url)
slightly different visual representation for this type of Activities for better perception as can be seen in Figure 1.

Formally, we define the MEDMOD as a profile on UML Class diagrams as can be seen in Figure 2 (OCL constraints are omitted here). If we perceive the process diagram as a Class diagram, we can talk about instances of the process (as instances of the underlying Class Diagram). Hence we can perceive the diagram as an ontology consisting of T-Boxes (Classes) and A-Boxes (their run-time instances).

However, we will not go in a greater detail concerning the theory of modeling languages. Instead, we will look at diagrams only from the doctor’s point of view because our basic assumption is that the MEDMOD language must be understandable intuitively without deep knowledge of the modeling science. Similar approach has been considered also in [11] The understandability of MEDMOD diagrams was confirmed by testing them on several doctors. After giving a brief instruction (about half an hour) about the language concepts, the doctors were asked questions about parts of various MEDMOD diagrams. The answers were mostly correct, and doctors generally approved that the language is also very well perceptible.

Figure 2 - The UML profile defining the MEDMOD language

When the process is described, is it very important for the doctor to be able to see the run-time instances (both within one transaction – set of instances originated from one common instance of the Start Activity – and over several transactions) with their respective attribute values from different points of view. One idea here could be to export all transactions over some period of time to Microsoft Excel and then use its features to analyze the data. The main problems here arise from the fact that we can have loops and cardinalities of type “many” allowing several run-time instances appear for a concrete Activity. Developing a non-trivial query for this case may involve serious “Excel programming” not being possible for a doctor. To overcome this problem, we have developed a simple process query language that is based on the process diagram that needs to be analyzed.

Process Query Language

The purpose of Process Query Language (PQL) is to allow a doctor interested in clinical processes querying (filtering) runtime data of hospital’s processes described using MEDMOD. In fact, a doctor should be able to ask questions like “What were the total expenses of all Dr. Jekyll’s patients?” or “Which patients with diagnosis Pneumonia (code: J15) had more than two X-ray diagnostics (code: 50012)?”.

There are works on querying the descriptions of the business processes without the underlying data, e.g., work of Beeri et.al. [12], where the visual query language BP-QL has been introduced, and the BPMN-Q language by Awad [13].

Beeri et.al. [14] went a step further by introducing the BPMMon – a query language for monitoring business processes, which allows the users defining the monitoring tasks and retrieving their associated reports visually. Although the language is simple enough for IT specialists, it is hardly useable by doctors in the hospitals. For example, the specification of reports (retrieved data) requires knowledge about XML.

Beheshti et.al. [15] introduced a process mining and querying methodology, where data are acquired also from the information system. These data are called Event Logs and are grouped into folder nodes – a similar concept to transactions presented in this paper. However, the query language is itself based on SPARQL making it impractical for a broader use by the hospital staff.

G.Barzdnis et. al. [16] have developed a graphical query language and tool for querying ontology instances. However, they use graphics to denote only the query and do not involve the graphical description of the ontology nor in the construction process of the query, nor in the displaying the results.

Figure 3 demonstrates various queries and their answers based on the process example seen in Figure 1. Asking questions begins with choosing the MEDMOD process diagram, which describes the process under inspection, switching to the filtering mode and setting the time interval the doctor is interested in. The time interval is shown as a box in the upper part of the diagram. Every activity node has an indicator (the attached box) showing the number of instances in the initial dataset.

Now the doctor can undertake two types of actions – she can set filtering conditions or retrieve data. There are several options for filtering. The first filtering option is the comparison operations like equals, greater than, less than, contains, begins with, etc. The actual list of operations depends on the data type of the attribute. The same principle has been used in spreadsheet applications like Microsoft Excel for setting simple filtering conditions on column values.

The second filtering option is the data partitioning operations like getting Top or Bottom instances based on some attribute. Doctor may ask for 10 transactions (patient treatings in hospital), where total expenses are the largest.

If there is a possibility that a transaction may contain more than one instance of the same type (e.g., if there is a composition with a cardinality “many” or flows heading backwards), then it is possible to set a filtering condition on aggregate functions. The filtering conditions may be applied to the Sum or Average of attribute values of instances in the transaction. The filtered dataset contains all instances from those transactions, which contains instances of the filtered type having sum or average value of the given attribute within values specified by the condition. For example, the doctor may ask for those transactions, where average cost of “Procedure is executed” is greater than 100. Another option is to set comparison condition on number of instances within a transaction using the Count operation. For example, doctor can ask for transactions where “Doctor sets diagnosis” happened more than once. Setting this condition may be initiated by clicking on the action node itself (not on an attribute). In the process query diagram a filtering condition appears as a label in the corresponding action’s node. Thus, user is always aware of filtering conditions have been set. Immediately after the filtering condition has been created or updated, it is applied on the dataset. The filtered dataset contains all instances from those transactions which contains instances conforming to the filtering condition. As a consequence, all data displayed in the diagram (e.g. the indicators of number of instances) are updated. For example, “Patient enters the hospital” activity has the filtering condition on the age attribute (age > 60).
There are also several options for data retrieving. The first option is the time interval. It can be drawn between actions containing date\&time attribute. The time interval symbol appears in the diagram. The incoming arrow denotes interval’s start activity and the outgoing arrow denotes interval’s end activity. Note that there may exist multiple interval values because of multiple start and end instances within the transaction. To specify more precisely the instances the time interval should be measured between, a conditional expression may be used. For example, if the doctor wants to measure time between “Doctor sets procedure” and “Procedure is executed” for those instances, whose procedure code matches, she should supply the conditional expression stating “start procedure code equals end procedure code”. If no conditional expression has been supplied, then the interval between two adjacent (in time) instances of corresponding types is measured.

The second data retrieving option is the aggregate functions which can be evaluated over the filtered dataset in order to obtain a single number as an answer to the question asked. They are: Count, Sum and Average, meaning respectively number of instances within dataset, sum of the given attribute values over all instances in the dataset and the average of the given attribute values over all instances. They can be applied by dragging the selected function from the palette to the corresponding activity node (for Count) or to the attribute (for Sum and Average). The result (one number) appears on the diagram.

Figure 3 - An example of Process Query Diagram describing a hospital operation.
as an indicator box which displays the computed value. For example, the box labelled “Sum of total expenses” displays the sum of total_expenses attribute value of all instances in the filtered dataset (100001).

The third possible option of retrieving data is a list of all instances corresponding to the selected activity as an instance table. Dragging corresponding palette element to an activity initiates the display of all instances of corresponding type in the filtered (or initial if no filtering conditions applied) data set. They are displayed as a table where each row represents an instance, but columns represent the attribute values. There is also one special column containing slice’s ID the instance belongs to. Since it is possible to display several instance tables at once, the presence of ID in each of them helps to recognize data from the same slice across several tables. There are two instance tables shown in Figure 3, one attached to the “Patient enters the hospital” activity, another attached to the “Procedure is executed” activity.

The main advantages of the PQL are: 1) the view on data through “glasses” of familiar process, 2) the simple and easy-to-perceive means of setting filtering conditions require no more expertise than using spreadsheet applications (like MS Excel), and 3) the dynamic response to each step in construction of the complete query – the doctor sees immediate reaction to every action. It shortens the learning curve greatly and encourages even non-experienced users to try this out.

Conclusions

To test the practical aspects of using this methodology we presented it to a group of seven doctors working in a hospital. Our primary interest was to assess the “readability” of the designed clinical process model and of the information filtered with its application by the end-users. After a short instruction about the syntax of process description, available filtering mechanisms and visualizing the retrieved information in data indicator boxes next to each of activity nodes, doctors were asked to explain the meaning of the three prepared screenshots representing retrieved data with a use of the query language. All participants of this test demonstrated that they could accurately retrieve the question to be asked by applying the proposed querying techniques in our hospital model. In general, all of the participating doctors rated the presented methodology positively and noted not only the potential for this tool to facilitate management and improve the transparency of clinical processes, but also its potential for research on the impact that certain variables have on the treatment outcomes.

Another aspect that is currently being evaluated is the efficiency of the possible implementation of the proposed query language. The ongoing development of the query language support tools have shown that the clinical data can be naturally structured in the way which is suitable for storing and retrieving from high performance key-value data stores (NoSQL databases). It allows fast retrieval of query results, which are mainly based on filtering values of the given attributes (keys).

Acknowledgments

This work has been partially supported by the European Regional Development Fund within the project Nr. 2010/0325/2DP/2.1.1.1.0/10/APIA/VIAA/109 and by the Latvian National Research Program Nr. 2 „Development of Innovative Multifunctional Materials, Signal Processing and Information Technologies for Competitive Science Intensive Products” within the project Nr. 5 „New Information Technologies Based on Ontologies and Model Transformations”.

References


Institute of Mathematics and Computer Science, University of Latvia, Raina Blvd. 29, LV-1459, Riga, Latvia

Edgars Rencis, edgars.rencis@lu.lv