**Managing proposals and evaluations of updates to medical knowledge: theory and applications**

# Managing proposals and evaluations of updates to medical knowledge: theory and applications

Luca Anselma%, Alessio Bottrighi*, Stefania Montani*, Paolo Terenziani*

% (corresponding author) Dipartimento di Informatica, Università di Torino

corso Svizzera 185, 10149 Torino, Italy

Phone: +39 011 6706769 Fax: +39 011 751603 E-mail: anselma@di.unito.it

* Dipartimento di Informatica, Università del Piemonte Orientale "Amedeo Avogadro"

via Teresa Michel 11, 15100 Alessandria, Italy

Phone: +39 0131 360174 Fax: +39 0131 360198

E-mail: {alessio.bottrighi, stefania, terenz}@mfn.unipmn.it

## Abstract

The process of keeping up-to-date the medical knowledge stored in relational databases is of paramount importance. Since quality and reliability of medical knowledge are essential, in many cases physicians' proposals of updates must undergo experts' evaluation before (possibly) becoming effective. However, until now no theoretical framework has been provided in order to cope with this phenomenon in a principled and non ad-hoc (task and domain independent) way. Indeed, such a framework is important not only in the medical domain, but in all Wikipedia-like contexts in which evaluation of update proposals is required. In this paper we propose GPVM (General Proposal Vetting Model), a general model to cope with update proposal\evaluation in relational databases. GPVM extends the current theory of temporal relational databases (and, in particular, BCDM –Bitemporal Conceptual Data Model– "consensus" model), providing a new data model, new operations to propose and accept\reject updates, and new algebraic operators to query proposals. The properties of GPVM are also studied. In particular, GPVM is a consistent extension of BCDM and it is reducible to it. These properties ensure consistency and interoperability with most relational temporal database frameworks, facilitating implementation (on top of current frameworks) and their adoption in application contexts.

**KEYWORDS**: Temporal clinical relational databases; BCDM semantic model; Temporal data model, algebra, proposal/evaluation operations; Proof of reducibility to standard models.

## 1 Introduction

Our research group has a long-term cooperation with Azienda Ospedaliera San Giovanni Battista in Turin, Italy (henceforth called "Hospital" for short), one of the largest hospitals in Italy. Such a cooperation is part of the GLARE (GuideLine Acquisition, Representation and Execution) project [1,2] for the development of a computer-based approach for the management of clinical guidelines. In the Hospital, clinical guidelines are usually built incrementally either from scratch or as an adaptation and contextualization of pre-existent guidelines and need to be kept up-to-date with new therapeutic and\or diagnostic procedures. In practice, alternative proposals of insertion\update\deletion issued by specialists are periodically evaluated by a team of experts who are responsible for the final result. Accepted proposals lead to a new version of the guideline, which becomes the reference one for all the medical and paramedical personnel of the hospital. Past versions of the guidelines must be maintained, e.g., for legal purposes. In the following we term such a work paradigm "*proposal vetting*". In fact, users cooperatively propose updates to a reference version of data\knowledge

and, afterward, a team of experts evaluates such proposals. Eventually, the approved proposals lead to a new *data\knowledge reference version*

We believe that the development of proper tools supporting proposal vetting can play an important role in the definition and, even more importantly, in the dissemination of clinical guidelines, and it can improve the acceptance and adoption of guidelines in the clinical practice. Indeed, the increasing demand for standardized and high-quality health assistance grounded on medical evidence has lead to the definition of thousands of clinical guidelines at the international, national or local (e.g., for a specific hospital) level. Both public and private guidelines have been developed, sometimes pursuing different tasks (consider, e.g., public hospitals vs. insurance companies guidelines) and devising different policies to enforce their adoption in the clinical practice. In such a heterogeneous context, tools managing proposal vetting can support the development of new guidelines as an evolution\adaptation of pre-existent ones. In particular, in clinical guideline contextualization [3] international and national guidelines are adapted to fit the needs, resources and policies of a specific hospital. In such a case, proposal vetting can be exploited to allow all the physicians involved in the guideline execution to play an active role in the adaptation. Indeed, such an active participation is very likely to facilitate the acceptance and effective adoption of the guidelines by the physicians themselves.

Though clinical guidelines are the current application context of our approach, the problem we face and the solutions we propose are more general. Proposal vetting finalized to cooperative modeling\update of shared data\knowledge is an important paradigm in Computer Science, and seems to become more and more important due to the large-scale availability of the Internet. For instance, the construction of a free encyclopedia as Wikipedia [4], and of vocabularies like Wiktionary [5], with the contributions of hundreds of thousands of authors, is a modern phenomenon with yet undiscovered social and cultural impacts. Among the others, the Wiki technology is also being used to build Citizendium [6], a free encyclopedia in which experts are called to approve the piece of data proposed by contributors, for the sake of reliability and quality.

All the above-mentioned applications take advantage of *relational database management systems* (henceforth, DBMSs), exploiting their generality and efficiency. For instance, Wikipedia stores data into a relational DBMS such as MySQL [7]. Indeed, the relational model is the most successful data model for data-processing and the vast majority of data-processing applications (including those used in healthcare domains) are based on relational DBMSs. For these reasons in this work we assume that data/knowledge are stored in a relational DBMS.

It is worth noticing that proposal vetting involves an explicit treatment of ***time***. This is obviously the case, e.g., of the clinical guideline domain. Physicians have to be compliant with the *reference* guidelines, and such a compliance may be checked (e.g., for legal reasons or for quality evaluation). Thus, the history of the guidelines needs to be

maintained, to grant that, even if a guideline has been changed afterwards, one has a record of the fact that a physician was compliant with the old reference version. Moreover, in many cases, stored data is intrinsically temporal.

### 1.1 Structure of the paper

The paper is organized as follows. Section 2 provides a background on the relational model and on the BCDM model. Section 3 introduces a clinical guideline example, that will be resorted to as a running example in the rest of the paper. Section 4 clarifies our goals and methodology. Sections 5 to 7 represent the core contribution of our paper. Specifically, Section 5 illustrates the extended data model we have introduced to deal with the proposal vetting phenomenon. Section 6 describes the manipulation operations we have defined to allow users properly manipulate the data. Section 7 is related to our extended relational algebra. Section 8 describes an implementation. Finally, Section 9 discusses some related works, and Section 10 is devoted to conclusions and future research directions.

## 2 Background

## 2.1 The relational model

The relational model has been proposed in 1970 by Edgar F. Codd [8]. Data are represented as a collection of two-dimensional tables called relations. Relations are sets of "tuples", i.e., sets of the rows of the tables. Each relation has a schema, which includes its attributes. Information in relations is represented in the tuples of the relations by means of the values of the attributes. For example, the tables below represent the relations FACULTY and DEPARTMENT. The relation FACULTY has schema (Name, Role, DeptName) and it contains two tuples representing the fact that Mary is an assistant professor at the department of computer science and that John is a full professor at the department of history. The relation DEPARTMENT has schema (DeptName, Address) and it contains a tuple for each department, representing its name and address.

FACULTY

| Name | Role | DeptName |
|---|---|---|
| Mary | assist prof | Computer Science |
| John | full prof | History |

DEPARTMENT

| DeptName | Address |
|---|---|
| Computer Science | 101 North Street |
| History | 25 South Street |

Notice that relations can contain a large number of tuples and that information is "broken" among different relations, for the sake of avoiding redundancy. Therefore, a relational database must be queried in order to extract information. Codd [8] proposed a relational algebra with operators that take one or two relations as inputs and produce a new relation as output. Here we recall five basic relational algebraic operators: relational union ($\cup$), relational difference ($-$), selection ($\sigma_P$), projection ($\pi_X$) and natural join ($\bowtie$). Relational union and difference consider the relations as sets of tuples and compute set-theoretic union and difference. Notice that two tuples with the same values are considered as the same tuple. Selection picks from a relation the tuples that satisfy a logical predicate. Projection picks from a relation the chosen attributes, possibly collapsing the tuples resulting as the same tuple. Natural join operates on two relations; it gives as a result a relation consisting of the set of all combinations of tuples in the input relations that are equal on their common attribute names. Since in the following we consider the extension of natural join as a bitemporal operator and as a proposal-vetting operator, here we provide a formal definition of the standard relational natural join.

**Notation.** Given a tuple x defined on the schema $R=(A_1,...,A_n, B_1,...,B_l)$, we denote with A the set of attributes $(A_1,...,A_n)$. Then x[A] denotes the values in x of the attributes in A.

Given two relations r and s with schema $(A_1,\ldots,A_n, B_1,\ldots,B_m)$ and $(A_1,\ldots,A_n, C_1,\ldots,C_k)$ respectively, natural join $\bowtie$ provides as an output a relation over the schema $(A_1,\ldots,A_n, B_1,\ldots,B_m, C_1,\ldots,C_k)$ defined as follows (let A stand for $A_1,\ldots,A_n$, B for $B_1,\ldots,B_m$ and C for $C_1,\ldots,C_k$):

$r \bowtie s = \{z \setminus \exists\, x \in r, \exists\, y \in s\ (x[A]=y[A] \wedge z[A]=x[A] \wedge z[B]=x[B] \wedge z[C]=y[C])\}$ ♦

As an example of query we consider the case where we want to extract the name and work address of Mary. Such a query can be expressed as $\pi_{Name,Address}(\sigma_{Name="Mary"}(\text{FACULTY} \bowtie \text{DEPARTMENT}))$. The innermost operator is the natural join operator and it combines the tuples of FACULTY and DEPARTMENT that are equal on the common attribute DeptName. Then, the selection operator picks only the tuple regarding Mary. Finally, only the attributes Name and Address are retained. The result is the following relation:

$\pi_{Name,Address}(\sigma_{Name="Mary"}(FACULTY \bowtie DEPARTMENT))$

| Name | Address |
|---|---|
| Mary | 101 North Street |

## 2.2 The relational bitemporal model

In clinical domains, as in many other real-world domains, time has a pervasive nature and it may be represented as a particular kind of information. Over twenty years of research in temporal databases (henceforth, TDBs) have identified some core concepts. It is convenient to represent time as an attribute: i.e., tuples are associated with one or more temporal attributes. However, time is different from other attributes: in fact, the treatment of time in the relational approach involves the solution of difficult problems, and the adoption of advanced dedicated techniques [9]. In this spirit, many extensions to the standard *relational* model were devised, and more than 2000 papers on TDBs were published over the last two decades (cf., the cumulative bibliography in [10], the section about TDBs in the Springer Encyclopedia of Databases [11], the entry "Temporal Database" in [11], and the surveys in [12, 13, 14, 15]). In many of such approaches two independent time dimensions have been identified, namely transaction time and valid time. *Transaction time* represents the time when a tuple is present in the database. *Valid time* represents the time when the fact described by a tuple holds in the modeled world. Let us consider the following example.

**Example 2.1.** The human resources division has *inserted in the database at time 1* and *deleted from the database at time 10* (transaction time) the following information: Mary works as an assistant professor *from time 2 to time 20* (valid time). Then the human resources division has *inserted in the database at time 10* and this tuple is *still in the database* (transaction time) the following information: Mary works as an associate professor *from time 15 to time 40* (valid time).

BCDM (Bitemporal Conceptual Data Model [16]) is a unifying data model developed in order to isolate the "core" notions underlying many temporal relational approaches including the "consensus" TSQL2 one [17]. To identify such a "core", BCDM does not face issues such as data representation and storage optimization, aiming at a "semantic" approach. Please note that, as BCDM, also our approach operates at the semantic level (not to be confused with the "conceptual" -e.g., Entity-Relationship- level). In BCDM tuples are associated with *valid time* and *transaction time*. For both domains, a limited precision is assumed and the *chronon* is the basic time unit. Both time domains are totally ordered and isomorphic to the subsets of the domain of natural numbers. The domain of valid times $D_{VT}$ is given as a set $D_{VT}=\{t_1,t_2,\ldots,t_k\}$ of chronons, and the domain of transaction times as $D_{TT}=\{t'_1,t'_2,\ldots,t'_j\}\cup\{UC\}$ (where UC –Until Changed– is a distinguished value). In general, the schema of a bitemporal conceptual relation $R=(A_1,...,A_n|T)$ consists of an arbitrary number of non-timestamp attributes $A_1, \ldots,A_n$, encoding some fact, and of a timestamp attribute T, with domain $D_{TT}\times D_{VT}$. Thus, a tuple $x=(a_1,\ldots,a_n|t_b)$ in a bitemporal relation r(*R*) on the schema R, henceforth called a BCDM

bitemporal tuple, consists of a number of attribute values associated with a set of bitemporal chronons $t_{b_i}=(c_{t_i}, c_{v_i})$, with $c_{t_i} \in D_{TT}$ and $c_{v_i} \in D_{VT}$. The intended meaning of a bitemporal BCDM tuple is that the recorded fact is *true in the modeled reality* during each *valid-time* chronon in the set, and is *current in the relation* during each *transaction-time* chronon in the set. Valid-time, transaction-time and non temporal tuples are special cases, in which either the transaction time, or the valid time, or none of them are present.

**Notation.** Given a tuple x defined on the schema $R=(A_1,...,A_n \mid T)$, x[T] denotes the set of bitemporal chronons constituting the timestamp of x, $x[T_v]$ and $x[T_t]$ denote the valid and transaction time of a valid-time and transaction-time tuple respectively.

**Terminology.** As in BCDM (and TSQL2) will call $A_1,...,A_n$ ***explicit*** attributes, and $T_v$ and $T_t$ ***implicit*** attributes. We will call ***value-equivalent*** two (or more) tuples having the same values for the explicit attributes. ♦

It is important to stress that the BCDM model explicitly requires that *no two value-equivalent are allowed in the same temporal relation* [17]. This choice is essential in order to enhance the semantic clarity of the model, since it grants that the full time history of a fact is recorded in a single tuple (instead of being scattered between different tuples). As a consequence, BCDM is a not-ambiguous data model, in the sense that, in BCDM, there is a unique way of coding any temporal information (see the Uniqueness Property of BCDM [17]). On the contrary, allowing value-equivalent tuples would lead to ambiguous representations, i.e., to alternative semantically equivalent representations for the same content (consider, e.g., the notion of snapshot-equivalence in TSQL2 [17]).

A special routine makes explicit the semantics of the special value UC: as time passes, at each clock tick for each bitemporal chronon $(UC,c_v)$, a new bitemporal chronon $(c_t,c_v)$ is added to the set of chronons, where $c_t$ is the new transaction-time value.

**Notation.** A bitemporal BCDM tuple x is *current* if it is present at the current time ("now") in the database (i.e., it has not been updated or deleted yet). Formally, this means that the bitemporal chronons of x contain UC as a transaction time, i.e., *current(x)*: $\exists c_v \setminus (UC,c_v) \in x[T]$. ♦

Example 2.1 can be represented in BCDM relations as following. For the sake of brevity, in the attribute T we represent the bitemporal chronons as the Cartesian product between the interval of the transaction time and the interval of the valid time, i.e., $[1,UC]\times[1,10000]$ represents the set of bitemporal chronons {(1,1), (1,2), …, (1, 10000), (2,1), …, (2,10000), …, (UC,1), …, (UC,10000)}.

FACULTY

| Name | Role | DeptName | T |
|---|---|---|---|
| Mary | assist prof | Computer Science | [1,9]×[2,20] |
| Mary | assoc prof | Computer Science | [10, UC]×[15,40] |
| John | full prof | History | [1,UC]×[1,100] |

DEPARTMENT

| DeptName | Address | T |
|---|---|---|
| Computer Science | 101 North Street | [1,UC]×[1,100] |
| History | 25 South Street | [1,UC]×[1,100] |

*Insertion* and *deletion* of tuples are directly defined in BCDM. Insertion in a relation r of values $(a_1,\ldots,a_n)$ valid at time $t_v$ results in the fact that relation r will include a tuple with the provided values and valid time and with UC as a chronon in the transaction time. Deletion concerns the *logical* removal of a tuple from the current state. In order to retain the history, the tuple is *not* physically deleted, but only logically removed. Thus, BCDM deletes from the tuple all bitemporal chronons $(UC, c_v)$, where $c_v$ is any valid-time chronon.

**Terminology**. In the paper, we will call "*closure*" the operation of removing chronons containing UC, and we will say that the corresponding tuple is "*closed*".

*Algebraic* operators are defined on the bitemporal model as a temporal extension of Codd's operators. As in most approaches in the TDB literature (see, e.g., the survey in [12]), in BCDM such extensions behave like standard non-temporal operators on the explicit attributes, and involve the application of set operators on the implicit attributes for value-equivalent tuples. This approach ensures that the temporal algebraic operators are a *consistent extension* of Codd's operators and are *reducible* to them when the temporal dimension is removed. This definition can be also motivated by the *sequenced* semantics [18]: the results of algebraic operations should be valid independently at each point of time. More precisely, bitemporal relational union performs the union on the temporal attributes, bitemporal relational difference performs difference, selection does not alter tuples, and projection performs union on the temporal attributes. Bitemporal natural join operates as the non-temporal natural join on the explicit attributes and intersects the implicit attributes on value-equivalent tuples. More formally, given two relations r and s with schema $(A_1,\ldots,A_n, B_1,\ldots,B_m|T)$ and $(A_1,\ldots,A_n, C_1,\ldots,C_k|T)$ respectively, bitemporal natural join $\bowtie$ provides as an output a relation over the schema $(A_1,\ldots,A_n, B_1,\ldots,B_m, C_1,\ldots,C_k|T)$ defined as follows (let A stand for $A_1,\ldots,A_n$, B for $B_1,\ldots,B_m$ and C for $C_1,\ldots,C_k$):

$$r \bowtie^B s = \{z \setminus \exists\, x \in r, \exists\, y \in s(x[A]=y[A] \wedge x[T] \cap y[T] \neq \emptyset \wedge z[A]=x[A] \wedge z[B]=x[B] \wedge z[C]=y[C] \wedge$$

$$z[T]=x[T] \cap y[T])\} \blacklozenge$$

As an example of bitemporal query we consider the case where we want to extract the name and work address of Mary. Such a query can be expressed as $\pi^B_{Name,Address}(\sigma^B_{Name="Mary"}(\text{FACULTY} \bowtie^B \text{DEPARTMENT}))$. The result is the following relation. The bitemporal attribute is the result of the intersection of the bitemporal values of the tuples

regarding Mary in relation FACULTY and regarding Computer Science department in relation DEPARTMENT, and in the union of the chronons of the resulting two tuples.

$\pi_{Name,Address}(\sigma_{Name="Mary"}(FACULTY \bowtie DEPARTMENT))$

| **Name** | **Address** | **T** |
|---|---|---|
| Mary | 101 North Street | [1,9]×[2,20] ∪ [10, UC]×[15,40] |

**Property. Reducibility** [17]**.**

BCDM algebraic operators behave like the corresponding non-temporal relational algebra operators: with identical arguments, they always return identical results. ♦

Such a property is essential in order to grant that, when time is disregarded, BCDM algebraic operators behave like standard SQL one. This property grants "continuity" for users and, above all, interoperability with pre-existent non-temporal approaches. As a remarkable side-effect, this also grants for the fact that BCDM can be implemented as an additional layer on top of the SQL model (and, indeed, several prototypical implementations have been already devised following such a strategy).

In summary, the BCDM model is, from the one side, a general unifying semantic model for several TDB approaches, including TSQL2; from the other side, it is a "consistent extension" of the standard (non temporal) relational algebra. Moreover, as we will see in Section 5, substantial extensions to BCDM are needed to cope with proposal vetting. Such extensions will be studied in the rest of the paper.

## 3 Case study: proposal vetting about clinical guidelines

We consider the guideline for the management of suspected acute pulmonary embolism [19, 20] adopted by the Hospital.

The guideline indicates some diagnostic investigations to confirm or discard the suspect of acute pulmonary embolism and, in case such suspect is confirmed, it dictates the proper set of therapeutic actions. The first action of the guideline is *pulmonary embolus detection*. In the initial version of the Hospital guideline, such an action had to be executed through pulmonary ventilation perfusion scintigraphy, performed using isotope lung scanning (VQS). The estimated cost of such operation is about 100 €, and image acquisition lasts about 15 minutes.

In Figure 1, we show a fragment of the Entity-Relationship (ER) model of the guideline's clinical actions. For simplicity, in the figure we use a standard ER diagram, augmented with the use of transaction time (Ts and Te stand for the start and the end of the transaction time respectively) and valid time (Vs and Ve stand for the start and the end of the valid time respectively). The use of four atomic-valued timestamp attributes to represent bitemporal chronons is derived from the TSQL2 representational approach [17]. A more accurate treatment of temporal aspects at the conceptual level could be obtained using, e.g., ST-USM [21]; however, such a conceptual treatment is outside the goals of this paper.

Notice, in particular, that all the entities and the relationships have a transaction time, since we need to model the full history of the evolution of the guideline into the database. The valid time associated with the "CLINICAL_ACTION" entity models the time when the action is to be executed, expressed as a temporal distance from the beginning of the execution of the guideline to which the action belongs[1].

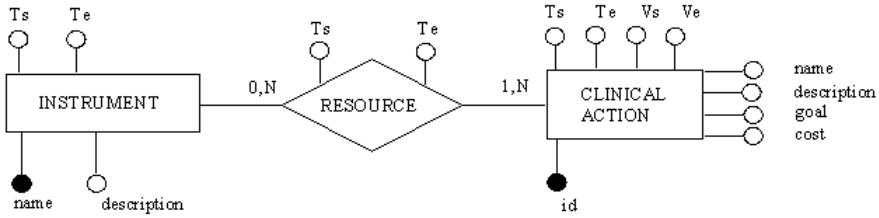


*Figure 1: a fragment of the conceptual model for the guideline for suspected acute pulmonary embolism.*

In Figure 2 we show three relations modeling such a fragment of a conceptual model. The transaction time start (i.e., 2/20/2001, i.e., chronon 0) denotes the timestamp when the tuples were entered into the database (i.e., when the guideline was acquired). In the relation CLINICAL_ACTION, Vs is 0 to denote the fact that the action has to be executed as soon as the guideline about suspected acute pulmonary embolism is started.

INSTRUMENT

| **Name** | **Description** | **TT** |
|---|---|---|
| VQS | ventilation perfusion scintigraphy | [0,UC] |

RESOURCE

| **INSTR_name** | **Action_ID** | **TT** |
|---|---|---|
| VQS | 101 | [0,UC] |

CLINICAL_ACTION

| **Action_ID** | **name** | **Description** | **goal** | **cost** | **T** |
|---|---|---|---|---|---|
| 101 | pulmonary embolus detection | detection by imaging techniques | diagnosis of pulmonary embolism | 100€ | [0,UC] × [0s,3000s] |

*Figure 2: three relations modeling the fragment of conceptual model in Figure 1.*

In 2002, the guideline was modified to reflect the availability of a more sophisticated tool to detect pulmonary emboli: the computed tomographic pulmonary angiography (CTPA). Although the estimated cost of the action increases to about 300 €, the use of CTPA has several advantages: the execution time is much shorter (about 15 seconds), CTPA is relatively more available than VQS and, moreover, it is advantageous in terms of sensitivity, specificity, positive predictive value, negative predictive value and accuracy [20].

[1] It is worth noticing that, in this example, we adopt a non-standard notation for the valid time, since we represent it as a displacement from the starting point of the guideline instead of a displacement from a standard reference point (such as, e.g., the birthday of Christ in the Gregorian calendar) as, e.g., in BCDM.

Now we show how the guideline can be updated through a session of cooperative work in which the proposers P1, P2, P3 and P4 propose some updates and the evaluators E1 and E2 incrementally accept\reject such proposals. While the previous part of the example is real (although, for the sake of brevity, simplified), the working session we describe below is hypothetical, and aims at presenting the different possible operations of proposers and evaluators. The working session is introduced as a sequence of steps (figure 3).

*Step 1*. Proposer P1 proposes to insert CTPA into the INSTRUMENT relation.

*Step 2*. Evaluator E1 accepts the proposal of P1.

*Step 3*. Proposer P2 proposes:

*(1)* to update the relation RESOURCE to store the fact that now CTPA is the instrument to be used for pulmonary embolus detection.

*(2)* to update the relation CLINICAL_ACTION to modify the cost of action 101 (pulmonary embolus detection) from 100 € to 1000 €.

*Step 4*. Proposer P3 proposes:

*(1)* to update the relation RESOURCE as suggested by P2.

*(2)* to update the relation CLINICAL_ACTION to modify the cost of action 101 (pulmonary embolus detection) from 100 € to 300 € and its duration from 3000 seconds to 60 seconds.

*Step 5*. Proposer P4 proposes:

*(1)* to update the relation RESOURCE, as suggested by P2.

*(2)* to update the update proposal about the relation CLINICAL_ACTION issued by Proposer P3 to modify the duration of action 101 from 60 seconds to 15 seconds.

*Step 6*. Evaluator E2 asks a query about the current proposals about resources, time and costs to perform pulmonary embolus detection (action 101).

*Step 7*. Evaluator E2 accepts the proposal issued at step 5.

# 4 Goals, methodology, and main results

Given the wide diffusion and increasing relevance of proposal vetting, as well as the deep impact it has on *data semantics*, we strongly believe that, once again, a *general* solution is needed here.

### 4.1 Main challenges

Coping with proposal vetting in the relational context involves addressing many new challenges.

First of all, we have to support two different ***classes of users*** (proposers and evaluators), providing them with suitable manipulation operations.

Entirely ***new operations*** (with respect to relational model and BCDM) must be introduced to allow evaluators to accept or reject the proposals. On the other hand, proposers can propose changes to the current (reference) status of the database, or to other proposals. Proposals may be complex, in the sense that a proposer may suggest a sequence of changes (insertions, deletions, updates) *as a whole*. We indicate this sequence ***macroproposal***. Through a macroproposal, the proposer underlines that some single operations have to be considered as elementary steps of a more complex change at the data level and that the overall change has to be seen as atomic, because it would make no sense to implement only some steps of it. It is worth noting that also elementary changes at the data level (e.g., a single insertion) can be managed in our framework, by issuing macroproposals containing a single operation. Of course, users may issue more than one macroproposals.

The treatment of time is an intrinsic part of coping with the new data model and algebra since the history of the reference database, and of the different proposals, must be supported. This demands for the treatment of transaction time (and, optionally, of valid time of the stored data).

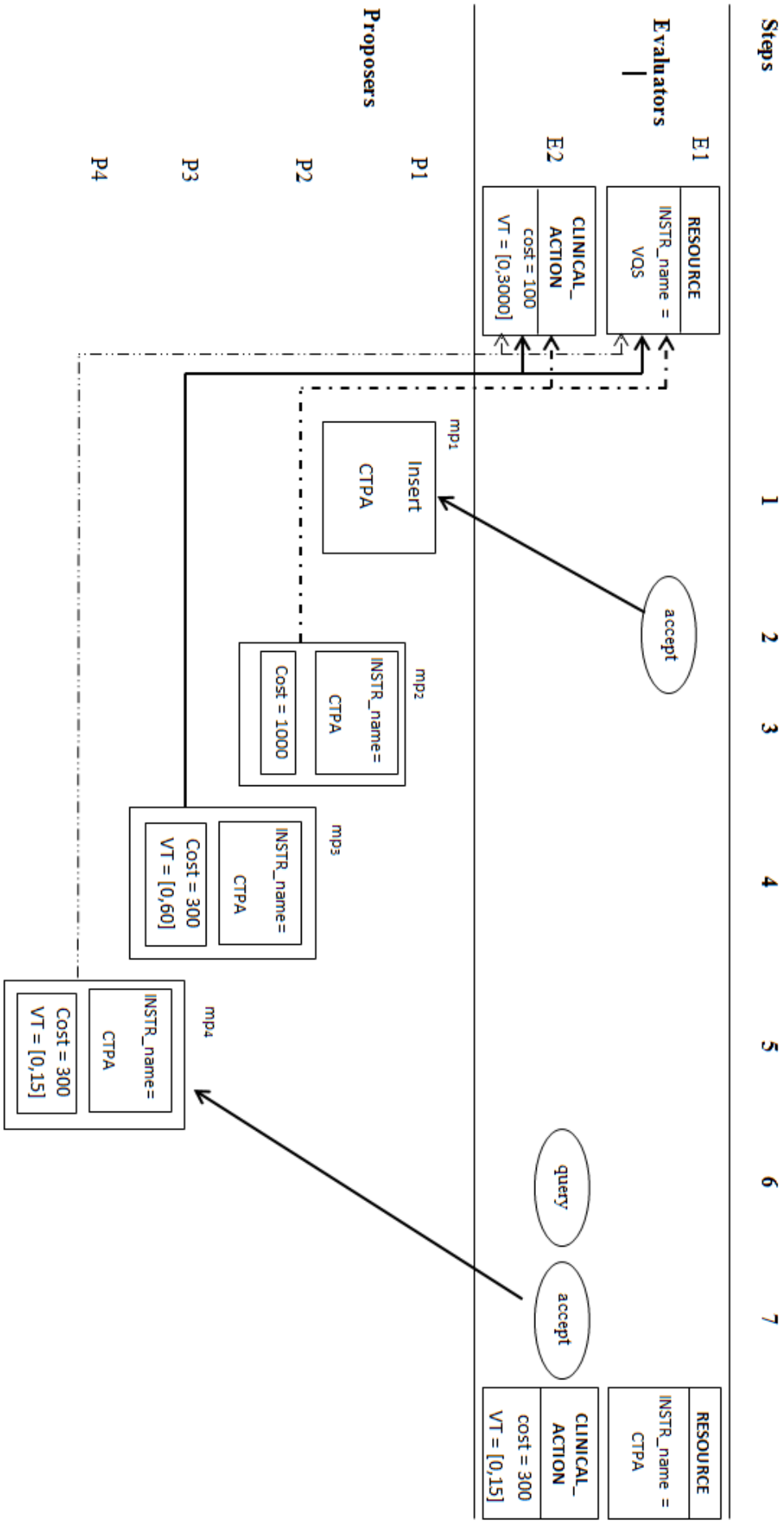


*Figure 3: the sequence of operations in our running example*

Therefore, proposal vetting demands for a substantial ***extension of the data model*** of BCDM. The new data model must be extended to support two types of data: (i) reference data (approved by evaluators) and (ii) proposals (issued by proposers). Querying such a new model requires the definition of a ***new query language***.

The notion of alternatives is an intrinsic part of the proposal vetting phenomenon. Indeed, if users propose different updates of the same tuple, such updates are alternative, because evaluators can accept only one of them. However, to the best of our knowledge no relational data model in literature supports disjunctive pieces of information such as ***alternatives***. Therefore, proposal vetting requires the definition of a new data model and a new semantics for it, allowing one to represent alternative data, and to properly operate on them. As an example of the problems that must be tackled, let us consider the case where an evaluator accepts an update to a given tuple. This fact should automatically trigger the rejection of all the alternative updates to those tuples. Furthermore, since we support macroproposals, also the macroproposals containing the rejected updates must be considered as rejected as a whole (in fact, it would make no sense to allow the execution of only part of them).

Defining a general and principled treatment of alternative proposals in the relational context is one of the main challenges of our approach. It requires a substantial departure from existing SQL and TDB approaches. First of all, in SQL and TDB approaches, updates are usually modeled (at least at the semantic level) as a pair of operations, i.e., a deletion followed by an insertion. Since we aim at modeling directly alternative updates, we introduce a ***primitive semantic notion*** – the *Update-proposal* – to explicitly cope with *disjunctions* of pieces of information and to allow proposers/evaluators to recall what macroproposals are to be interpreted as alternatives. An Update-proposal groups together all the alternative proposals concerning a given tuple (thus resembling, e.g., the notion of *Design Object* in [22]). Defining such a primitive notion also provides several advantages. In fact, it simplifies the definition of ***manipulation*** and ***algebraic operators*** (e.g., it allows to automatically discard all mutually exclusive alternatives, once a specific proposal of update has been accepted).

### 4.2 Methodology

Instead of directly extending a specific TDB model (such as, for instance, TSQL2 [17]), for the sake of generality and clarity, we have chosen to operate at the semantic level, proposing **GPVM** (General Proposal Vetting Model), an extension of BCDM unifying semantic model.

*Consistency* with the relational model and *implementability* are fundamental goals of our approach. In fact, we define our new model as a consistent *upper layer* upon temporal relational models in order to provide proposal vetting facilities. It is worth noticing that the same methodology has been exploited in BCDM, which can be regarded as a consistent upper layer upon standard relational model to support valid and transaction time.

The main original contributions of our approach lie in extending BCDM to support proposals and evaluation of updates. Specifically, we propose:

1. a new ***data model*** to cope with both reference (accepted) and proposed (to be evaluated) data; in particular, we support *alternative* data proposals, while in BCDM relations are defined as *conjunctive* sets of tuples only;
2. new ***manipulation operations*** to provide proposer with the operations of insertion, deletion and update and evaluators with the operations of accept and reject of proposals;
3. new ***algebraic operations*** to query data represented in the extended data model.

**4.3 Main results**

We propose a *general* and *implementable* theoretical support to proposal vetting consistent with the relational model. In fact, we have proved that:

i. GPVM data model is *reducible* to BCDM one (see Section 5);
ii. GPVM manipulation operations are a *"proposal vetting" consistent extension* to BCDM ones (see Section 6);
iii. GPVM algebraic operations are *reducible* to BCDM ones (see Section 7).

By proving properties i-iii, we grant that our approach can be added on top of any of the relational TDB approaches grounded on the BCDM semantics. This fact enhances the generality of our work, as well as its implementability (see also Section 8). Concerning implementability, it is worth noting that Oracle Database, since version 10g, supports both transaction time and valid time consistently with BCDM [23].

# 5. Extending the data model

To cope with the issues outlined above, in our data model we need to distinguish between accepted data and proposals that still need to be evaluated.

To this end, we introduce a two-layered approach. In it (1) we define two categories of users: a set of proposers, who issue proposals, and a set of evaluators, who can accept/reject them. Moreover, (2) we split the data in two levels: evaluator data level and proposer data level.

The two categories of users are formally defined as below:

**Definition 5.0.1: Proposers and Evaluators**. We term Proposers=$\{p_1,\ldots,p_y\}$ and Evaluators=$\{e_1,\ldots,e_z\}$ the sets of proposers and evaluators respectively. ♦

Our approach is independent of whether Proposers and Evaluators are disjoint sets or not (so that different policies can be implemented).

As for the two data levels, we split our database into two parts: DB_Reference and DB_Proposers. DB_Reference is a set of relations meant to maintain all validated data, accepted by evaluators. Current (i.e., not deleted, having UC as the transaction time end, see section 2 on BCDM) data in the evaluator data level constitute the reference (accepted) version of data.

On the other hand, we store all the proposals, generated by any proposer, in DB_Proposers. DB_Proposers maintains three sets of proposals (i.e., proposals of insertion, deletion and update with respect to the content of DB_Reference). Proposals supposed to be evaluated as an indivisible set are grouped into a single macroproposal, as explained in Section 4.1. The two definitions below formalize these concepts.

**Definition 5.0.2: Macroproposals**. We term Macroproposals={$id\_mp_1$,…,$id\_mp_{x...}$} the set containing the identifiers of the macroproposals. ♦

**Definition 5.0.3:** We define a database as a pair **<DB_Reference, DB_Proposers>**. DB_Reference is a set of relations {$r_1(R_1)$,...,$r_k(R_k)$} where $r_i$ ($1 \le i \le k$) is an instance of the schema $R_i$. DB_Proposers contains the following sets[2]:

**pi($r_i$)**, containing proposals of *insertion* into $r_i \in$DB_Reference,

**pd($r_i$)**, containing proposals of *deletion* of tuples in $r_i \in$DB_Reference,

**pu($r_i$)**, containing proposals of *update* (concerning tuples in $r_i$, $pi(r_i)$ and $pu(r_i)$ with $r_i \in$DB_Reference).♦

Both in DB_Reference and in DB_Proposers we deal with different types of *implicit* attributes (see section 2). First, we consider the valid time of tuples and/or their transaction time. Specifically, while transaction time is necessarily always present (since it copes with the history of operations on the database), valid time (which provides the time when the facts represented in the relational tuples hold in the real world) may be required in some relations and not in others. Referring to our running example, the INSTRUMENT relation, which describes the available instruments, is a transaction-time relation (i.e., no valid time is reported). On the other hand, the CLINICAL_ACTION relation, which provides the time when a certain action has to be executed with respect to the guideline start, needs a valid time attribute, to cope with this information.

Moreover, we associate every tuple in DB_Reference with one (or more) elements in the Evaluators set, corresponding to the evaluators who accepted the tuple after a proposal-vetting session. Similarly, we associate all tuples in DB_Proposers with one (or more) elements in the Proposers set. Finally in DB_Proposers we associate the tuples with the respective macroproposal. *Evaluators, proposers* and *macroproposals* are *implicit* attributes as well.

As in BCDM, *value equivalent* tuples are no admitted in the same relation.

A detailed description of the contents of DB_Reference and DB_Proposers is provided in the following subsections. Examples are also given to illustrate the definitions.

[2] In this section, we present an abstract data model; in section 8 we will discuss a possible relational implementation.

### 5.1 DB_Reference

We denote as $T_{eval}$ the attribute with domain Evaluators×$D_{TT}$×$D_{VT}$.

**Definition 5.1.1: DB_Reference.** We denote with R=($A_1$,…,$A_n$|$T_{eval}$) the schema of a relation r∈DB_Reference, with $T_{eval}$ defined as above. (**Condition 5.1.2**): We do not admit value-equivalent tuples in the same relation r∈DB_Reference. ♦

### 5.2 DB_Proposers

In this section, first we briefly introduce the definitions concerning proposals of insertion and of deletion. Then we move to one of the main contributions of our approach, namely the definition of proposals of update.

We denote as $T_{prop}$ the attribute with domain Macroproposals×Proposers×$D_{TT}$×$D_{VT}$.

#### 5.2.1 Proposals of insertion

**Definition 5.2.1.1: pi(r).**

Given a relation r∈DB_Reference with schema R=($A_1$,…,$A_n$|$T_{eval}$), we define pi(r) as the set containing the tuples x which are proposed for insertion into r. The *schema* of pi(r) is R'=($A_1$,…,$A_n$|$T_{prop}$). (**Condition 5.2.1.2**): We do not admit value-equivalent tuples in the same set of proposals of insertion. ♦

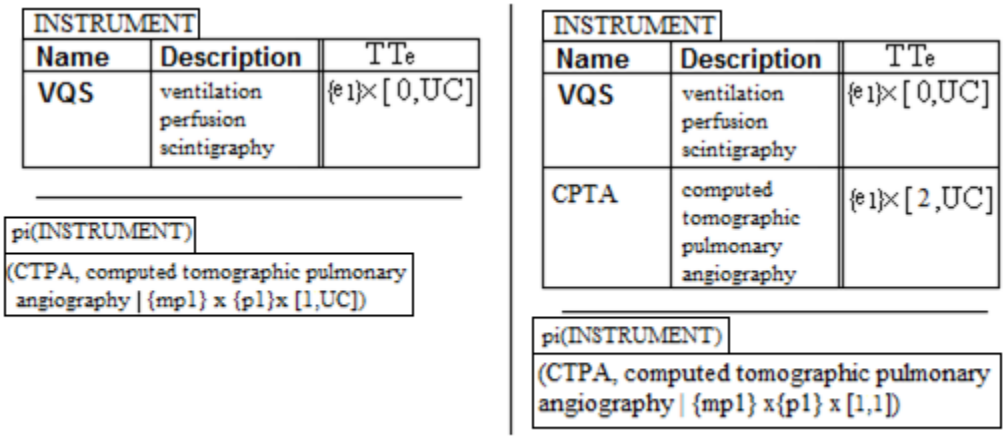


*Figure 4: A representation of our running example after Step 1 (left) and after Step 2 (right). In this figure, we group all macroproposal identifiers and all proposer/evaluator identifiers in the form of a set (e.g., {p1}). We number the macroproposals according to the step at which they were issued (see Section 3) and assume that step* n *occurs at the transaction time* n. *In each figure, the upper part represents the relation INSTRUMENT in DB_Reference and the lower part represents the content of DB_Proposers. In*

*this figure, pi(INSTRUMENT) is reported, which contains a proposal of insertion. Observe that INSTRUMENT is a transaction-time relation; therefore, valid time is missing in the proposals referring to INSTRUMENT.*

**Example 5.1.** Figure 4 reports a proposal of insertion, referring to steps 1 and 2 of our running example (see Section 3). In particular, on the left side of the figure, the content of set pi(INSTRUMENT) at step 1 is reported. The set contains a proposal of insertion referred to relation INSTRUMENT∈DB_Reference. The explicit attributes of the proposal provide the name and the description of the new diagnostic instrument CTPA. The implicit attributes provide the identifier of the macroproposal (*mp1*), the proposer (*p1*) and the transaction time. In particular 1 represents the time at which the alternative has been issued by $p_1$. Notice that, actually, a calendar time should be provided. UC denotes the fact that the proposal is *current*. In this specific proposal, it was not required to represent also valid time .

### 5.2.2 Proposals of deletion

**Definition 5.2.2.1: pd(r).** Given a relation r∈DB_Reference with schema $R=(A_1,\ldots,A_n|T_{eval})$, we define pd(r) as the set containing the tuples x which are proposed for deletion from r. The *schema* of pd(r) is $R'=(A_1,\ldots,A_n|TT_{prop})$, where $TT_{prop}$ has domain Macroproposals×Proposers×$D_{TT}$. (**Condition 5.2.2.2**): We do not admit value-equivalent tuples in the same set of proposals of deletion. ♦

A tuple in pd(r) identifies the tuple in r to be deleted. Since we do not admit value equivalent tuples in DB_Reference relation, the explicit attributes are sufficient to univocally identify the tuple in r to be deleted. Therefore, its valid time is not needed, and it is not stored in pd(r).

### 5.2.3 Proposals of update

Proposals of update record the tuple which should be modified, and the specific changes that should be made to it (i.e., they consist of a pair <old tuple, new tuple>). In principle, we could model each proposal of update independently of the others. However, the underlying semantics we want to assign to our model is that all the proposals of modification concerning the same tuple must be interpreted as mutually exclusive alternatives. In fact the acceptance of one proposal implicitly involves the rejection of all the others (and of the macroproposals containing them; see the discussion in Section 4.1).

On the other hand, a proposer might issue different proposals referring to the same tuple, and store them into different macroproposals (but we explicitly disallow a proposer to propose two alternative updates to the same tuple in the same macroproposal, for the sake of coherency). Observe that this does not mean that the same proposer cannot issue two alternative updates to the same tuple. Simply, s/he is forced to store them into two different macroproposals.

Indeed, we believe it would be quite meaningless to propose two contradictory elementary steps within an atomic, more complex proposal of change to the data.

Following these considerations, we have introduced a *primitive semantic notion* – the *Update-proposal* – to explicitly group all the *alternative* updates concerning the same tuple.

**Definition 5.2.3.1: Update-proposal**. An Update-proposal may concern a tuple x which may be either (i) a tuple in an evaluator-level relation r, or (ii) a proposal – related to an evaluator relation r – already issued by some proposer.

Let $[A_1, \ldots, A_n]$ be the explicit attributes of tuple x. An Update-proposal $up \in pu(r)$ concerning x can be defined as up = $<o, Alt(alt_1,\ldots,alt_m)>$, where $o=x[A_1, \ldots, A_n]$ and $alt_i$ $(1 \leq i \leq m)$ are tuples defined on the schema $(A_1,\ldots,A_n|T_{prop})$. *o* is used in order to identify the tuple *x* to be updated and $Alt(alt_1,\ldots,alt_m)$ is a non-empty set of alternative tuples referring to the tuple *x*, representing the different alternative proposals of update concerning *x*. ♦

**Terminology (type of an Update-proposal).** Given the Definition 5.2.3.1, we call the pair $<(A_1, \ldots, A_n), (A_1, \ldots, A_n|T_{prop})>$ the *type* of the Update-Proposal *up*. ♦

**Terminology (origin, alternatives of an Update-proposal).** Given the Definition 5.2.3.1, we call x the *origin* of the Update-proposal and $\{alt_1,..,alt_m\}$ its *alternatives*. Since *o* is used in order to uniquely identify *x*, in the following, we call both *x* and *o* "*origin*". ♦

**Definition 5.2.3.2: origin(up) and alternatives(up).** Given an Update-proposal *up* = $<o,Alt(alt_1,alt_2,..,alt_m)>$, origin(*up*) = o, and alternatives(*up*) = $\{alt_1,alt_2,..,alt_m\}$.♦

We can finally define the set pu(r) of update proposals.

**Definition 5.2.3.3: Set of Update-proposals pu(r).** Given a relation $r \in DB\_Reference$ with schema $R=(A_1,\ldots,A_n|T_{eval})$, we define pu(r) (henceforth called *set of Update-proposals*) as the set containing the Update-proposals up=$<o, Alt(alt_1,\ldots,alt_m)>$ whose origin o identifies a tuple in r or a proposal related to r, already issued by some proposer. The *type* of pu(r) is $<(A_1, \ldots, A_n), (A_1, \ldots, A_n|T_{prop})>$. (**Condition 5.2.3.4**): Different Update-proposals having the same origin are not admitted in the same set of Update-proposals. (**Condition 5.2.3.5**): We do not admit value-equivalent alternatives to the same origin. ♦

By means of Condition 5.2.3.5, we uniquely identify each Update-proposal in a set of Update-proposals with its origin. Then, proposals of update concerning a preceding Update-proposal $up \in pu(r_i)$ are directly referred to the origin of *up* (which may be either a tuple in r or in pi(r)). This does not mean that we do not admit proposals of update regarding other proposals. Indeed, chains of proposals of update are managed in our framework, even though not *explicitly*. Our solution relates each proposal to the original tuple to be modified (and not to the alternative proposal it directly modifies). This somehow collapses the chain of updates, because we do not explicitly capture the notion of

"what update depends on what other update", as we could have done by storing the dependencies between updates by means of, e.g., a tree structure. However, our choice does not lead to any information loss, as the acceptance of an update in the more complex representation would lead to the very same result, at the evaluator data level, than the corresponding acceptance in the simpler representation we propose in this paper. Moreover, we have proved that, using the more complex representation, the property of uniqueness of model, inherited from BCDM (see Section 2) does no longer hold. This seems to us a relevant drawback, since it would lead to ambiguous representations, i.e., to alternative semantically equivalent representations for the same data content. Additionally, the definition of the algebraic operators on the more complex data model would be far from clear and intuitive. We thus believe that our choice is advantageous in practice.

**Example 5.2**. Considering our running example (see Section 3), figure 5(C) shows the Update-proposals representing all the alternative update proposals issued until step 5. Only the Action_id and Cost explicit attributes in relation CLINICAL_ACTION are reported in the figure, for the sake of brevity. Considering the proposals concerning the CLINICAL_ACTION evaluator tuple "(101, 100)", in the first alternative, the *Cost* explicit attribute is changed from 100 to 1000. As for the implicit attributes, *mp2* identifies the macroproposal the update belongs to; *p1* identifies the proposer; 4 is the transaction time start (i.e., the time at which the alternative has been issued by $p_1$), and UC is the transaction time end (i.e., the proposal is current); 0 is the valid time start, and 3000 is the proposed valid time end, expressed in seconds. Observe that figure 5(C) groups all the alternative update proposals issued with respect to the evaluator tuple "(101, 100)". In particular, the second and third alternatives don't differ for the explicit attribute values, but for the proposed valid time. Moreover, despite the fact that the third alternative was issued as an update to the second alternative (see Section 3), all the update chaining is not explicitly maintained in our framework (but both proposals are referred to the same origin, i.e., the original evaluator tuple they aim at modifying).

It is also worth noting that the following key result holds:

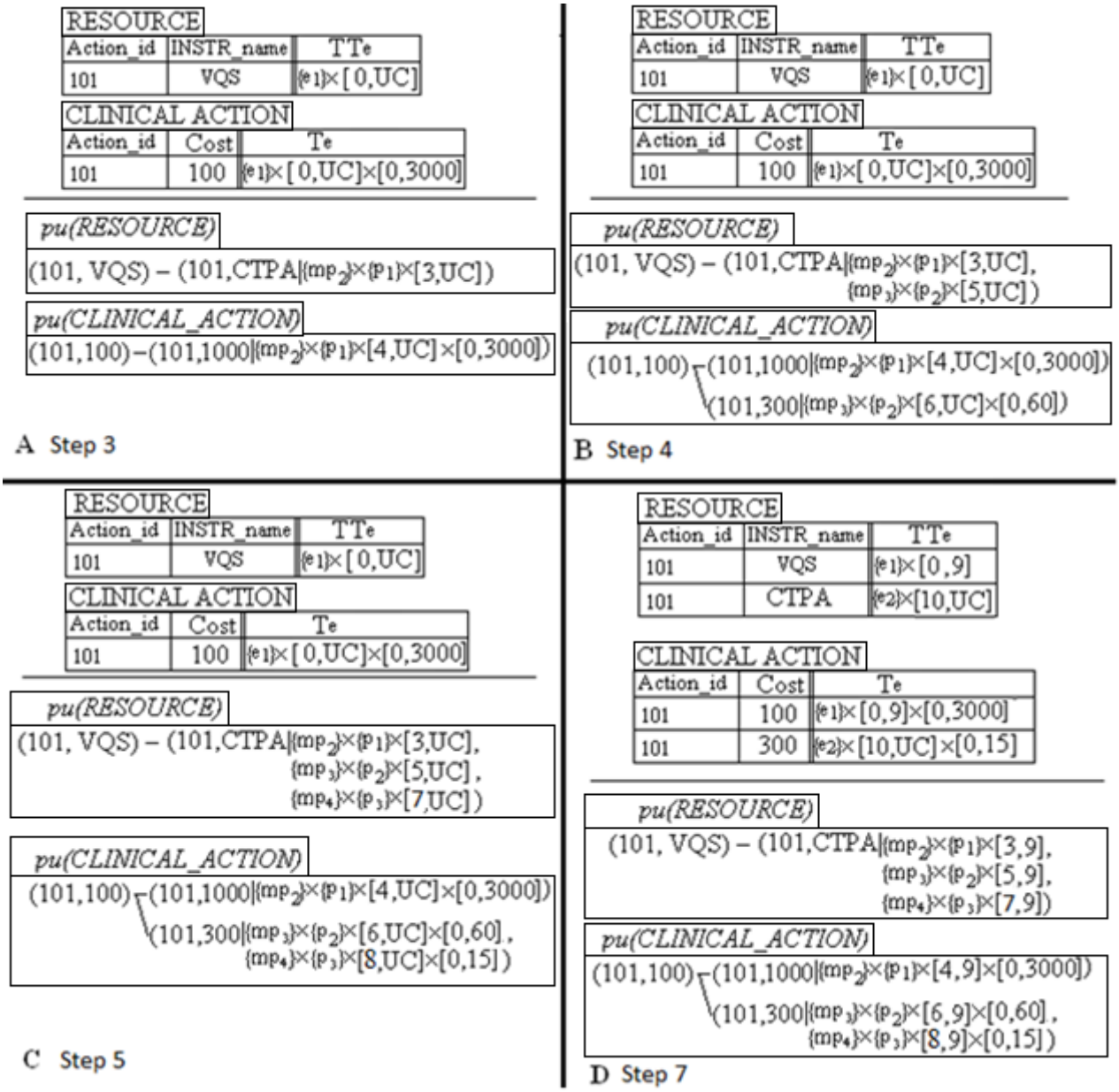


*Figure 5: A representation of our running example after Step 3 (A), after Step 4 (B), after Step 5 (C), and after Step 7 (D). In each subfigure, the upper part represents the relations RESOURCE and CLINICAL_ACTION in DB_*Reference *and the lower part represents the content of DB_Proposers. Both pu(RESOURCE) and pu(CLINICAL_ACTION) contain an Update-proposal, represented – for the sake of readability – as a two-level tree. The origin of the Update-proposal is the root (on the left) (e.g., (101,100) is the origin of Update-proposal in pu(CLINICAL_ACTION)) and the alternatives are its children (on the right).*

**Property 5.2.3.6: Reducibility of GPVM data model to BCDM data model**. The GPVM data model reduces to the BCDM data model in case no proposals are proposed/evaluated. ♦

The property of reducibility to BCDM holds, since the pair <DB_Reference, DB_Proposers> trivially reduces to a BCDM database in case we take into account only one level of data (i.e., DB_Reference), and we disregard evaluators'

information. This case models the "non-proposal vetting" context in which users can directly operate insert/delete/update operations on the data and no evaluation is needed.

# 6 Manipulation Operations

In this section we define the manipulation operations of GPVM. In particular, in our model we introduce two levels of operations: proposer operations and evaluator operations. As regards proposer operations, we allow to define macroproposals; on the other hand, evaluators can either accept or reject macroproposals.

## 6.1 Proposer operations

Proposer operations allow a proposer to define a macroproposal, i.e. to suggest a set of changes (insertions, deletions, updates) to DB_Reference as a whole. We define the operation that allows the user to define a macroproposal as below:

**Definition 6.1.1 create_macroproposal.** The operation to create a macroproposal is defined as:

*create_macroproposal*($op_1 \ldots op_n$, $p_{new}$)

where the arguments are an ordered set of operations ($op_1 \ldots op_n$) and the proposer who suggests them ($p_{new} \in$Proposers). ♦

We allow three types of operations: (i) proposal of insertion, (ii) proposal of deletion, and (iii) proposal of update (propose_update). Storing a proposal of insertion usually leads to the insertion of a new tuple in the set of proposals of insertions. Sometimes it simply requires an update of the implicit attributes of an existing proposal (if such an old proposal is value equivalent to the new one). A proposal of deletion operates similarly.

In the following, we will concentrate on proposals of update, since they are the most complex operation.

**Definition 6.1.2: propose_update.** Given a relation r∈DB_Reference with schema $R=(A_1,\ldots,A_n|T_{eval})$, let $<(A_1,\ldots,A_n), (A_1,\ldots,A_n|T_{prop})>$ be the type of pu(r). We define propose_update as follows:

*propose_update*($r$, ($a_1,\ldots,a_n$), ($a_1'$,...,$a_n'$), $t_{vt\_new}$)

where the arguments are the DB_Reference relation $r$, which contains the tuple to be updated, the explicit attributes of the tuple to be updated ($a_1,\ldots,a_n$), the newly proposed explicit attributes ($a_1'$,…,$a_n'$), and the newly proposed valid time ($t_{vt\_new}$).♦

Note that in the proposal of update the proposer has to specify only the explicit attributes of the tuple to be updated. Indeed, they are enough to univocally identify it. We interpret a macroproposal as an *indivisible* set of proposals. To this

end, its definition is embedded within a DBMS transaction [24]. This choice grants for the maintenance of the well-known "ACID" (i.e., atomicity, consistency, isolation and durability) transaction properties.

**Example 6.1** The macroproposal at Step 3 by proposer $p_2$ can be expressed as follows:

```
create_macroproposal(
    propose_update(RESOURCE,(101,VQS), (101,CTPA)),
    propose_update(CLINICAL_ACTION,(101, 100), (101, 1000),[0,3000]),
    p2)
```

In the first propose_update, proposer $p_2$ proposes to update the tuple identified by the explicit attributes (101,VQS) in the RESOURCE relation as follows: (101,CTPA). Note that the valid time is not required in this case: since the relation RESOURCE in DB_Reference is a transaction-time relation. In the second propose_update, proposer $p_2$ proposes to update the tuple identified by the explicit attributes (101,100) in the CLINICAL_ACTION relation as follows: (101,1000). Moreover the proposed valid time is summarized by the interval [0,3000]. Observe that the proposer parameter ($p_2$ in the example) is only provided once (implicitly it applies to all the operations in the macroproposal).

Macroproposals need to be stored, waiting for evaluation. Storing a macroproposal consists in storing all its proposals. In our approach we do not control whether the operations are *valid* with respect to the current status of DB_Reference (i.e., if all the operations, if accepted, could be successfully executed to update the current status of DB_Reference). We only check their *admissibility* (i.e., if they are not redundant or contrasting). The control of validity is postponed at execution time (i.e., at the time the macroproposal is accepted), since the state of DB_Reference can evolve between macroproposal definition and macroproposal acceptance. In this way, our approach is more flexible and some unnecessary check , which would become obsolete if the database state changes, are avoided.

Admissibility is checked in order to verify that some minimum data consistency criteria are met: redundant and incoherent situations should be avoided. Moreover, the technical choices we made when defining the data model (see Section 5.2.3) have to be respected.

Specifically, a macroproposal is *admissible* if its set of proposals does not contain: (i) two or more deletions of the same tuple, (ii) two or more insertions of value equivalent tuples, (iii) two or more alternative updates to the same DB_Reference tuple.

Condition (i) avoids redundant (and useless) deletion proposals.

Condition (ii) avoids redundancy as well. Moreover, it forbids the proposal of insertion of two or more tuples sharing the explicit attributes, but with a different valid time. Observe that such kind of insertions would be admissible if issued within different macroproposals.

Condition (iii) explicitly disallows a proposer to propose two or more alternative updates to the same tuple in the same macroproposal, for the sake of coherency, as already discussed in Section 5.2.3.

In the case that conditions (i-iii) hold, the macroproposal is admissible; only in this situation, it is assigned a unique identifier $mp_i$, and all its operations are stored in DB_Proposers.

For what concerns the storage of a proposal of update, we have two cases: (i) there is no Update-proposal in pu(r) having as an origin the tuple the proposer wants to modify; or (ii) an Update-proposal having such an origin is already present in pu(r).

In the first case, *propose_update(r, $(a_1,...,a_n)$, $(a_1',...,a_n')$ , $t_{vt\_new}$)* creates a new Update-proposal in pu(r), whose origin is $(a_1,...,a_n)$ and whose only alternative is $(a_1',...,a_n')$. As for its implicit attributes, the proposer is proposer who issued the create_macroproposal, the valid time value is $t_{vt\_new}$, and the current transaction time and the macroproposal identifier $mp_i$ are provided by the system.

In the second case, *propose_update(r, $(a_1,...,a_n)$, $(a_1',...,a_n')$, $t_{vt\_new}$)* modifies an Update-proposal already present in pu(r). $(a_1,...,a_n)$ is the origin of this Update-proposal to be considered. We can then further distinguish between two situations. If no value equivalent alternative, with respect to the newly proposed one, exists in this Update-proposal, a new alternative is added. Specifically, its explicit attributes are $(a_1',...,a_n')$. As for its implicit attributes, they are set as above. On the other hand, if the identified Update-proposal already contains an alternative which is value equivalent to the newly proposed one, only its implicit attributes are properly updated. This allows to account for the new macroproposal identifier, the new proposer, the new valid time and the new transaction time.

**Example 6.1 (continued)**. In the example above, since the two propose_update operations fulfill the conditions described above (in particular condition (iii)), the macroproposal can be stored in DB_Proposers as shown in Fig 4(A): a new Update-proposal is inserted in pu(RESOURCE) and a new Update-proposal is inserted in pu(CLINICAL_ACTION). The new Update-proposal in pu(RESOURCE) has the origin defined by the propose_update second parameter while the alternative explicit attributes are taken from the propose_update third parameter (101,CTPA). For what concerns implicit attributes, the proposer ($p_1$) is taken from the create_macroproposal parameter, the transaction time ([3,UC]) and the macroproposal identifier ($mp_2$) are provided by the system. The new Update-proposal in pu(CLINICAL_ACTION) has the origin defined by the propose_update second parameter (101,100), while the alternative explicit attributes (101,1000) are taken from the propose_update third parameter. For what concerns implicit attributes, the proposer ($p_1$) is taken from the create_macroproposal parameter, the valid time value ([0,3000]) is taken from the propose_update fourth parameter , the transaction time ([4,UC]) and the macroproposal identifier ($mp_2$) are provided by the system. The situation after storing macroproposal $mp_2$ is shown n Fig. 4(A).

### 6.2 Evaluator operations

In our approach, evaluators can reject or accept macroproposals stored in DB_Proposers. Thus, we provide them with the operation *reject* and the operation *accept*. Both operations have the same input: the macroproposal indentifier, on which the operation will be performed, and the evaluator, who evaluates the macroprosal.

Accepting or rejecting a macroproposal means to accept or to reject all of its operations, in an atomic way, and it is not possible to accept or to reject just a subset of the macroproposal operations. To this end, the acceptance and the rejection of a macroproposal are embedded within a DBMS transaction [24].

First, we present the reject operation.

**Definition 6.2.1: reject.** We define the reject operation as *reject(mp, eval)*, where *mp* ∈ Macroproposals and *eval* ∈ Evaluators; *mp* indentifies the macroproposal, while *eval* identifies the evaluator who wants to make the rejection. ♦

The rejection of a macroproposal *mp* consists of the rejection of all its operations (i.e., all the proposals of insertion, deletion, and update which are contained in *mp*). The rejection of a proposal consists in its deletion. However, since we want to retain the whole database history, including the history of macroproposals, such proposals are not physically deleted from DB_Proposers. Conforming to BCDM, they are logically deleted (see section 2). Thus, all operations in *mp* are "closed" (i.e., UC as a transaction time has to be removed from the implicit attributes, and has to be substituted with the time of rejection).

See for example Fig. 4(D) where the transaction-time end is set to "9" for the alternatives of the Update-proposal in the pu(RESOURCE)).

**Definition 6.2.2: accept.** We define the accept operation as *accept(mp, eval)*, where *mp* ∈ Macroproposals and *eval* ∈ Evaluators; *mp* indentifies the macroproposal, while *eval* identifies the evaluator who wants to make the acceptance. ♦

The acceptance of a macroproposal is used by evaluators to make a given macroproposal effective, i.e., to execute all the proposals it contains on the DB_Reference. Since in our approach a macroproposal must be interpreted as an *indivisible* set of proposals, if even only one of such proposals cannot be executed, the execution of all the operations in the set has to be stopped / rolled back.

**Example 6.2.** The acceptance of macroproposal $mp_4$ issued at Step 7 by evaluator $e_2$ in the running example consists in the operation *accept*($mp_4$, $e_2$) .It means to accept every operation in $mp_4$.

The acceptance operation will be performed as:

```
begin_transaction
    accept(propose_update(RESOURCE,(101,VQS), (101,CTPA)),e2),
```

```
    accept(propose_update(CLINICAL_ACTION,(101, 100), (101, 1000),[0,3000]),e2)
end_transaction
```

The macroproposal $mp_4$ contains two proposals of update, which are accepted. We embed the two accept operations within a DBMS transaction.

Operatively, we check if each proposal in the macroproposal can be executed, by following the order in which they are listed. Each checked proposal is immediately executed (even if, as observed above, rollback may be later required).

A proposal of insertion can be executed if there are no current value equivalent tuples to it in the DB_Reference relation r at hand. In this case, we insert a new tuple in r, where the explicit attributes and the valid time are set to the values specified in the proposal, the evaluator is taken from the accept operation parameter, and the transaction time is set (current) by the system. Otherwise, if a current value equivalent tuple with respect to the proposed one already exists in r, two cases are possible: (i) such a tuple shares the same valid time of the proposed one, and (ii) such a tuple has a different valid time with respect to the proposed one. In case (i), the state of DB_Reference conforms the intended meaning of the proposal (i.e., that a tuple with the explicit attributes and valid time at hand is current in DB_Reference). In this situation, we perform no operations and the execution of the macroproposal can continue. Otherwise, in case (ii), the state of DB_Reference is not consistent with the goal of such a proposal (indeed, only an update to the existing DB_Evaluator tuple would be acceptable), and the operation (and therefore the whole macroproposal) fails. If the execution of the proposal of insertion is successful, the tuple in pi(r) is closed, since such a proposal is no longer active.

A proposal of deletion can always be executed. Indeed, two cases are possible: (i) the tuple is current in *r*, thus the execution of its deletion means to close such a tuple; (ii) the tuple is not current, thus no actions are required (in fact, the database state is already coherent with the deletion goal).

A proposal of update can be executed if there are no current tuples in the DB_Evaluator relation r value equivalent to the chosen alternative (see third parameter in propose_update). The execution of a proposal of update consists in the execution of a deletion followed by the execution of an insertion in an atomic way, as in BCDM. In our case, the operation of deletion refers to the origin of the Update-proposal (see second parameter in propose_update) and the insertion refers to the selected alternative. The explicit attributes and the valid time are set to the values specified in the proposal, the evaluator is taken from the accept operation parameter, and the transaction time is set (current) by the system. If the execution of the proposal of update is successful, we close all the alternatives in the Update_proposal at hand (i.e., both the selected one and the mutually exclusive others). Moreover, the rejection of the mutually exclusive alternatives with respect to the accepted one implies the rejection of the entire macroproposals they belong to. This rejection is implemented as a logical deletion, i.e., the tuple is not physically removed, but all bitemporal chronons (UC, $c_v$), where $c_v$ is any valid-time chronon, are deleted from the tuple itself (as in BCDM).

**Example 6.2 (continued).** Fig 4(D) shows the result of execution of these operations. The acceptance of propose_update(RESOURCE,(101,VQS)) at step 7 consist in the logical deletion of tuple (101,VQS) in the relation RESOURCE (the transaction time end is set to 9), and in the insertion of the new tuple (101, CTPA | $\{e_2\} \times [10,UC]$) in RESOURCE. Such acceptance produces the rejection of the macroproposal $mp_2$ and $mp_3$ issued at Step 3 and Step 4 respectively, since they have current alternatives to the accepted Update-proposal. Note that all operations in $mp_2$ and $mp_3$ are "closed" (i.e., UC as a transaction time end is be removed, and is set to 9). Similar actions are performed for the acceptance of propose_update(CLINICAL_ACTION,(101, 100)). The tuple (101,100) is logically deleted and a new tuple (101, 300| $\{e_2\} \times [20,UC] \times [0,15]$) is inserted and its alternatives in Update-proposal are closed. Moreover, the macroproposals, which are alternative to the macroproposal $mp_4$, i.e. the macroproposals, are rejected.

### 6.3 Properties of manipulation operations

In most cases (consider, e.g., TSQL2) temporal approaches have been devised in such a way to be a consistent extension of conventional ones [17]. This guarantees their interoperability with pre-existent approaches. Since our approach is an extension of BCDM, we might aim at providing a consistent extension of BCDM manipulation operators. However, GPVM manipulations operations cannot be a consistent extensions of BCDM, since in the GPVM context direct insertion/deletion/update operations are not supported.

Nevertheless, we have developed our approach so that the following (less strict than consistent extension) property holds:

**Property 6.3.1**: **"General Proposal vetting" consistent extension of BCDM**. If all users are both evaluators and proposers, our model is a "general proposal vetting" consistent extension of the BCDM model (considering only data in DB_Reference, and neglecting the "Evaluator" implicit attribute). ♦

As a matter of fact, in our approach, we can perform each manipulation operation $Op^B$ in BCDM as an atomic pair of operations <create_macroproposal; accept>. In this pair, the macroproposal contains only the proposal corresponding to $Op^B$ . The accept operation refers to such a macroproposal. The result we obtain is the same we would have in BCDM, if we just focus on DB_Reference.

## 7 Relational Algebra

Besides manipulation operation, we had to provide also query operators, in order to support the possibility of querying data, selecting and joining them. This helps evaluators in taking their acceptance/rejection decisions, as well as proposers in proposing updates to data. For example, selection can be used in order to focus the attention only on a subset of proposals that satisfy some conditions. Moreover, the data of interest can be stored into more than one relation.

Therefore, join operations can be useful for proposers and evaluators, e.g., in order to have a global view of all the data concerning the same action.

For instance, in step 6 of the running example, evaluator E2 requires joining the relations RESOURCE and CLINICAL_ACTION, followed by a selection on the clinical action "pulmonary embolus detection". This allows to "reconstruct" proposals concerning the resources and the clinical actions regarding pulmonary embolus detection. Additionally, since the evaluator requires that only current proposals are taken into account, also a form of temporal selection on the transaction time is involved in the query at step 6.

Since in this paper we operate at the semantic level, the query language for our extended data model is provided at the algebraic level, as an extension of BCDM temporal algebra.

For the sake of brevity, we do not report the exhaustive listing of all our extended algebraic operators and we focus on operators on sets of Update-proposals. However, it is worth mentioning that in our extended algebra we also provide: (i) operators on DB_Reference relations, (ii) extended versions of algebraic operators to cope with "mixed" cases in which sets/relations have different types (e.g., natural join between a set of Update-proposals and DB_Reference relations); (iii) slicing operators, that remove proposers, macroproposals, valid time and/or transaction time; (iv) temporal selection operators ($\sigma_t$). In all cases, we follow the general methodology applied in the TDB area. In particular, we define our operators so that they behave as the standard relational non-temporal operators on non-temporal attributes, and we use set operators on the temporal components (e.g., intersection for Cartesian product and union for relational union).

**Example 7.1.** The query at Step 6 in the Example can be expressed as follows:

$\sigma_t{}^{P}{}_{TT=UC}(pu(RESOURCE) \bowtie^{P} \sigma^{P}{}_{Action\_id=101}(pu(CLINICAL\_ACTION)))$

where $\bowtie^{P}$ is a natural join between the sets of Update-proposals RESOURCE and CLINICAL_ACTION, $\sigma^{P}$ is the selection operator on non-temporal attributes for selecting the action "pulmonary embolus detection", and $\sigma_t$ is the temporal selection operator, used to select only current tuples.♦

## 7.1 Relational Algebra on sets of Update-proposals

The treatment of proposals demands for the definition of new algebraic operators operating on sets of Update-proposals. As an example, we present here the natural join operator on sets of Update-proposals. We characterize the output of natural join as a set of Update-proposals z of the general form <origin(z), alternatives(z)>, that can be defined by alternative cases. In the formula, we assume the standard "nesting" policy for the scope of the variables in the conditions.

**Definition 7.1.1: natural join $\bowtie^P$.** Given the sets of Update-proposals $s_1$ and $s_2$ corresponding to relations $r_1 \in$ DB_Reference and $r_2 \in$ DB_Referencewith schema $(A_1,\ldots,A_n, B_1,\ldots,B_m \mid T_{eval})$ and $(A_1,\ldots,A_n, C_1,\ldots,C_k \mid T_{eval})$ respectively, let the types of $s_1$ and $s_2$ be $<(A_1,\ldots,A_n,B_1,\ldots,B_m), (A_1,\ldots,A_n, B_1,\ldots,B_m \mid T_{prop})>$ and $<(A_1,\ldots, A_n, C_1, \ldots, C_k), (A_1,\ldots,A_n, C_1,\ldots,C_k \mid T_{prop})>$ respectively. Natural join $\bowtie^P$ provides as an output a set of Update-proposals defined as follows (let A stand for $A_1,\ldots,A_n$, B for $B_1,\ldots,B_m$ and C for $C_1,\ldots,C_k$):

$s_1 \bowtie^P s_2 = \{ <\text{origin}(z),\text{alternatives}(z)> :$

**if** $\exists up_1 \in s_1, \exists up_2 \in s_2 : \text{origin}(up_1)[A]=\text{origin}(up_2)[A] \wedge \exists alt_1 \in \text{alternatives}(up_1), \exists alt_2 \in \text{alternatives}(up_2)$:

$alt_1[A]=alt_2[A] \wedge alt_1[T_{prop}] \cap alt_2[T_{prop}] \neq \emptyset$ **then**

origin(z)[A]← origin($up_1$)[A]; origin(z)[B]← origin($up_1$)[B]; origin(z)[C]← origin($up_2$)[C]

alternatives(z) = {alt :

**if** $alt_1[T_{prop}] \cap alt_2[T_{prop}] \neq \emptyset$ **then**

alt[A] ←$alt_1$[A]; alt[B] ←$alt_1$[B]; alt[C] ←$alt_2$[C]

alt[$T_{prop}$] ←$alt_1[T_{prop}] \cap alt_2[T_{prop}]$ }} ♦

The result of natural join on Update-proposals is a set of Update-proposals built as follows. Two Update-proposals $up_1$ and $up_2$ with origins value-equivalent on the common attributes $A_1,\ldots,A_n$ are merged into one Update-proposal having as origin the standard natural join of the origins. The alternatives of the new tuple are built by performing the standard natural join on the explicit attributes and the intersection of the implicit attributes. Only if this intersection is not empty the alternative is stored as an output.

For Update-proposals, we also define the temporal selection operator. It allows one to select tuples that satisfy a temporal selection predicate.

**Definition 7.1.2: temporal selection $\sigma_t^P{}_\varphi$.** Given a set of Update-proposals s with type $<(A_1,\ldots,A_n), (A_1,\ldots,A_n \mid T_{prop})>$, temporal selection $\sigma_t^P{}_\varphi$ provides as an output a set of Update-proposals over the type $<(A_1,\ldots,A_n), (A_1,\ldots,A_n \mid T_{prop})>$ defined as follows (let T stand for the bitemporal attributes):

$\sigma_t^P{}_\varphi(s)= \{ z : z \in s \wedge \varphi(z[T]) \}$.♦

**Example 7.2.** The result of the query $\sigma_t{}^{P}{}_{TT=UC}(pu(RESOURCE) \bowtie^{P} \sigma^{P}{}_{Action\_id=101}(pu(CLINICAL\_ACTION)))$ is the set of Update-proposals in Figure 6, with type <(Action_id, INSTR_name, name, Description, goal, cost), (Action_id, INSTR_name, Description, goal, cost | $T_{prop}$)>[3].

$$(101,100,VQS) \begin{cases} (101,1000,CTPA \mid \{mp_2\}\times\{p_1\}\times[4,UC]\times[0,3000]) \\ (101,300,CTPA \mid \{mp_3\}\times\{p_2\}\times[6,UC]\times[0,60], \\ \qquad\qquad\qquad\quad \{mp_4\}\times\{p_3\}\times[8,UC]\times[0,15]) \end{cases}$$

*Figure 6: The result of the query in step 6 of the running example. We report only the explicit attributes Action_id, Cost and INSTR_NAME.*

We have defined our algebraic operators on sets of Update-proposals in such a way that the property of reducibility [17] with respect to BCDM algebraic operators holds. Proposals of update cannot be directly modeled within BCDM, mainly due to the fact that they model proposers and macroproposals (as implicit attributes). Thus, the reduction to BCDM involves the choice of a macroproposal and of a proposer. After this choice, each resulting Update-proposal (having just one alternative) can be easily mapped onto a BCDM tuple in a relation with the proper schema. The macroproposal/proposer-slice operator $\eta_{idm,p}$ is used for selecting a specific macroproposal *idm* and proposer *p*.

**Definition 7.1.3: macroproposal/proposer-slice operator on sets of Update-proposals.** Given a set of Update-proposals *s* defined over the type <($A_1$,…,$A_n$), ($A_1$,…,$A_n$|$T_{prop}$)>, the result of the macroproposal/proposer-slice operator $\eta_{idm,p}(s)$ is a BCDM relation defined over the schema ($A_1$,…,$A_n$, $A_1$',…,$A_n$'|T) (where the attributes $A_1$',…,$A_n$' are a renaming of $A_1$,…,$A_n$ respectively) built as follows: each tuple in *s* is examined and, if the value of its implicit attributes match *idm* and *p*, a tuple with the same explicit values and bitemporal chronons becomes a tuple of the result.

$\eta_{idm,p}(s)=\{z : \exists x\in s : z[A]=x[A] \wedge z[T]=\{(t,v) : (idm,p,t,v)\in x[T_{prop}]\} \wedge z[T]\neq\emptyset\}$. ♦

Now we can give the reducibility property on algebraic operators over sets of Update-proposals. This property grants that, if we remove from our approach the treatment of macroproposals and proposers (i.e., if we reduce our approach to the treatment of bitemporal relations only), then our relational algebraic operators behave exactly as BCDM relational algebraic operators. Thus, the reducibility property grants interoperability with BCDM.

**Property 7.1.4: Reducibility of GPVM algebra on sets of Update-proposals to BCDM algebra.** GPVM algebraic operators on sets of Update-proposals are reducible to BCDM algebraic operators. This means that, for each algebraic unary operator $Op^P$ in our model, and indicating with $Op^B$ the corresponding BCDM operator, for each set of Update-proposals *s*, the following holds (the analogous holds for binary operators):

[3] Since the relation Resource has no valid time, for performing natural join, we considered a modified version of the natural join operator in Definition 7.1.1. In it, we consider a set of Update-proposals with transaction time and a set of Update-proposals with both valid time and transaction time, and preserve the valid time in the resulting set of Update-proposals.

$\eta_{idm,p}(Op^P(s)) = Op^B(\eta_{idm,p}(s))$

where *idm* is an arbitrary identifier of a macroproposal and *p* is a proposer in Proposers. ♦

Finally, given the fact BCDM algebraic operators reduce to relational algebra operators [17], also Corollary 7.2.5 trivially holds. It states that if we remove from our approach both the treatment of macroproposals and of proposers, and the treatment of temporal information, then our relational algebraic operators behave as standard relational algebraic operators.

**Corollary 7.1.5: Reducibility of GPVM algebra to relational algebra.** The GPVM algebraic operators are reducible to relational algebra operators. ♦

## 8 Implementation

As a proof of concept, we have developed a prototypical implementation of our approach [25]. Since GPVM *data model* and *algebra* are ***reducible*** to the BCDM one, and since GPVM *manipulation operations* are a ***consistent extension*** to BCDM ones, we have developed our prototype as an upper layer on a relational TDB grounded on the BCDM semantics. In particular, our prototype is implemented on top of TIMEDB [26], a TSQL2-like database based on BCDM semantics. TIMEDB is implemented in JAVA [27] and supports both IBM Cloudscape 10 [28] and Oracle 10g [23]. Our prototype is implemented in PHP 5 [29] and stores data in Oracle 10g. The architecture of our prototype – focusing on the treatment of Update-proposals – is shown in figure 7.

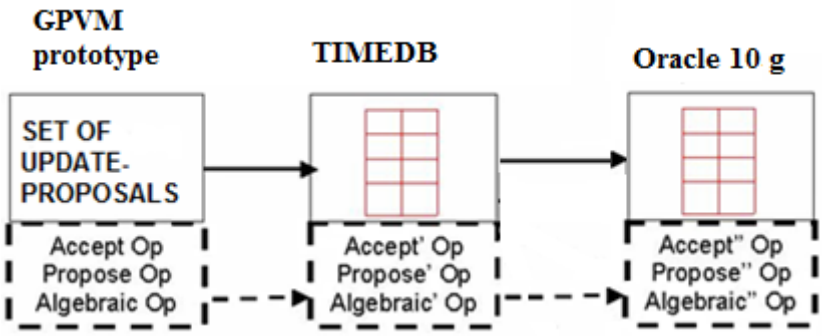


*Figure 7: the architecture of our prototypical implementation.*

In our prototype, we have mapped the GPVM data model to TIMEDB structures. First, a set of Update-proposals has been converted into a set of tuples in TIMEDB (i.e., a set of tuples in the BCDM model). Observe that these tuples still need to be interpreted as "disjunctions" (while in a standard BCDM relation in TIMEDB they would be interpreted as conjunctions). It is the role of the underlying operations (properly converted to TSQL2 in our upper layer) to provide the correct interpretation. To use the object-oriented programming terminology, we could say that the manipulation and

algebraic operations, working at the BCDM semantic level, act as methods operating on an object, which is similar, but not identical to a BCDM relation (since its tuples may be in disjunction). By means of these methods, users can correctly manipulate or query the data, having preserved the correct semantics.

The operations are taken in input via a simple web interface (i.e., a HTML [30] page). Then, a parser manages the input. In case no proposal vetting facility is used, thanks to the reducibility and consistent extension properties, standard TIMEDB operations are provided, without any additional cost. On the other hand, proposal-vetting operations are implemented by mapping the GPVM operation to the corresponding TIMEDB operation(s).

Our prototype also supports DDL (Data Definition Language) commands (i.e., the commands which allow to create a relation, to remove a relation, and to alter a relation). The interpretation of such commands has been extended to link each relation to three additional relations. Such additional relations implement the set of proposals of insertion, of proposals of deletion and of Update-proposals. Observe that they are stored as TIMEDB temporal relations, but are managed by the additional layer.

The proposer and the evaluator operations are defined and managed in the additional layer too. The additional computational complexity of our implementation (with respect to TIMEDB) is quite limited.

In our prototypical implementation [25], we have implemented a simpler version of the data model described in the current paper, in which only transaction time is considered, and only manipulation operations are provided. The realization of algebraic operators in such a tool is one of the goals of our future work.

Moreover, it is worth stressing that, since TSQL2 is a "consistent extension" of SQL, our approach may be conceived as a further layer on top of a SQL database, too.

## 9 Related works

In this paper, we have discussed our extensions to the BCDM model, to cope with proposal vetting in a relational environment, in which both valid time and transaction time can be supported.

We are not aware of any other approach in the literature coping with such an issue. However, we think that it might be important to compare our approach with other approaches that share at least some of our goals. In particular, in the area of database versioning, many object-oriented approaches have been devised to face changes to a database, due to the proposals of different versions, evolving in time. In general, a main difference between object-oriented approaches and relational approaches has been pointed-out by Sciore [31, page 425]: "*The relational model has a limited modeling capacity, and so researchers in historical relations have all being forced to extend the relational model in some way. On the other hand, object-oriented models are able to encapsulate the notion of time in classes. Thus there is no need to*

*develop a new historical object-oriented model; what we need is a methodology for using these classes in our existing model*".

Specifically, in [31, 32], Sciore has proposed an approach coping with transaction and valid time, and alternative versions. Issues such as physical strategies to store versions, change notification and schema evolution are explicitly outside the scope of that approach. Transaction and valid time, and alternative versions are dealt with using the notion of *annotated variables* (roughly speaking, annotated variables in [31, 32] are variables whose intension can be addressed by time and\or version indexes), and introducing proper *methods* to access and manipulate them. Separate annotations and methods are defined to cope with the different aspects (valid time, transaction time and alternative versions). On the other hand, as in BCDM (and, in general, in all relational approaches to bitemporal data), in our approach we explicitly cope with valid and transaction time in an integrated way. Through the definition of Update-proposals, we give an explicit data model for data in which both bitemporal aspects and alternative version are considered.

The approach in [33] implements bitemporal databases using database versions expressed with Database Version model (DBV) [34]. Branching alternatives are expressed using alternative identifiers managed at the application level. The approach has been implemented on top of $O_2$ object-oriented DBMS. In [33] the main goal is that of providing minimal extensions to DBV in order to cope with transaction and valid time, but neither the underlying semantics of the interplay of time and alternative versions, nor the formal properties of the extension being built are taken into account.

More recently, another object-oriented data model has been proposed for dealing both with valid and transaction time and with versions, the Temporal Versions Model (TVM) [35, 36, 37]. However, although in [37] an operational semantics of (an extension of) TVQL is given, the treatment of the temporal aspects is not explicitly stated. Last, but not least, no property of being a consistent extension of any previous model is provided.

In the recent years most approaches supporting changes in databases focused on schema evolution and schema versioning [38, 39, 40]. Roddick in [41] surveyed the main issues involved with schema versioning and evolution. When changes to the schema are performed, two main problems have to be dealt with: maintaining the consistency of the schema, and handling the consistency of data with regard to the modified schema. Several approaches have been proposed regarding the various data models: for the relational model (see, e.g., [42]), for the object-oriented model (e.g., [43, 44]) and for conceptual models such as ERM (e.g., [45]). Such approaches seem to us only loosely related with our one, since we operate at the level of data (tuples), not at the level of schema. Specifically, our approach aims at managing the change to data values. On the other hand, in the schema versioning approaches, data change is usually not managed as a "primitive" notion, but as a (possibly automatically managed) process induced by changes to the schema.

## 10 Discussion and future work

In this paper we face the problem of storing in a relational database and evaluating (accepting/rejecting) proposals of updates to clinical knowledge. While medical data (and, specifically, clinical guidelines) constitute our current application context, the approach we propose is general. We can apply it to wider application context, including emerging phenomena such as the collaborative definition of encyclopedias (such as Wikipedia [] and Citizendium[]).
Specifically, we have proposed GPVM, a *semantic* framework supporting proposal vetting (i.e., proposal and evaluation of update) about data in a relational environment, in which transaction time and (possibly) valid time have to be managed.

In our approach we have defined a new data model, manipulation operations and algebra. We have based our approach on the "unifying" BCDM semantic model, extending it to support cooperative sessions of work. Specifically, the most relevant extensions to the BCDM model are:

(1) the treatment of mutually exclusive alternatives of relational tuples. We faced this phenomenon with the introduction of the basic notion of Update-proposal, which is the core of our approach. Notice that, while the notion of alternative versions of data has been already explored by some database data versioning approaches (based on the object-oriented paradigm – see, e.g., [34]) this notion is, to the best of our knowledge, new in the relational environment. In this context, in fact, relations are usually interpreted as sets (i.e., *conjunctions*) of tuples. The extension to BCDM to cope with *alternative* (and mutually exclusive) tuples has involved substantial changes to BCDM itself at the level of (i) data model, (ii) manipulation language, (iii) algebra;

(2) the treatment of two levels of data (the evaluator and the proposal levels) and of users (evaluators and proposers), each one with its manipulation operations;

(3) the treatment of sets of proposals as "macro" operations, to be executed as an atomic operation.

Our extensions have been devised in such a way that GPVM can be regarded as an *upper layer* built upon BCDM, i.e., we have proved that (i) GPVM *data model* and (ii) *algebra* are ***reducible*** to the BCDM one, and that (iii) GPVM *manipulation operations* are a ***consistent extension*** to BCDM ones.
By proving properties i-iii, we grant that our approach can be added as a support for update proposal and evaluation on top of any of the temporal relational database approaches grounded on the BCDM semantics. This fact enhances the ***generality*** of our work, as well as its ***implementability***.

As proof of its implementability, we have developed a prototypical implementation of our approach on the top of TIMEDB [26] (a prototype implementing –an extension of– TSQL2). Since our prototype implements only

manipulation and DDL operations, we plan to complete it with algebraic operations. Observe that the data model and algebraic level, at which our approach operates, is suitable for clearly defining the specifications of the work, and for providing the semantic basis of the implementation. However, they are not directly usable by proposers and evaluators. A higher-level interface, exploitable to execute SQL-like queries, is thus required, and is the goal of our future work.

## References


[1] P. Terenziani, G, Molino, M. Torchio A Modular Approach for Representing and Executing Clinical Guidelines. Artificial Intelligence in Medicine 23, 249-276, 2001.

[2] P. Terenziani, S. Montani, A. Bottrighi, G. Molino, M. Torchio, Applying Artificial Intelligence to Clinical Guidelines: the GLARE Approach, in Computer-based Medical Guidelines and Protocols: A Primer and Current Trends, Volume 139 Studies in Health Technology and Informatics, Edited by: A. Ten Teije, S. Miksch and P. Lucas, July 2008.

[3] A. Bottrighi, P. Terenziani, S. Montani, M. Torchio, G. Molino, Clinical Guidelines Contextualization in GLARE, Proc. AMIA'06, Washington, November 2006.

[4] http://www.wikipedia.org Wikipedia, the free encyclopedia (URL last accessed on 02/09/2012)

[5] http://www. www.wiktionary.org Wiktionary, the free dictionary (URL last accessed on 02/09/2012)

[6] http://www.citizendium.org/, Citizendium, a citizens' compendium of everything (URL last accessed on 12/07/2011).

[7] http://www.mysql.org (URL last accessed on 02/09/2012)

[8] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", in Communications of the ACM, 1970.

[9] R.T. Snodgrass, Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann Publishers, Inc., San Francisco, July, 1999.

[10] Y. Wu, S. Jajodia, X. Sean Wang: Temporal Database Bibliography Update. Temporal Databases, Dagstuhl: 338-366, 1997.

[11] L. Liu, M. Tamer Özsu (Eds.), Encyclopedia of Database Systems. Springer US, 2009.

[12] L. Edwin McKenzie, Richard T. Snodgrass: Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. ACM Comput. Surv. 23(4): 501-543, 1991.

[13] A. Tansel, J. Cliffor, S. Gadia, S. Jajodia, A. Segev and R.T. Snodgrass (eds), Temporal databases: theory, design and implementation, Benjamin/Cummings, 1994.

[14] G. Özsoyoglu, R. T. Snodgrass: Temporal and Real-Time Databases: A Survey. IEEE Trans. Knowl. Data Eng. 7(4): 513-53, 1995.

[15] C.S. Jensen, R.T. Snodgrass: Temporal Data Management. IEEE Trans. Knowl. Data Eng. (TKDE) 11(1):36-44, 1999.

[16] C.S. Jensen, R.T. Snodgrass, Semantics of Time-Varying Information, Information Systems, 21(4), 311–352, 1996.

[17] R. T. Snodgrass (Ed.), The TSQL2 Temporal Query Language. Kluwer 1995.

[18] J. Dunn, S. Davey, A. Descour, and R.T. Snodgrass, Sequenced Subset Operators: Definition and Implementation, proc. ICDE'02, 2002.

[19] British Thoracic Society guidelines for the management of suspected acute pulmonary embolism, Thorax 2003, 58, 470-483, 2003.

[20] W.B.G. Macdonald, A.P. Patrikeos, R.I. Thompson, B.D. Adler, and A.A. van der Schaaf, Diagnosis of pulmonary embolism: Ventilation perfusion scintigraphy versus helical computed tomography pulmonary angiography, Australasian Radiology 49, 32-38, 2005.

[21] V. Khatri, S. Ram and R.T. Snodgrass, Augmenting a Conceptual Model with Geospatiotemporal Annotations, IEEE Transactions on Knowledge and Data Engineering 16(11), 1324-1338, November 2004.

[22] K.R. Dittrich, R.A. Lorie, Version Support for Engineering Database Systems. IEEE Trans. Software Eng. 14(4): 429-437, 1988.

[23] Oracle Database 10g Workspace Manager Overview. An Oracle White Paper http://www.oracle.com/technology/products/database/workspace_manager/pdf/twp_AppDev_Workspace_Manager_10gR2.pdf (URL last accessed on 05/05/2012).

[24] R. Elmasri, S. Navathe. Fundamentals of Database Systems, 6/Ed. Addison-Wesley, 2011

[25] A. Vigo. Estensioni alle basi di dati temporali per il supporto alla proposta ed alla valutazione di modifiche di dati, laura degree thesis in computer science, Università del Piemonte Orientale, 2009.

[26] http://www.timeconsult.com/Software/Software.html (URL last accessed on 13/09/2012).

[27] www.java.com (URL last accessed on 13/09/2012)

[28] http://www.ibm.com/developerworks/data/library/techarticle/dm-0408anderson/ (URL last accessed on 14/09/2012)

[29] http://www.php.net/ (URL last accessed on 14/09/2012)

[30] http://www.w3.org/html/ (URL last accessed on 18/09/2012)

[31] E. Sciore, Using Annotations to Support Multiple Kinds of Versioning in an Object-Oriented Database System. ACM Trans. Database Syst. 16(3), 417-438, 1991.

[32] E. Sciore, Versioning and Configuration Management in an Object-Oriented Data Model VLDB J. 3(1), 77-106, 1994.

[33] S. Gançarski, Database Versions to Represent Bitemporal Databases. In Proceedings of the 10th international Conference on Database and Expert Systems Applications (August 30 - September 03, 1999). T. J. Bench-Capon, G. Soda, and A. M. Tjoa, Eds. LNCS 1677. Springer-Verlag, London, 832-841, 1999.

[34] W. Cellary and G. Jomier, Consistency of Versions in Object Oriented Databases. In Proc. 16th VLDB, pages 432-441, 1990.

[35] M. M. Moro, S.M. Saggiorato, N. Edelweiss and C.S. Santos, Adding Time to an Object-Oriented Versions Model. In Proceedings of the 12th international Conference on Database and Expert Systems Applications. H. C. Mayr, J. Lazanský, G. Quirchmayr, and P. Vogel, Eds. LNCS 2113. Springer-Verlag, London, 805-814., 2001.

[36] M.M. Moro, N. Edelweiss, A.P. Zaupa and C.S. Santos, TVQL - Temporal Versioned Query Language. In Proceedings of the 13th international Conference on Database and Expert Systems. A. Hameurlain, R. Cicchetti, and R. Traunmüller, Eds. LNCS 2453. Springer-Verlag, London, 618-627, 2002.

[37] R. Machado, Á. F. Moreira, R. de Matos Galante and M. M. Moro, Type-safe Versioned Object Query Language; Journal Of Universal Computer Science JUcs, volume 12, issue 7, september 2006, Pages: 938-957, ISSN: 0948-695X.

[38] Second International Workshop on Evolution and Change in Data Management, in M. Genero, F. Grandi, W.J. van den Heuvel, J. Krogstie, K. Lyytinen, H.C. Mayr, J. Nelson, A. Olivé, M. Piattini, G. Poels, J.F. Roddick, K. Siau, M. Yoshikawa, E.S.K. Yu (Eds.): Advanced Conceptual Modeling Techniques, ER 2002 Workshops: ECDM, MobIMod, IWCMQ, and eCOMO, Tampere, Finland, October 7-11, 2002, Revised Papers. L 2784 Springer 2003.

[39] Third International Workshop on Evolution and Change in Data Management, in S. Wang, D. Yang, K. Tanaka, F. Grandi, S. Zhou, E.E. Mangina, T. Wang Ling, I.Y. Song, J. Guan, H.C. Mayr (Eds.): Conceptual Modeling for Advanced Application Domains, ER 2004 Workshops CoMoGIS, COMWIM, ECDM, CoMoA, DGOV, and ECOMO, Shanghai, China, November 8-12, 2004, Proceedings. LNCS 3289 Springer 2004.

[40] 4th International Workshop on Evolution and Change in Data Management, in J.F. Roddick, V.R. Benjamins, S. Si-Said Cherfi, R.H.L. Chiang, C. Claramunt, R. Elmasri, F. Grandi, H. Han, M. Hepp, M.D. Lytras, V.B. Misic, G. Poels, I.Y. Song, J. Trujillo, C. Vangenot (Eds.): Advances in Conceptual Modeling - Theory and Practice, ER 2006 Workshops BP-UML, CoMoGIS, COSS, ECDM, OIS, QoIS, SemWAT, Tucson, AZ, USA, LNCS 4231 Springer 2006.

[41] J.F. Roddick, A survey of schema versioning issues for database systems, Information and Software Technology, 37(7),383-393, 1995.

[42] C. De Castro, F. Grandi and M.R. Scalas, Schema versioning for multi-temporal relational databases. Information Systems 22(5), 249-290, 1997.

[43] F. Grandi, F. Mandreoli and M.R. Scalas, A Generalized modeling framework for schema versioning support. Proceedings of the Australian Database Conference (ADC 2000),Canberra, Australia, Maria Orlowska (Ed.), 33-40, 2000

[44] F. Grandi, F. Mandreoli, A formal model for temporal schema versioning in object-oriented databases. Data Knowl. Eng. 46(2), 123-167, 2003.

[45] C.T. Liu, S.K. Chang and P.K. Chrysanthis, An entity-relationship approach to schema evolution. Proceedings of the International Conference on Computing and Information. Abou-Rabia, 0., Chang, C. K., and Koczkodaj, W. W. (Ed.), 575-578, 1993