Modeling Temporal Databases and Temporal Constraints

Mohamed Mkaouar, Mohamed Moalla
LIP2 Laboratory
University of Tunis El Manar, Faculty of Science
Campus Universitaire 2092 - El Manar Tunis, Tunisia
Mkaouar.Mohamed@gmail.com, Mohamed.Moalla@fst.rnu.tn

Rafik Bouaziz
MIRACL Laboratory
University of Sfax, Faculty of Economics and Management
Route de l'Aéroport 3018, Sfax, Tunisia
Raf.Bouaziz@fsegs.rnu.tn

Abstract— **Applications requiring a full and efficient management of data feel the need to consider, beside the current facts, historical and future facts, and to keep the track of the manipulation of facts by the DBMS. In this paper, we define concepts and modeling tools to express constraints, taking into account the temporal dimension. The expression of these constraints is achieved through an independent platform modeling, in the UML-TF profile. Next, we propose extensions to the SQL3 to be able to convert enhanced UML-TF representations in a specific object-relational platform.**

Keywords-Modeling Temporal Databases; Temporal Constraints; UML profile; SQL3 extension

I. INTRODUCTION

Over the past twenty five years there have been many studies concerning the integration of different temporal specifications in databases (DB), and of new languages and temporal features in the DBMS [11][14][15][25]. Nevertheless, there is not yet a general implementation of Temporal DBMS and DB by manufacturers and designers. We attribute this fact to two main reasons: the importance of legacy DB and the complexity of the temporal environment.

Indeed, the importance of legacy DB, and of applications that exploit them, makes any translation from a modeling and/or development environment to another difficult and expensive. That explains the slow transition from navigational (network) DB to relational DB, and why, despite the dominance of object-oriented programming, OODB have yet to find dominance. It is the same for the temporal dimension; there is now a gentle introduction to some temporal features in the current DBMS [16][20][21], which are still limited.

Investigations concerning works dealing with temporal environment are not yet sufficient to develop temporal DBMS. Other investigations remain indispensable to master temporal data management and schema evolution [2], concurrency control in temporal DB [17], development of temporal applications [18], etc.

Such development requires appropriate formalisms and tools, as well as a solid training for developers to this new environment. It also requires rigorous use of the principles of abstraction, provided by systemic methods and adopted by object-oriented approaches, especially with the advent of the MDA (Model Driven Architecture) introducing three levels of abstraction: CIM (Computational Independent Model), PIM (Platform Independent Model) and PSM (Platform Specific Model).

We must therefore define a CIM and a PIM before proceeding within a development considering a specific platform in order to simplify the adoption of temporal DB. But this principle of abstraction has not been adopted by most of the works on temporal DB.

In this paper, we start by enriching the UML-TF profile [19] to be able to model different classes of constraints related to time, in the CIM and PIM levels. We then study how to transform an UML-TF representation into a specific object-relational representation, including temporal constraints.

The remainder of this paper is organized into six sections as follows. Section two provides a brief overview of the state of the art. The third Section describes UML-TF and reviews on how to model specifications incorporating the temporal dimension in this profile, with an illustration based on a real application. In the fourth Section, we present different classes of constraints on these specifications and how to express them using examples from the same application. In the fifth Section we propose a transformation of a UML-TF representation into a specific object-relational representation.

II. STATE OF THE ART AND RELATED WORK

Time can be taken into account in accordance with what is happening in the real world and/or in DB, whose updates can be done with some phase shifts. So, two standards time types, *valid time* and *transaction time* [13], and several kinds of temporal facts which are mainly, *Valid-time facts*, *Transaction-time facts* and *bitemporal facts*, are defined.

Valid-time facts may relate to the past, to the present or to the future. Each such fact is represented by a timestamp with their valid-times in reality. Thus, it becomes possible to maintain the history of all valid facts, which can be updated in real time with retroactive effect or postactive effect, but not to keep track of deletions and corrections of errors.

Transaction-time facts can keep track of the manipulation of facts by the DBMS, which timestamps them by the execution time of the transaction that manipulates each fact. This track covers insert operations, update operations — whether evolution updates or error correction— and delete operations. Transaction-time facts timestamps are defined according to the schedule adopted by the operating system and a granularity generally equal to the second, but could be thinner if necessary. Thus, it becomes possible to maintain the history of all the facts, valid or erroneous, past or current, but not future. But only the current facts may be updated; updates can not be made here either with retroactive effect, or with postactive effect.

These valid or transaction histories have then insufficiencies. A complete history can be assured only if facts are timestamped by both valid and transaction-times, thus we obtain bitemporal facts. It then becomes possible to update the facts with retroactive or postactive effects, keep track of valid facts and erroneous ones, and distinguish between valid facts and erroneous ones. However, the management of these facts becomes increasingly complicated [2]. Then, the use of temporal dimensions must be justified by the needs of users. A temporal DB must contain conventional facts (non-temporal), when we do not need historical, valid-time facts, when we need a valid history of facts in reality, transaction-time facts, when we need a history that keeps track of the manipulation of facts by the DBMS, and bitemporal facts, when we want to have a complete history. In addition, the DBMS must include effective techniques to handle these kinds of facts [2][17][23], on the one hand, and design methods must fully adopt the principle of abstraction and include means for expressing temporal specifications [19], on the other hand.

We want to expand in this paper the enhancement of the UML-TF to be able to model temporal constraints in the CIM and PIM levels. We then propose to classify the constraints into several classes and sub-classes (cf. Section 4) and suggests ways of expression for each of these classes and subclasses. To our knowledge, this aspect has not been sufficiently addressed, either by the temporal DB models, or by the works dealing with constraint classification and checking [3][7][9].

To the works concerning PSM, we note here that the expression of temporal constraints under SQL [6][24]. Snodgrass [24] were limited to examining primary keys and foreign keys, without detailing the other constraints that we may declare at the CIM and PIM. Authors of reference [6] proposed to represent all the constraints, even simple constraints of not null fields or of columns domain’s, by means of assertions without using standard SQL statements, which do not promote their adoption. We want to enrich SQL3 to allow the expression of temporal constraints, affecting different levels of abstraction, in a declarative and the simplest possible manner.

III. Modeling temporal DB with UML-TF

We first review the main notation allowing the modeling of temporal specifications with UML-TF [19] and then illustrate such modeling with a case study.

A. *Temporal notation*

With UML-TF, modeling is made through the various levels of abstraction proposed by MDA, while expressing the temporal dimensions of any facts by stereotypes that we classify into three categories: valid-time stereotypes, transaction-time stereotypes and bitemporal stereotypes. To these three categories we associate the following icons:

- ‘’ which symbolizes the real world clock, and is intended to the first category of stereotypes. This icon is used at the CIM level of the MDA approach.
- ‘’ which represents the clock of the machine, and is intended to the second category of stereotypes. This icon is used at the PIM level of MDA.
- ‘’ which symbolizes both clocks, and is intended to bitemporal stereotypes. Such icon results from any two declarations, first by a valid-time icon, then by a transaction-time icon.

To realize the UML-TF profile, we defined an abstract stereotype for each of the three categories of stereotypes. Each abstract stereotype is a realization of the Evolutionary Stereotype [8] that allows to aggregate new semantic definitions. An abstract stereotype is then characterized by appropriate meta-properties [19]. Some of these meta-properties (Calendar, Type, Granule) concern timestamps and are initialized by default values. These values, which can be modified depending on the context, are to be exploited by the temporal features that the DBMS should ensure to allow the attribution of timestamps. Other meta-properties allow expressing constraints on timestamps or on temporal instances. These constraints, which we will detail in Section 4, are to be included in the schema of the DB and must be supported by the DBMS.

Each of the valid-time stereotypes, transaction-time stereotypes and bitemporal stereotypes represents both a realization of the concerned abstract stereotype, with appropriate values of its meta-properties, and an extension of the concerned meta-class of the UML metamodel. Due to space limitations, we limit ourselves here to mention these stereotypes and explain their effect. Further details concerning their definition can be found in [19].

1) Valid-time stereotypes: Any need of a history according to the valid-time dimension is expressed in the CIM level, by one of the following valid-time stereotypes:

- <<VTA>> (Valid-Time Attribute): associated with an attribute, this stereotype models the need to keep all the valid values that have taken or will take effect in reality. Each value is stamped with its valid-time.
- <<VTAs>> (Valid-Time Association): associated with an association, with or without associative-class, this stereotype models the need to preserve all valid links that have taken or will take effect in reality. If such a need is limited to one direction of the association, we use the stereotype <<VTR>> (Valid-Time Role).
- <<VTC>> (Valid-Time Class): This stereotype can associate valid timestamps to each object of the class.
- <<VTGS>> (Valid-Time Generalization-Specialization): This stereotype can associate valid timestamps to each object of the concerned sub-classes.

2) Transaction-time stereotypes: Such stereotypes can be used, at the PIM level, for any element to model the need to keep track of their instances by the DBMS. We define here, as in the previous case, a stereotype for each type of element: <<TTA>> (Transaction-Time Attribute), <<TTAs>> (Transaction-Time Association), <<TTR>> (Transaction-Time Role), <<TTC>> (Transaction-Time Class) and <<TTGS>> (Transaction-Time Generalization-Specialization).

3) *Bitemporal stereotypes:* Any element for which is associated first a valid-time stereotype and second a transaction-time stereotype is considred bitemporal; the two sterotypes are systematically replaced by a bitemporal sterotype which contains the meta-properties of the two others and represented by the bitemporal icon. We define the following bitemporal stereotypes: <<BTA>> (BiTemporal Attribute) <<BTAs>> (BiTemporal Association) <<BTR>> (BiTemporal Role) <<BTC>> (BiTemporal Class) and <<BTGS>> (BiTemporal Generalization-Specialization).

B. *Case study*

As an example, let us consider the UML-TF class diagram of Figure 1. At this level, we briefly describe the application in general, and we further detail, in the following sections, how to declare constraints related to time. This diagram, concerns a higher education institution system. A person can be a student, a teacher or both. A teacher may be responsible at most than nine modules. Between seven and fifteen modules are taught for any given audience. Any training session concerns a teacher, a module, a group of an audience or an audience. It is described by a day, an hour and a classroom number. Each student should have, for each module that he studies, two or three marks.

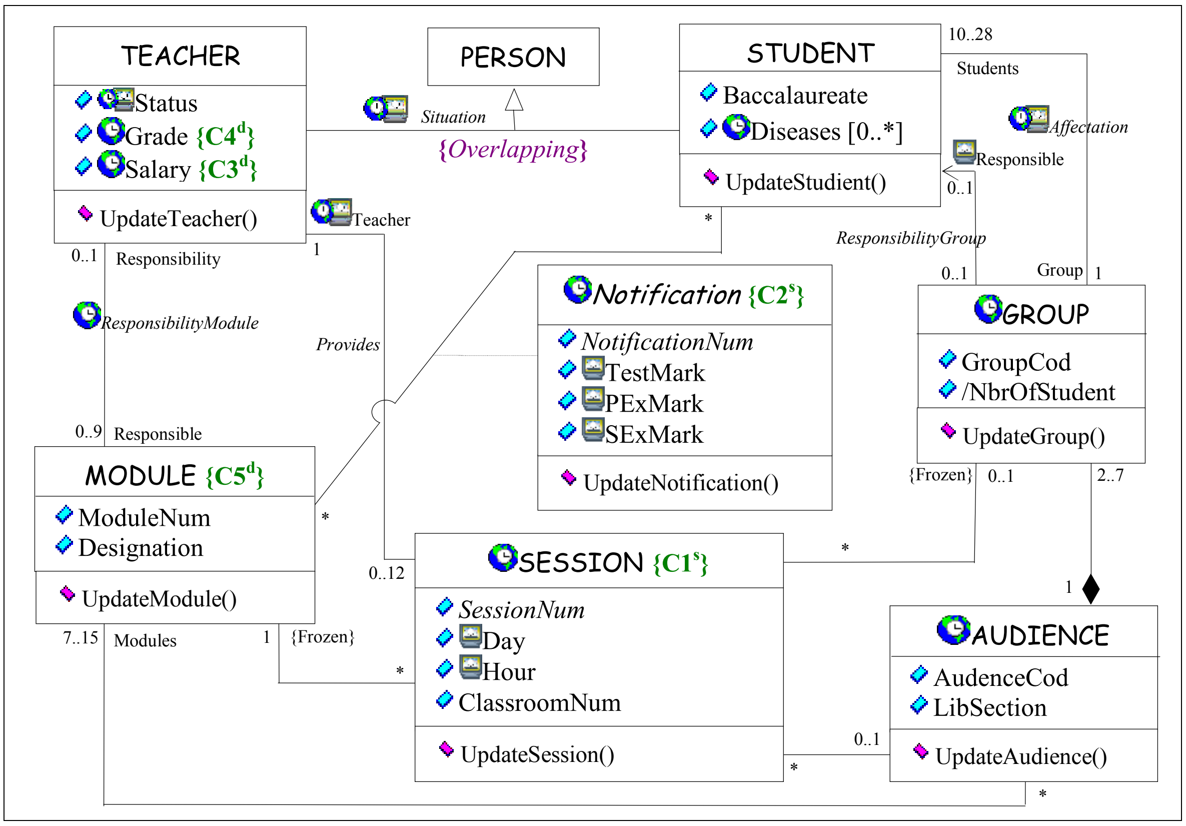

Figure 1. UML-TF class diagram modeling the DB of an application for a higher education institution.

In this diagram:

- 'Grade', 'Salary' and 'Diseases' are bitemporal attributes stereotyped by <<VTA>>;
- The association "ResponsabilityModule" and the associative-classes "Notification" are stereotyped by <<VTAs>>;
- AUDIENCE, GROUP and SESSION are three valid-time classes stereotyped by <<VTC>>;
- 'Responsible' is a transaction-time role stereotyped by <<TTR>>;
- 'TestMark', 'PExMark', 'SExMark', 'Day' and 'Hour' are transaction-time attributes stereotyped by <<TTA>>;
- 'Status' is a bitemporal attribute stereotyped by <<BTA>>;
- "Affectation" is a bitemporal association;
- 'Teacher' is a bitemporal role stereotyped by <<BTR>>;
- TEACHER and STUDENT are bitemporal classes stereotyped by <<BTGS>>.

IV. New means for expressing temporal constraints

Commonly, the constraints are classified into static and dynamic [10]. A static constraint controlling values or links that can take an attribute or a relationship, respectively, while a dynamic constraint control the evolution of these values or these links (the employee's salary can not decrease, for example).

For static constraints, we distinguish in UML simple constraints, called predefined, which usually focus on one element of the diagram and for which the notation defines "means" to express them in the diagram in a light manner. There are also complex static constraints, which are usually focused on more than one element of the diagram, and that we call here not-predefined constraints; using OCL [22], it is possible to formulate these constraints in different ways, depending on the context. To avoid overloading the diagrams, we propose to represent non-predefined constraints, and also dynamic constraints, not by their explicit expressions, but by codes assigned to them. These codes are to be placed between two brackets beside the name of the chosen context, as the five examples of the diagram in Figure 1 (**{C1^s}**, **{C2^s}**, **{C3^d}**, **{C4^d}** and **{C5^d}**; it is possible to define many others temporal constraints in this diagram). To distinguish a static complex constraint from a dynamic constraint in the diagram, we use the exhibitors '^{s}' and '^{d}' next to the code of the constraint. The explicit expression in OCL of these constraints is then attached to the considered diagram.

To allow the expression of constraints on temporal instances, especially dynamic constraints, in OCL, it is required to extend this constraint language. Indeed, a temporal instance is characterized by elements from the following: Value, Start Valid-Time, End Valid-Time, Start Transaction-Time, End Transaction-Time and Index. The extension that we propose allows identifying these elements, by using the following keywords: **Value**, **SVT**, **EVT**, **STT**, **ETT** and **Index**. These keywords are to be used for any temporal instance using dot notation; the default is **Value**.

Following our analysis of the impact of the temporal dimension on the expression of constraints, we have proposed three categories of constraints related to time. We study these categories in the three following sub-sections.

A. *Static constraints involving a temporal dimension*

When a static constraint is declared on an element which is associated with a valid-time or a transaction-time stereotype, it *often* changes meaning; the DBMS is not sufficient to that verification at the current time, but extends this verification at any time of the maintained history to also cover past and, possibly, future facts.

The new meanings of these constraints do not affect the means of their expression, but rather affect their verification. However, we propose to express them by means of temporal invariants, denoted by '**Temporal inv:**'; we therefore propose to enrich our extension of OCL with this keyword.

Consider first the new meaning of an example of a predefined constraint of the diagram in Figure 1: "The multiplicity of roles of the association "Affectation" means that at a given instant, a group of education is associated with at least 10 students and at most 28 students, and a student is assigned to one and a single group. However, a group may be associated with more than 28 students and a student may be assigned to multiple groups if we consider a history that stretches over a long period".

The constraint {*overlapping*}, which is not in fact, does not change meaning. Applied to the Generalization-Specialization *Situation*, it continues to mean that a person can be both a student and a teacher, either for the current time or at any past or future time.

We now detail the two non-predefined constraints **{C1^s}** and **{C2^s}**.

- **{C1^s}** : A teacher can not teach the same group more than twice at the same period.

  Context SESSION **Temporal inv: C1**

  Session.**allinstances->forAll**(s1, s2, s3 | s1 <> s2 **and** s2 <> s3 **and** s1.Audience = s2.Audience = s3.Audience **implies** s1.Teacher <> s2.Teacher **or** s2.Teacher <> s3.Teacher **and** s1.Group <> s2.Group **or** s2.Group <> s3.Group)

- **{C2^s}** : A mark is to attribute to a student for a given module only if this student is already enrolled in an audience for which the module is taught.

  Context Notification **Temporal inv: C2**

  (**self.**Module) **includes** **self.**Student.Group.Audience.Modules

When these constraints are applied to bitemporal elements, only valid instances are taken into account. Incorrect or deleted (in a non-destructive manner) instances are not affected since they have already been verified.

B. *Dynamic constraints*

A formal expression of these constraints requires the use of a constraint language incorporating temporal operators, as already studied by several other works, especially those who have proposed temporal extensions to OCL [5][26]. This expression also requires the enrichment of OCL by the proposed keywords to be able to access to the various elements of a temporal instance. Nevertheless, for some constraints in this category, mainly those that focus on a single element, it is possible to find 'semi-graphic' means to their expression; this expression is easier to use in modeling.

Constraints **{C3^d}**, **{C4^d}** and **{C5^d}** are examples of dynamic constraints. In what follows, we detail their meaning and their expression in OCL extended by the above mentioned keywords and by operators of temporal logic. For the two constraints **{C3^d}** and **{C4^d},** we also propose 'semi-graphic' means to use as patterns for these types.

- **{C3^d}** : The salary of a teacher can not decrease.

  Context TEACHER **inv : C3**

  self. Salary**->forAll**(s1 : Salary, s2 : Salary | s1.**SVT** < s2.**SVT** **implies** s1.**value** < s2.**value**)

  /*or s1 ***precedes*** s2 **implies** s1.**value** < s2.**value**) */

  The 'semi-graphic' pattern that we propose to this type of constraint is the following: **{↗}**

- **{C4^d}** : The qualification of a teacher in his career follows the following order: 'Assistant Professor' (A), 'Associate Professor' (AP) and 'Professor' (Pr); but a teacher can begin as 'AP'.

  Context TEACHER **inv : C4**
  self.Grade**->forAll**(g1 : grade, g2 : grade |
  g1.**SVT** < g2.**SVT implies** g1.**value** = 'A' **and** g2 = 'AP' **or** g1 = 'AP' **and** g2 = 'Pr')
  and self.Grad**->forAll** (g : grad | g.**index** =1
  implies g = 'A' **or** g = 'AP')

  The 'semi-graphic' pattern that we propose to this type of constraint is the following:
  {Order : 'A', 'AP', 'Pr'}
  ▲ ▲
- **{C5^d}** : The responsibility of a module is assigned to a teacher only if this teacher involved in teaching this module since 5 years.

  Context MODULE **inv : C5**
  self.Responsability **implies** *since 5 years*
  self.Responsability.Session**->notEmpty()**

C. *Constraints on timestamps and on temporal instances*

Compared to the two first categories of constraints, the constraints in this category may cover one or two temporal dimensions. These constraints concern stereotyped elements, as was announced in Section 2, and for which we have defined meta-properties in the abstract stereotypes. This allows expressing them in a simple manner, simply by valorizing the concerned meta-properties. We define two classes for this category: constraints on timestamps, and constraints on timestamped values of temporal instances.

1) Constraints on timestamps: In this class of constraints we distinguish three sub-classes as follows:

The constraints of the first sub-class concern the definition of default values or restrictions for a valid or transaction stamp. These constraints may relate to the timestamps of all instances of a stereotyped element, or for some instances of this element. Some restrictions depend on the properties of the timestamp, for example, it is not possible to restrict the duration of a timestamp of type instant.

As examples of such constraints, we can define for the considered application the following ones: "A teaching period starts on 15/9 of each year and ends on 31/7 of that year."; "A teacher can not remain in the grade of 'Associate Professor' less then 3 years.".

The second sub-class concerns all constraints defined between a valid timestamp and a transaction timestamp of a bitemporal element. These constraints have been widely studied in [12]. As examples, we define the following constraints: "The transaction-time of a status can not exceed its valid-time more than one month.".

The constraints of the third sub-class are systematically imposed by the association of valid-time stereotypes in the diagram. These constraints depend on the type of the timestamp. In particular, it ensures data integrity when the timestamp type is 'temporal interval' or 'temporal element'. Indeed, in this case:

- The timestamp of a link should be included (or equal) in (to) the timestamp of each concerned object; this is the case for example of links defined between STUDENT and GROUP.
- The timestamp of an object of an aggregate class must be included (or equal) in (to) the timestamp of the concerned object in the component class; this is the case of objects of the GROUP class vis-à-vis to objects of the AUDIENCE class.

2) Constraints on temporal instances: We retain at this level the following four main types:

- The extent of the maintenance of past valid values, in number of values or in duration relative to the current time; this concerns the *vacuuming* parameters [23] which is a way to destructively delete data becoming obsolete. The default values assigned to the two concerned meta-properties (V_Nbr and V_Duration) are equal to infinity. In the example, we can impose that: "The number of groups for each audience is not to historize beyond three periods."; "The label of a section is to historize only for the last three values.".
- The number of corrections or changes. By default, this number is unlimited. In the example, we can impose that it is not possible to correct the assignment of a training session to a teacher more than twice for the same period.
- How to keep the instances of a temporal collection vis-à-vis the *coalescing* operator [4]. By default, these instances are to keep coalesced, that is two instances that follow in time must have different values. In the example, links that concern affectations of students to groups, and notifications of students are to keep not coalesced.
- The management strategy for future data; this type of constraints is specific to bitemporal elements; by default, this management is to perform in a destructive manner.

V. SPECIFIC REPRESENTATION TO THE OBJECT-RELATIONAL PLATFORM

We present in this section how to extend the SQL3 standard by new keywords to allow the representation of timestamped tables and columns, as well as constraints related to time.

Our objective is to raise the issue and propose some solutions. More adequate solutions, especially as regards the expression of complex static constraints and dynamic ones, require substantial investment and resources that are beyond us.

A. *Expression of timestamps*

About the declaration of timestamps in SQL3 orders, we propose to put the letter 'V', the letter 'T' or the two letters, after the name of the concerned table or column. Each letter, representing a timestamp, is followed by the values of the meta-properties defined to the timestamp, when the defaults values have to be changed.

B. *Static constraints involving temporal dimension*

As in UML-TF, we propose that the expression of static constraints is to done in the same way, whether with or without a time dimension, apart from the fact that the syntax is enriched by the word '**Temporal**', to indicate that the constraint must be checked at any instant and not only at the current time. Indeed, the constraints checking module of the DBMS must ensure the checks within temporal DB.

To simplify the expression of different non-predefined constraints, *i.e.*, complex static ones, we propose not to use assertions, instead we use **CHECK** constraints. Also, we propose extending the use of such constraints by the possibility of the employment of many free variables (**Self**) that can or not be connected to the same table. For example, we propose to declare the constraint **{C1^s}** as follows:

```
ALTER TABLE SESSION
ADD Temporal CONSTRAINT  Ck_C1
CHECK  Self1-TEACHER.TeacherNum = Self2-
TEACHER.TeacherNum AND Self1-AUDIENCE.AudienceCod
= Self2-AUDIENCE.AudienceCod AND Self1-
GROUP.GroupCod = Self2-GROUP.GroupCod
AND NOT EXISTS
    (Self1-TEACHER.TeacherNum = Self3-
    TEACHER.TeacherNum AND Self1-
    AUDIENCE.AudienceCod = Self3-
    AUDIENCE.AudienceCod AND Self1-GROUP.GroupCod =
    Self3-GROUP.GroupCod));
```

In this constraint, we used three free variables 'Self1', 'Self2' and 'Self3' for each of the three tables: TEACHER, AUDIENCE and GROUP.

C. *Dynamic constraints*

For the constraints of this class for which we found 'semi-graphic' expressions, it is also possible to find patterns to represent them in a simple manner. For examples:

- for the example of the constraint **{C3^d}**, we propose the pattern: **NOT DECREASE**.
- for the constraint **{C4^d}**, we propose the following pattern: **IN THIS ORDER** {'Val$_1$', 'Val$_2$', …} [**START WITH** 'Val$_1$' [**OR** 'Val$_2$']*];

For the other dynamic constraints, which are currently represented by relational DBMS with triggers, we also propose to express them by **CHECK** constraints. The expression of these constraints requires the use of temporal logic operators. For example, we propose to declare the constraint **{C5^d}** as follows:

```
ALTER TABLE MODULE
ADD CONSTRAINT  Ck_5
CHECK  Self.NumTeacherResponsible
AND EXISTS
    (Self-TEACHER.NumTeacher =
    Self.NumTeacherResponsible
    AND Self-TEACHER Since 5 Year Self);
```

D. *Constraints on timestamps and on temporal instances*

To express the constraints on timestamps, we use the arithmetic operators of comparison as well as Allen operators [1], *i.e.*, **BEFORE**, **AFTER**, **OVERLAPS**, **EQUAL**, **MEETS**, **DURING**, **CONTAINS**, **STARTS**, **FINISH**, etc.

In addition, to be able to express a restriction on the duration of a temporal interval, we propose the following pattern: {**MAXPERIOD** | **MINPERIOD**} Value ***SCALE*** [**ON** 'Val 1', 'Val 2', ...]. ***SCALE*** can be **YEAR**, **MONTH**, **DAY** etc.; it specifies the used temporal granule. Without the option [**ON** 'Val 1', 'Val 2', ...], the constraint is applied to all instances of the concerned element, with this option, the constraint is applied only to instances defined by the indicated values; for example, **MAXPERIOD** 7 **YEAR ON** 'Assistant Professor' is applied only to Assistant Professors.

For the four types of constraints on temporal instances proposed in Section 4.C.2, we propose to represent them using the following patterns:

- **VACCUMING AFTER** Value {***SCALE*** | **VALUES**} [**CASCADE**]. The **CASCADE** option can progress the vacuuming for data related to the concerned data.
- **ONLY** Value {**CORRECTIONS** | **EVOLUTIONS**} **ALLOWED**.
- [**COALESED** | **NOT COALESED** [**WITH** Value **MAX VALUEQUIVALENT**]}. By default, the instances are to keep coalesced. The option **WITH** Value **MAX VALUEQUIVALENT** precise the maximum number of non-coalesced equivalent values.
- [**FUTURE UPDATES DESTRUCTIVE** | **FUTURE UPDATES NOT DESTRUCTIVE**].

E. *Recapitulative example*

We briefly illustrate here how we plan to use our proposals to enrich SQL3, through an excerpt of the command concerning the creation of the TEACHER table:

```
CREATE TABLE TEACHER       V (DEFAULT, TEMPORAL
    ELEMENT, DEFAULT)  T
(TeacherNum   NUMBER(3)
CONSTRAINT  Pk_Teacher         PRIMARY KEY,
LName          VARCHAR2(20)    NOT NULL,
Status V  (DEFAULT, DEFAULT, YEAR)  T  VARCHAR2(2),
…
CONSTRAINT  Dur_TeacherStatus     Status.VT  DURING
    OR  EQUAL  TEACHER.VT,
CONSTRAINT  SVT_STT_Status        Status.STT  <=
    Status.SVT  +  1  MONTH,
CONSTRAINT  Vacc_TeacherStatus    Status VACCUMING
    AFTER  3  VALUES,
CONSTRAINT  Ito_TeacherGrade      Grade IN  THIS  ORDER
    {'A', 'AP', 'Pr'} START WITH 'A' Or 'AP',
CONSTRAINT  MinEs_TeacherGrade  Grade MINPERIOD  3
    YEARS ON 'AP',
CONSTRAINT  O2E_TeacherGrade      Grade ONLY  2
    EVOLUTIONS ALLOWED,
CONSTRAINT  Nd_TeacherSal          Salary NOT
    DECREASE,
CONSTRAINT  Fund_TeacherSalary    Salary FUTURE
    UPDATES NOT DESTRUCTIVE,
CONSTRAINT  Vacc_TEACHER          TEACHER
    VACUUMING  AFTER  80  YEAR  CASCADE );
```

VI. Conclusion and Future Work

We presented in this paper new concepts to improve the modeling of four different kinds of facts: conventional facts (non-temporal), valid-time facts, transaction-time facts, and bitemporal facts.

Our proposals concern, first, the expression of constraints in the UML-TF profile [19], dedicated to the modeling of temporal DB. This profile allows such modeling in a simple, customizable and progressive manner, using expressive decorations. With the enrichment provided by these proposals, it is possible to take into account temporal constraints, classified into three categories: static constraints invoking a temporal dimension, dynamic constraints, and constraints on the timestamps or on temporal instances. This classification into three categories, refined if necessary on classes and sub-classes, helped us to identify appropriate ways for expressing each type of constraint. In particular, the formal expression of these constraints required the extension of OCL with new keywords.

Secondly, we have proposed enhancements to SQL3 to enable the mapping of UML-TF class diagrams in an object-relational model, attempting to take advantage of the "declarative" approach of constraints adopted by DB languages. Thus, the designer can focus his attention on the expression without worrying about the modules to be integrated into the DBMS providing their checking.

Our future work focuses on the development of new tools and the enrichment of platforms that support them in order to implement our proposals. We also plan to study how checking temporal constraints according to our proposals.

References

[1] Allen J. F., Maintaining Knowledge about temporal intervals, *Communication of the ACM*, 1983, pp. 832-843.

[2] Bouaziz R. and Brahmia Z., Gestion des données temporelles dans un environnement multi-versions de schémas, *Technique et Science Informatiques*, vol. 28 n° 1, 2009, pp. 39-74.

[3] Böhlen M. H., Valid Time Integrity Constraints, Technical Report (94-30), University of Arizona, 1994.

[4] Böhlen M. H., Snodgrass R. T., and Soo M. D., "Coalescing in Temporal Databases.", *Proceedings of the 22nd International Conference on Very Large DataBases (VLDB)*, Bombay, India, September 1996, pp. 180-191.

[5] Cengarle M. V. and Knappe A., Towards OCL/RT, *Lecture Notes in Computer Science*, vol. 2391, 2002, pp. 390–409.

[6] Cordeiro R. L. F., Edelweiss N., Galante R. M., and dos Santos C. S., "TVCL: Temporal Versioned Constraint Langage.", *20 Simpósio Brasileiro de Bancos de Dados, Anais/Proceedings*, 2005, pp. 55–69.

[7] Cordeiro R. L. F., Galante R. M., Edelweiss N., and dos Santos C. S., "A Deep Classification of Temporal Versioned Integrity Constraints for Designing Database Application.", *Proceedings of the 19th International Conference on Software Engineering & Knowledge Engineering*, 2007, pp. 416-421.

[8] Debnath N., Riesco D., Montejano G., Grumelli A., Maccio A., and Martellotto P., "Definition of new kind of UML Stereotype based on OMG Metamodel", *Proceedings of the Arab International Conference on Computer Systems and Applications: AICCSA'03*, Tunis, 14-18 July 2003, Tunisia.

[9] Doucet A., Fauvet M. C., Gançarski S., Jomier G., and Monties S., "Using Database Version to Implement Temporal Integrity Constraints.", *Proceedings of the Second International Workshop on Constraint Databases*, 1997.

[10] de Brock E. O., "A general treatment of dynamic integrity constraints.", *Data & Knowledge Engineering*, 32, 2000, pp. 223-246.

[11] Etzion O., Jajodia S., and Sripada S. M. (Editors), *Temporal Databases: Research and Practice*, Spring-Verlag, *Lecture Notes in Computer Science*, vol. 1399, 1998.

[12] Jensen C. S. and Snodgrass R. T, "Temporal Specialization and Generalization.", *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, n° 6, 1994, pp. 954-974.

[13] Jensen C. S. and Dyreson C. E. (Editors), Böhlen M. H., Clifford J., Elmasri R., Gadia S. K., et al., "The Consensus Glossary of Temporal Database Concepts.", in Etzion et al., 1998.

[14] Jensen C. S., "Temporal Database Management.", dr.techn. thesis by Christian S. Jensen, defended April 2000 available at: http://www.cs.auc.dk/~csj/Thesis.

[15] Jensen C. S. and Snodgrass R. T. (Editors), Temporal Database Entries for the Springer Encyclopaedia of Database Systems, Technical Report TR-35, TimeCenter, May, 2008.

[16] Lomet D., Barga R., Mokbel M. F., and Shegalov G., "Transaction Time Support Inside a Database Engine.", *Proceedings of the International Conference on Data Engineering*, 2006, pp. 35-46.

[17] Makni A and Bouaziz R., Concurrency Control for Temporal Databases, *International Journal of Databases Management Systems*, vol. 2 n° 1, 2010, pp. 39-58.

[18] Mkaouar M. and Bouaziz R., L'édification de framework pour l'aide au développement d'applications temporelles, *Information Science for Decision Making*, n° 21, 2005.

[19] Mkaouar M. and Bouaziz R., UML-TF: un profil UML pour la représentation des faits temporels, *Technique et Science Informatiques*, vol. 26 n° 3-4, March-April 2007, pp. 305-338.

[20] Oracle Corporation, *Advanced Application Developer's Guide: Using Oracle Flashback Technology*, Oracle Documentation, 2008.

[21] Oracle Corporation, *Workspace Manager Developer's Guide*, Oracle Documentation, 2009.

[22] OMG, *Object Constraint Language (OCL) Specification*, Object Management Group, www.omg.org, 2001-2010.

[23] Roddick J. F., Schema Vacuuming in Temporal Databases, *IEEE Transactions on Knowledge and data engineering*, vol. 21 n° 5, 2009, pp. 744-747.

[24] Snodgrass R. T., Böhlen M. H., Jensen C. S., and Steiner A., "Adding Valid Time to SQL/Temporal.", *SQL/Temporal Change Proposal, ANSI X3H2-96-501r2, ISO / IEC / JTC1 / SC21 / WG3 DBL-MAD-146r2*, "Adding Transaction Time to SQL/Temporal.", *SQL/Temporal Change Proposal, ANSI X3H2-96-502r2, ISO / IEC / JTC1 / SC21 / WG3 DBL-MCI-143*, 1996.

[25] Wu Y., Jajodia S., and Wang X. S., "Temporal Database Bibliography Update.", in Etzion et al., 1998.

[26] Ziemann P. and Gogolla M., OCL Extended with Temporal Logic, *Lecture Notes in Computer Science*, vol. 2890, 2003, pp. 617-633.