L'EDIFICATION DE FRAMEWORK POUR L'AIDE AU DEVELOPPEMENT D'APPLICATIONS TEMPORELLES

Mohamed Mkaouar,
Doctorant en Informatique
Mohamed.Mkaouar@fsegs.rnu.tn

Rafik Bouaziz,
Maître-Assistant en Informatique
Raf.Bouaziz@fsegs.rnu.tn

Adresse professionnelle
Faculté des Sciences Économiques et de Gestion de Sfax ★ BP 1088 ★ 3088 Sfax, Tunisie

Résumé : Dans cet article, nous proposons une approche pour l'extension d'un SGBD Relationnel permettant une manipulation aisée des faits temporels, c'est-à-dire des faits associés à des valeurs de temps. Cette approche consiste en la définition de framework, constitué de fonctionnalités temporelles, qui dispense le développeur de l'écriture de requêtes complexes. Il devient ainsi possible de construire des formulaires permettant aux utilisateurs de mettre à jour et d'interroger des faits temporels d'une manière simple et confortable.

Summary: In this paper, we propose an approach for the extension of a Relational DBMS to easy manipulate temporal facts, i.e. facts associated to time values. Our approach consists in the definition of a framework, composed of temporal functionalities, which dispenses the developer for writing complex queries. So, it becomes possible to construct forms that allow users updating and querying temporal facts in a simple and comfortable manner.

Mots clés : Développement d'Applications Temporelles, Bases de Données Temporelles, Manipulation des Faits Temporels, Requêtes Dynamiques.

Key words: Development of Temporal Applications, Temporal Databases, Manipulating Temporal Facts, Dynamic Queries.

L'édification de framework pour l'aide au développement d'applications temporelles

1 - INTRODUCTION

Plusieurs applications, telles que la gestion des projets et la gestion des carrières des employés, nécessitent de garder trace, dans leurs Bases de Données (BD), de l'évolution des faits dans le temps. Les problématiques relatives à la conception et la manipulation des faits temporels ont fait l'objet de plusieurs travaux de recherche dans le domaine des BD Temporelles (BDT), depuis plus de vingt ans déjà [Jensen et Snodgrass (1997), Skjellang (1997-a, 1997-b), Wu, Jajodia et Wang (1998), Tsotras et Wang (1999), Grandi (2003)].

Beaucoup de ces travaux ont intéressé l'extension du modèle relationnel et de son langage non procédural standard SQL. Quelques dizaines d'extensions de SQL ont été déjà proposées dans Chomicki (1994). Par ailleurs, Gao et Snodgrass (2003), entre autres, proposent une extension temporelle pour XML.

SQL/Temporal [Eisenberg et Melton (2000)], résultat des travaux de TSQL2 [Snodgrass (1995)], est actuellement le standard pour la définition et la manipulation des BDT conformément à la technologie relationnelle. Cependant, ce standard n'est pas encore adopté par les constructeurs actuels des SGBD, d'une part, et il est critiqué par les auteurs d'autres extensions, en l'occurrence SQL/TP [Toman (2000)], notamment en ce qui concerne la complexité de la syntaxe et la pauvreté de la sémantique, d'autre part.

De ce fait, les développeurs continuent à faire recours à des solutions ad hoc, basées sur les seuls outils proposés par les SGBD actuellement commercialisés, pour le développement des applications manipulant des faits temporels. D'ailleurs, R.T. SNODGRASS a montré la possibilité d'utiliser SQL pour l'implémentation et la manipulation des faits temporels [Snodgrass (2000)]. Cependant, l'auteur signale aussi la difficulté d'exprimer de telles requêtes et la nécessité d'avoir une connaissance approfondie de la stratégie de manipulation des faits temporels. Nous avons recensé de telles difficultés à travers le développement de l'application "Centrale des Renseignements" de la Banque Centrale de Tunisie (BCT) [Mkaouar (1999)]. Ces difficultés constituent un frein à l'expansion des BDT.

Face à ces difficultés, comment peut-on promouvoir la mise en œuvre des BDT ?

Construire de toute pièce un "SGBD Temporel", comportant de naissance les éléments nécessaires à la prise en compte des faits temporels (stockage, indexation, contrôle de concurrence, etc.), est certainement l'alternative la plus performante, mais requiert des ressources très importantes et exige une acceptation du standard. Étendre un SGBD existant, pour disposer d'un "support temporel", est beaucoup plus simple à mettre en œuvre. En effet, cette deuxième alternative tire profit des services offerts par le SGBD utilisé.

C'est ainsi que la plupart des supports temporels proposés, en l'occurrence ceux qui constituent des extensions de SGBDR comme les propositions de [Torp, Jensen et Snodgrass (1997), Yang, Ying, et Widom (2000), Slivinskas, Jensen, et Snodgrass (2001), Matus-Castillejos et Jentzsch (2005), Noh et Gadia (2005)], ont été confectionnés dans le cadre de cette alternative. Ces extensions permettent l'implémentation et la manipulation des BDT à travers l'écriture de requêtes temporelles. Deux principales approches ont été suivies :

La première, la plus utilisée, consiste à implémenter une extension de SQL, en tant qu'une couche logicielle au-dessus d'un SGBD existant. C'est cette couche qui assure la traduction des requêtes temporelles, exprimées selon la syntaxe du nouveau langage, en requêtes SQL à exécuter par le SGBD utilisé. TimeDB [Steiner (2005)] est une extension qui adopte cette approche et qui peut être intégrée à l'un des trois SGBDR, Oracle, Sybase et Cloudscape's JBMS. Elle utilise une version de TSQL2 intitulée ATSQL2 (*Applied TSQL2*).

L'inconvénient majeur d'une telle approche réside dans le fait qu'il est difficile, voir impossible, de traduire des requêtes temporelles complexes [Torp, Jensen et Snodgrass (1997)].

La deuxième approche ne prévoit pas la définition d'un nouveau langage de requêtes. Elle consiste plutôt à procéder à l'exploitation des supports d'extensibilité des SGBD, tels que *Informix Datablades*, *DB2 Extenders* ou *Oracle Cartridges*, pour enrichir ces SGBD par de nouveaux types et routines permettant la prise en compte des faits temporels. Le langage de requêtes du SGBD utilisé peut ainsi permettre l'implémentation et la manipulation des BDT, tout en gardant sa syntaxe intacte.

TIP (T*emporal* I*nformation* P*rocessor*) [Yang, Ying et Widom (2000)] est, à notre connaissance, le seul support temporel adoptant cette approche. Il enrichit le SGBDR Informix par des types temporels (*Chronon*, *Span*, *Instant*, *Period* et *Element*) et par différentes routines de comparaison, de conversion et d'agrégation de valeurs temporelles.

L'inconvénient majeur de cette approche réside dans le fait qu'il est difficile, voir impossible, d'exprimer des requêtes temporelles complexes. D'ailleurs, les perspectives des travaux de TIP ont été annoncées pour élargir son pouvoir expressif en ce qui concerne de telles requêtes.

L'écriture des requêtes temporelles pour l'implémentation et la manipulation des BDT, que ce soit directement sous SQL ou à travers ces deux approches, pose donc des problèmes d'expressivité, même pour les développeurs expérimentés. Par ailleurs, de telles requêtes sont souvent peu performantes du fait que le SGBD utilisé n'est pas équipé d'algorithmes d'optimisation appropriés ; il est alors nécessaire de développer de tels algorithmes [Slivinskas, Jensen et Snodgrass (2001), Bowman et Toman (2001), Yang et Widom (2003)].

Dans cet article, nous proposons une approche pour l'édification de framework temporel, en tant qu'ensemble de composants logiciels aidant au développement d'applications temporelles.

Cette approche consiste en la génération dynamique de requêtes SQL requises pour les opérations de mise à jour et de consultation des faits temporels. Il s'agit de construire des fonctionnalités temporelles (bibliothèques de fonctions et canevas-types) à exploiter au sein d'un environnement de développement rapide d'applications RAD (*Rapid Application Development*). Ces fonctionnalités dispensent les développeurs de l'écriture de requêtes temporelles. Elles leur offrent la possibilité de construire des formulaires permettant à l'utilisateur de mettre à jour et d'interroger les faits temporels d'une manière simple et confortable.

Le type de framework que nous proposons doit donc prendre en charge la déclaration des opérations de mise à jour (insertion, modification et suppression non destructives) des faits temporels et d'interrogation des faits courants et des faits historiques ; ces derniers peuvent être valides ou erronés. Un tel framework gagne aussi à fournir des canevas-types pour simplifier et homogénéiser la manipulation des estampilles et la navigation dans l'historique des faits.

La suite de l'article est organisée comme suit. La deuxième section présente le modèle temporel que nous avons utilisé. La troisième section propose de définir formellement les opérations de mise à jour et de consultation des faits temporels dans le cadre de ce modèle. Les composantes du type de framework proposé, ainsi que sa spécificité par rapport aux autres supports de la littérature, font l'objet de la quatrième section. Quant à la cinquième, elle présente un prototype que nous avons développé pour la validation de notre approche. Elle montre aussi le résultat de l'expérimentation des fonctionnalités de ce prototype à travers le développement d'une application concrète. La conclusion et les perspectives de ce travail constituent l'objet de la dernière section.

2 - MODELE TEMPOREL RETENU

Les extensions temporelles proposées pour le paradigme relationnel spécifient les faits temporels en procédant à l'estampillage de granules (n-uplets ou attributs) par des **temps de validité** et/ou des **temps de transaction**. Les temps de validité permettent de savoir quand un granule est considéré valide dans le monde réel ; ces temps sont fournis par les utilisateurs. Les temps de transaction

permettent de savoir quand un granule est courant dans la BD ; ces temps sont déterminés et gérés par le système. Aussi bien les temps de validité que les temps de transaction peuvent être définis sur le domaine des *instants*, le domaine des *intervalles temporels*, le domaine des *séquences d'intervalles temporels* ou tout autre domaine temporel.

Le modèle proposé par Bouaziz, Moalla, et Rolland, (1992) procède à l'estampillage des n-uplets (ou tuples selon le vocable anglais que nous allons utiliser dans la suite de l'article) par des intervalles temporels. Par ailleurs, et pour des raisons d'accélération des opérations d'accès aux données courantes, évidemment les plus utilisées, le modèle retient la proposition de ventiler les données de chaque relation temporelle sur différents espaces de stockage relatifs aux données dépassées, aux données courantes et, éventuellement, aux données futures [Nobecourt, Rolland, et Lingat, (1988), Sethuraman (2003)]. C'est un modèle temporel instantané, basé sur des points temporels, selon la classification présentée dans [Böhlen, Busatto et Jensen (1998)]. Il utilise les intervalles temporels pour représenter les ensembles de points temporels consécutifs. Chaque intervalle est défini par deux points qui précisent sa borne inférieure et sa borne supérieure.

Les auteurs du modèle proposent de désigner par **clé générique**, l'ensemble des attributs formant la clé de la relation, sans aucun attribut temporel, et par **tuple générique**, l'ensemble des tuples ayant une même valeur de clé générique.

Ce modèle, conforme aux conventions standards [Jensen et Dyreson (1998)], spécifie donc les faits temporels à travers des tuples, agrégés dans des tuples génériques, et définit les trois natures de relations temporelles standard comme suit :

- Une Relation de nature Temporelle de Validité (RTV) procède à l'estampillage de chaque tuple par un intervalle temporel de validité, mémorisé par deux attributs (temps de début de validité "TDV" et temps de fin de validité "TFV"). Une RTV est implémentée physiquement à travers trois espaces de stockage, réservés successivement pour les tuples dépassés (considérés tous comme valides), les tuples courants et les tuples futurs.
- Une Relation de nature Temporelle de Transaction (RTT) procède à l'estampillage de chaque tuple par un intervalle temporel de transaction, mémorisé par deux attributs (temps de début de transaction "TDT" et temps de fin de transaction "TFT"). Une RTT est implémentée physiquement à travers deux espaces de stockage, réservés successivement pour les tuples dépassés (valides et erronés) et les tuples courants. Une telle relation ne permet pas au système de distinguer les tuples valides dépassés des tuples erronés.
- Une Relation de nature BiTemporelle (RBT) procède à l'estampillage de chaque tuple par un intervalle temporel de validité et un intervalle temporel de transaction. Une RBT est implémentée, comme une RTV, à travers trois espaces de stockage, réservés successivement pour les tuples dépassés (valides et erronés), les tuples courants et les tuples futurs. Contrairement à une RTT, une RBT permet au système de distinguer les tuples valides dépassés des tuples erronés.

Les tuples valides d'un même tuple générique d'une RTV ou d'une RBT doivent avoir des intervalles de validité qui ne se chevauchent en aucun point.

Ce modèle a permis d'assurer aussi bien la gestion temporelle des données que le versionnement des schémas des relations [Bouaziz et Moalla (1996)]. Les tuples valides d'un même tuple générique peuvent être définis sous différentes versions, pouvant être de différentes natures. C'est ainsi que les structures des schémas des versions des relations sont enrichies par des attributs systèmes permettant d'instaurer un *lien horizontal* permanant entre tout tuple valide et le tuple valide qui le précède et celui qui le suit, conformément à l'ordre des intervalles temporels de validité ou, à défaut, de transaction. Il s'agit des attributs :

- N_absolu : numéro absolu du tuple ; les numéros absolus sont tenus pour chaque tuple générique à part et sont alloués et gérés par le SGBD. Pour chaque tuple générique, "N_absolu" prend les valeurs 1, 2, 3, etc., conformément à l'ordre de création des tuples.

- Na_pred : numéro absolu du tuple prédécesseur.
- Nvs_pred : numéro de la version du schéma sous laquelle est défini le tuple prédécesseur.
- Na_succ : numéro absolu du tuple successeur.
- Nvs_succ : numéro de la version du schéma sous laquelle est défini le tuple successeur.

La structure du schéma d'une version de nature BT est enrichie, de son côté, par deux autres attributs systèmes permettant d'instaurer un *lien vertical* permanant entre chaque *tuple de correction* et le tuple qu'il corrige (ou le premier tuple erroné lorsqu'il en corrige plusieurs), chaque *tuple de réduction* et le tuple qu'il réduit (cf. § 3.2) et chaque *tuple de suppression* et le tuple qu'il supprime (cf. § 3.4). Il s'agit des attributs :

- Na_erroné : numéro absolu du tuple erroné ;
- Nvs_erroné : numéro de la version du schéma sous laquelle est défini le tuple erroné.

En complément, ce modèle définit une relation système, constituée d'attributs systèmes, pour chaque relation R d'une BDT. Il s'agit d'une **relation générique**, nommée VG_R et contenant autant d'occurrences que de tuples génériques T^G de R. Elle permet :

- l'accès direct au premier tuple (à l'aide des attributs "Na_pt", "Nvs_pt" et "Esp_pt") et au dernier tuple (à l'aide des attributs "Na_dt", "Nvs_dt" et "Esp_dt") de T^G ; "Esp_pt" et "Esp_dt" indiquent les espaces de stockage dans lesquels sont stockés le premier et le dernier tuples valides de T^G ;
- de mémoriser, dans l'attribut "Nvs_tc", le numéro de la version du schéma sous laquelle est défini le tuple courant relatif à T^G.
- de mémoriser, dans l'attribut "Na_nt", le numéro absolu à attribuer au prochain nouveau tuple de T^G ;

Exemple :

Considérons le fait bitemporel "salaire des employés" implémenté à travers une RBT "S_EMP", ayant une seule version. Supposons que cette relation comporte, à la date du 10/10/2005, un tuple générique relatif à l'employé 'Edam', identifié par le numéro de matricule (NM) '55' et comportant trois tuples valides, dont deux sont dépassés, mémorisés dans l'espace réservé aux tuples dépassés de "S_EMP" et valorisés comme l'indique le tableau 1. Le troisième est un tuple courant, mémorisé dans l'espace réservé aux tuples courants de "S_EMP", valorisé comme l'indique le tableau 2.

	N_absolu	NM	Salaire	TDV	TFV	TDT	TFT	Na_pred	Nvs_pred	Na_succ	Nvs_succ	Na_erroné	Nvs_erroné
T^1	1	55	900	10/01/2003	31/12/2003	05/02/2003: $15^H\ 05'\ 19''$	31/12/2003: $23^H\ 59'\ 59''$	Nil	Nil	3	1	Nil	Nil
T^3	3	55	1 000	01/01/2004	28/02/2005	05/12/2003: $10^H\ 35'\ 15''$	28/2/2005: $23^H\ 59'\ 59''$	1	1	2	1	Nil	Nil

Tableau 1 : **Valeurs des attributs des tuples dépassés du tuple générique 55**

	N_absolu	NM	Salaire	TDV	TFV	TDT	Na_pred	Nvs_pred	Na_succ	Nvs_succ	Na_erroné	Nvs_erroné
T^2	2	55	1 200	01/03/2005	31/12/2006	05/12/2003: $10^H\ 35'\ 02''$	3	1	Nil	Nil	Nil	Nil

Tableau 2 : **Valeurs des attributs du tuple courant du tuple générique 55**

VG_S_EMP mémorise le tuple générique 55 : Na_pt = 1, Nvs_pt = 1, Esp_pt = 'P', Na_dt = 2, Nvs_dt = 1, Esp_dt = 'C', Nvs_tc = 1 et Na_nt = 4.

Le tuple T^1 a été introduit le 05/02/2003 avec effet rétroactif puisque sa validité devrait commencer le 10/01/2003. Le SGBD a cessé de le considérer comme tuple courant à la fin de sa période de validité.

Les tuples T^2 et T^3 ont été insérés, avec effet postactif, le 05/12/2003, respectivement à $10^H 35' 02''$ et $10^H 35' 15''$. Leurs numéros absolus (2 et 3) ont été accordés par le SGBD conformément à l'ordre de leurs arrivées, indépendamment du fait que l'intervalle de validité de T^3 précède celui de T^2. Le tuple T^3 a cessé d'être courant à la fin de son intervalle de validité. Quant à T^2, qui est encore un tuple courant, il n'a pas de temps de fin de transaction. C'est pour cela que l'espace réservé aux tuples courants ne comporte pas l'attribut "TFT". Il s'agit de même pour l'espace réservé aux tuples futurs.

3 - MISE A JOUR ET INTERROGATION DES FAITS TEMPORELS

Le modèle que nous avons utilisé définit une stratégie fine pour la mise à jour non destructive des tuples d'une BDT, pouvant appartenir à des relations de nature photo, temporelle de validité, temporelle de transaction ou bitemporelle [Bouaziz, Moalla et Rolland (1992)]. Toute opération de mise à jour non destructive est assurée à travers une combinaison d'opérations élémentaires de base (insertion, modification ou suppression de tuples et de tuples génériques), ainsi que de transferts de tuples.

Dans cet article, nous nous limitons à la définition formelle des opérations de consultation, de modification, d'insertion et de suppression de tuples bitemporels.

Cette définition formelle se situe dans le cadre d'un système de bases de données [Bernstein, Hadzilacos et Goodman (1987)], constitué d'un Gestionnaire de Transactions (GT), d'un Gestionnaire de Données (GD) et d'un Contrôleur de Concurrence (CC). Les problèmes de contrôle de concurrence, traités dans [Elloumi, Bouaziz et Moalla (1998), Bouaziz et Makni (2005)], ne sont pas considérés dans cet article.

Nous nous intéressons ici aux expressions à constituer par les applications, en faisant recours aux différentes composantes du framework. Le GT se charge de l'exécution de ces expressions et de la transmission des opérations de lecture, d'écriture et de suppression au GD. Ce dernier refuse toute opération lorsque les répercussions de cette opération ne sont pas conformes aux exigences de la gestion des faits temporels ou lorsque les valeurs des données introduites ne vérifient pas les contraintes d'intégrité spécifiées au niveau du schéma de la BDT.

La quatrième section traite des outils destinés aussi bien à l'expression des requêtes par l'utilisateur qu'à la réalisation des opérations de mise à jour sur les données. L'objectif des définitions formelles de la présente section consiste à cerner les fonctionnalités à assurer par le GD.

3.1 - Consultation des tuples bitemporels

L'interrogation d'une BDT, pouvant contenir des faits de différents "types" (photo, temporel de validité, temporel de transaction ou bitemporel), est une opération complexe. Nous l'avons traitée, rappelons-le, selon une approche qui se distingue des autres approches de la littérature par le fait qu'elle dispense le développeur de l'écriture des requêtes d'interrogation, généralement complexes.

Pour la consultation de tuples bitemporels, appartenant à une ou plusieurs RBT, une application doit offrir au GT le 5-uplet suivant :

<{R} ; {A} ; [C] ; [G] ; [T]>, où :

- {R} est la liste des relations.
- {A} est la liste des attributs concernés.
- C est une condition qui définit le critère de recherche.
- G présente la (les) colonne(s) de groupement.
- T présente la (les) colonne(s) de tri.

Par ailleurs, l'application peut utiliser les composants appropriés du framework pour assister l'utilisateur dans la spécification des valeurs et des prédicats nécessaires à la construction de C. Elle fait appel à d'autres composants pour la construction dynamique d'une ou de plusieurs requête(s) de sélection, conforme(s) à la syntaxe de SQL. Les tuples à sélectionner doivent vérifier la condition C qui spécifie, conformément à la syntaxe d'une clause WHERE de SQL, une ou plusieurs sous-conditions (SC).

Pour pouvoir consulter les tuples valides (considérés comme courants dans la BDT) à un point particulier dans le temps, C doit spécifier une ou plusieurs clauses "temporelles" (CT). Une CT peut concerner les temps de validité et/ou les temps de transaction. Elle peut spécifier :

- soit une valeur d'un point temporel qui doit appartenir aux intervalles temporels des tuples à sélectionner, les précéder ou les succéder,
- soit une valeur d'un intervalle temporel qui doit chevaucher entièrement les intervalles temporels des tuples à sélectionner.

C est donc définie comme suit :

C = < SC, [AND/OR, SC]*, [AND, CT]* >.

Le résultat de la consultation peut être groupé, selon le(s) attribut(s) de G, et/ou trié, selon le(s) attribut(s) de T. Ce résultat est défini comme suit :

Res_Interrogation : < Tuple(s) sélectionné(s) > / < Message "Aucun tuple ne satisfait la condition" > / < Message "Interrogation refusée", Raison >.

Exemple :

Supposons que l'on est le 25/10/2005 et nous souhaitons consulter le salaire de l'employé n° 55 valide au 20/09/2005, ainsi que sa période de validité. Selon notre approche, l'utilisateur n'a à spécifier que le numéro 55 et la valeur du point temporel désiré. L'application constitue alors le 5-uplet de consultation : <S_EMP ; Salaire, TDV, TFV ; NM = 55 AND appartient (20/09/2005, [TDV - TFV]) ; ; >. Le GT transmet ce 5-uplet au GD. Ce dernier génère la requête nécessaire à la recherche des données désirées. Il transmet le résultat au GT qui se charge enfin de l'affichage des valeurs mémorisées dans le tuple T^2 de l'exemple précédent.

3.2 - Modification d'un tuple bitemporel

Avant toute opération de modification d'un tuple bitemporel T, l'application concernée doit procéder à sa consultation. Le GD renvoie à cette application, à travers le GT, les valeurs des attributs de base de T et la valeur de son N_absolu. Après avoir effectué les modifications désirées par l'utilisateur, l'application offre au GT le 7-uplet suivant :

<R ; [I] ; {VAC} ; N_absolu ; {NAM} ; {VAM} ; TT>, où :

- R est le nom de la relation concernée.
- I est le numéro de la version de R à utiliser. C'est un paramètre optionnel qui n'est exigé que dans le cas de l'insertion d'un tuple dépassé sous une version dépassée.
- {VAC} définit la liste des valeurs des attributs clés. Les noms de ces attributs sont connus d'une manière implicite pour la relation considérée.
- {NAM} et {VAM} définissent respectivement les listes des noms et des valeurs des attributs modifiés.
- TT est la valeur du temps de transaction, utilisée pour renseigner les attributs de transaction concernés (TDT du tuple de mise à jour et, éventuellement, TFT de T et TFT du tuple de mise à jour).

Le GT transmet ce 7-uplet au GD pour procéder à la modification non destructive de T. Selon notre modèle, cette modification est effectuée par la définition d'un *tuple de mise à jour* (T^{maj}) au sein du tuple générique T^G de T. Le résultat d'une telle modification diffère selon que l'intervalle de validité de T est modifié ou non. Limitons-nous ici au cas où l'intervalle de validité de T est modifié. Il peut s'agir dans ce cas soit d'une mise à jour d'évolution de valeurs, soit d'une mise à jour de correction d'erreurs. Si le TFV de T est infini et le TDV de T^{maj} suit celui de T, la modification est considérée comme une mise à jour d'évolution de valeurs ; autrement, elle est considérée comme une correction d'erreurs.

Une mise à jour d'évolution de valeurs nécessite la modification des attributs systèmes de T pour être lié à son tuple successeur T^{maj}, le transfert de T pour devenir un tuple dépassé (mais restant valide), l'insertion de T^{maj} et la modification de T^G. Le résultat est donc défini comme suit :

R_Modif_Evol : < T modifié et transféré, T^{maj} inséré, T^G modifié > / < Message "Modification refusée", Raison >.

Une correction d'erreurs nécessite tout d'abord la recherche de tous les tuples valides de T^G, autres que T, ayant des intervalles de validité chevauchés par celui de T^{maj} :

- Pour tout chevauchement entier, le tuple couvert est considéré comme entièrement

erroné (T^{ee}), il est traité d'une manière similaire à T.

- Pour tout chevauchement partiel, le tuple concerné est considéré comme partiellement erroné (T^{pe}) ; il est erroné durant la période couverte par T^{maj} et valide durant la période non couverte. Cette validité de données est mémorisée à travers l'insertion d'un *tuple de réduction* (T^{r}) ayant un intervalle de validité égal à la période non couverte par T^{maj} et gardant les mêmes valeurs des attributs de base de T^{pe}.

Cette correction nécessite également l'aménagement des liens entre tous les tuples touchés : tuples valides ou tuples erronés, tuples introduits par l'utilisateur ou tuples de réduction et enfin tuples prédécesseurs ou tuples successeurs. De plus amples détails sont présentés dans [Bouaziz, Moalla et Rolland (1992)].

T^{maj} est inséré dans l'un des espaces dépassé, courant ou futur, selon la position du temps courant par rapport à son intervalle de validité.

Le résultat de correction dans le cadre de ce deuxième cas est donc défini comme suit :

Res_Modif_Corr : < T modifié [et transféré], [T^{ee} modifié [et transféré]]*, [T^{pe} modifié [et transféré], T^{r} inséré]$^{0..2}$, T^{maj} inséré, T^{G} modifié, Liens entre tuples touchés aménagés > / < Message "Modification refusée", Raison >.

Notons que le symbole "*" et la suite "$^{0..2}$" désignent le fait que la production de l'élément de résultat peut être répétitive, respectivement plusieurs fois ou au plus deux fois.

Exemple :

Supposons que l'on a décidé, le 25/10/2005, de corriger le tuple courant T^2 de l'employé n° 55 pour lui affecter un salaire de 1 250 au lieu de 1 200, et à partir du 01/01/2005, au lieu du 01/03/2005. Après la consultation de ce tuple, l'application concernée envoie au GD le 7-uplet de modification : <S_EMP ; ; 55 ; 2 ; Salaire, TDV ; 1250, 01/01/2005 ; 25/10/2005:12:15:06>. Cette modification a nécessité l'introduction d'un tuple de correction (T^4) et d'un tuple de réduction (T^5), réduisant l'intervalle de validité de T^3, d'une part, et la modification des attributs systèmes de T^1, T^2 et T^3, d'autre part. Après modification, "S_EMP" comporte, pour le tuple générique 55, 5 tuples, tel que c'est montré par le tableau 3.

	N_absolu	NM	Salaire	TDV	TFV	TDT	TFT	Na_pred	Nvs_pred	Na_succ	Nvs_succ	Na_erroné	Nvs_erroné
Espace réservé aux tuples dépassés													
T^1	1	55	900	10/01/2003	31/12/2003	05/02/2003: 15H 05' 19''	31/12/2003: 23H 59' 59''	Nil	Nil	**5**	**1**	Nil	Nil
T^3	3	55	1 000	01/01/2004	28/02/2005	05/12/2003: 10H 35' 15''	28/2/2005: 23H 59' 59''	**4**	**1**	2	1	Nil	Nil
T^2	2	55	1 200	01/03/2005	31/12/2006	05/12/2003: 10H 35' 02''	25/10/2005: 12H 15' 06''	3	1	**4**	**1**	Nil	Nil
T^5	5	55	1 000	01/01/2004	31/12/2004	25/10/2005: 12H 15' 06''	25/10/2005: 12H 15' 06''	**1**	**1**	**4**	**1**	**3**	**1**
Espace réservé aux tuples courants													
T^4	4	55	1 250	01/01/2005	31/12/2006	25/10/2005: 12H 15' 06''		**5**	**1**	Nil	Nil	**3**	**1**

Tableau 3 : **Valeurs des attributs des 5 tuples du tuple générique 55**

VG_S_EMP subit les modifications suivantes pour le tuple générique 55 : Na_dt = 4, Na_nt = 6.

Notons que :

- la chaîne des tuples valides est constituée maintenant par T^1, T^5 et T^4, ordonnés conformément à leurs intervalles de validité ;
- T^5 est lié verticalement au tuple partiellement erroné T^3.
- T^4 est lié verticalement aux tuples erronées T^3 et T^2, dans l'ordre de leurs intervalles de validité.

3.3 - Insertion d'un tuple bitemporel

Pour insérer un tuple bitemporel T, une application se limite à fournir au GT le 7-uplet suivant :

<R ; [I] ; {VAC} ; {VAV} ; {NAB} ; {VAB} ; TT>, où :

- R, I et {VAC} désignent les mêmes éléments que ceux présentés pour la modification.
- {VAV} définit la liste des valeurs des attributs de validité. Les noms de ces attributs sont connus d'une manière implicite.
- {NAB} et {VAB} définissent respectivement les listes des noms et des valeurs des attributs de base renseignés.
- TT est la valeur du temps de transaction, à affecter au TDT, et éventuellement au TFT, de T.

Le GT transmet tout 7-uplet d'insertion d'un tuple T au GD. Brièvement, le GD doit commencer par tester l'existence ou non d'un tuple générique T^G ayant une même valeur de clé générique que celle de T. Si T^G n'existe pas, il s'agit d'une insertion du premier tuple d'un nouveau tuple générique. Autrement, il s'agit d'une insertion d'un nouveau tuple mentionnant l'évolution d'un fait dans le temps. Le GD n'accepte cette insertion que dans le cas où il n'existe pas déjà un tuple de T^G ayant un intervalle de validité qui chevauche celui de T.

Le résultat de l'insertion est ainsi défini :

Res_Inser : < T inséré, T^G inséré / T^G modifié, [T^{pred} modifié], [T^{succ} modifié] > / < Message "Insertion refusée", Raison >.

3.4 – Suppression d'un tuple bitemporel

D'une manière similaire au cas de la modification, une opération de suppression d'un tuple bitemporel T doit être précédée par une opération de consultation de T. Ainsi, pour supprimer T, une application se limite à offrir au GD, à travers le GT, le 4-uplet suivant :

<R ; {VAC} ; N_absolu ; TT>.

Notons que TT est utilisée ici pour valoriser le TDT et le TFT du *tuple de suppression* et le TFT de T. En effet, la suppression non destructive d'un tuple bitemporel T, considéré jusque là comme valide, est assurée, selon notre modèle, par l'insertion d'un *tuple de suppression* (T^{sup}) au sein de l'espace des tuples dépassés, ainsi que par la modification des attributs systèmes de T pour qu'il soit considéré comme tuple supprimé. T est à transférer vers l'espace des tuples dépassés, s'il n'y est pas déjà.

Le résultat de la suppression est ainsi défini :

Res_Supp : < T modifié [et transféré], T^{sup} inséré, T^G modifié, [T^{pred} modifié], [T^{succ} modifié] > / < Message "Suppression refusée", Raison >.

4 - COMPOSANTES DU FRAMEWORK TEMPOREL PROPOSE

Le type de framework temporel que nous proposons doit comporter des bibliothèques de fonctions, à exploiter au sein d'un environnement RAD, qui dispensent le développeur de l'écriture des requêtes nécessaires à la mise à jour et à l'interrogation des faits temporels. Ces fonctions doivent permettre la génération dynamique de telles requêtes. Nous avons essayé de les recenser dans le diagramme de classes UML de la figure 1, en tant qu'opérations.

Un tel framework gagne aussi à fournir des canevas-types pour simplifier et homogénéiser la manipulation des estampilles et la navigation dans l'historique des faits.

Le diagramme de classes UML de la figure 1 modélise à la fois les méta-données (représentées par des classes de fond jaune) et les données des BDT conformément au modèle retenu. Ce diagramme montre les opérations utilisées pour la mise à jour et l'interrogation des faits temporels. Une implémentation de ce diagramme est décrite dans [Brahmia et Bouaziz (2005)].

Dans ce diagramme, toute BDT est composée de plusieurs relations. Une relation est composée d'un ou de plusieurs attributs clés, d'une relation générique et d'une ou de plusieurs versions.

Une version de relation est liée à ses éventuels attributs de base et, selon sa nature, à zéro, un ou deux estampilles. Elle est composée d'un, de deux ou de trois espaces de stockage, selon sa nature. Un espace de stockage mémorise un certain nombre de tuples dont chacun appartient à un tuple générique.

Un tuple est caractérisé par les valeurs relatives à ses attributs de base, à ses éventuelles estampilles et à ses attributs systèmes.

Un tuple est soit de type photo (TUPLE_PH), soit de type temporel de validité (TUPLE_TV), soit de type temporel de transaction (TUPLE_TT), soit enfin de type bitemporel (TUPLE_BT). Un tuple bitemporel comporte en sus une valeur pour chacun des attributs systèmes Na_erroné et Nvs_erroné.

Une relation générique mémorise des tuples génériques. Tout tuple générique est caractérisé par une valeur pour chacun des attributs clés de la relation concernée et une valeur pour chacun de ses attributs systèmes.

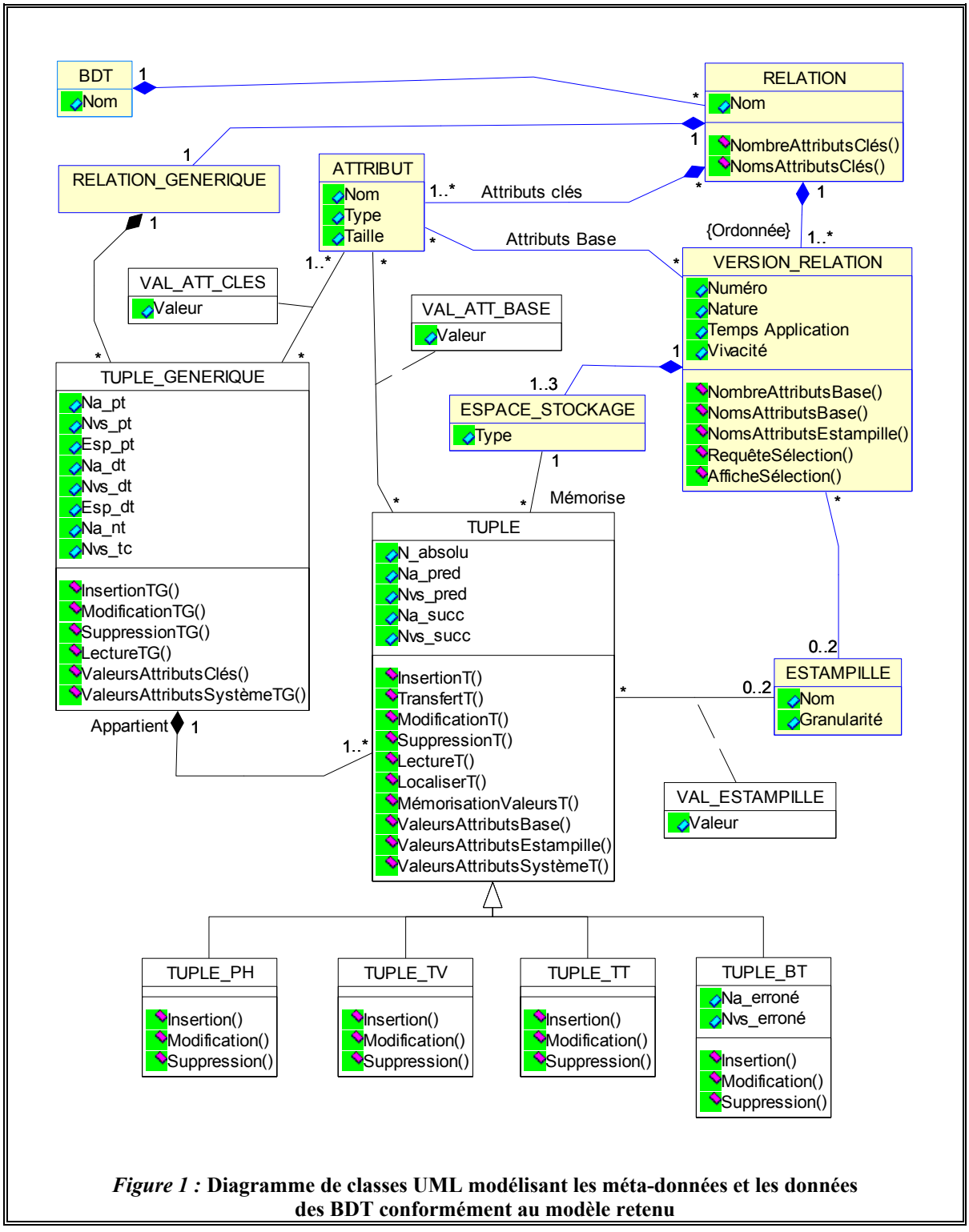

Figure 1 : **Diagramme de classes UML modélisant les méta-données et les données des BDT conformément au modèle retenu**

Le type de framework temporel que nous proposons exige une plate-forme logicielle permettant la manipulation des requêtes dynamiques. En effet, et comme nous l'avons déjà détaillé à la § 3, toute opération de mise à jour de tuples d'une BDT est assurée à travers une combinaison d'opérations (requêtes) élémentaires. La combinaison nécessaire pour toute opération ne peut être connue qu'au moment de son exécution. Par ailleurs, la ou les espaces de stockage concernés par une telle opération ou par une opération d'interrogation ne peuvent être connus qu'au moment de l'exécution.

Contrairement aux autres supports temporels de la littérature, notre framework permet d'exploiter les fonctionnalités d'un environnement RAD. Un tel environnement permet le développement de formulaires simples à exploiter par l'utilisateur final et pouvant atteindre un niveau élevé d'ergonomie. C'est ainsi qu'un tel framework propose des canevas-types, gérés par des blocs-types, pour la manipulation des valeurs des estampilles et la navigation dans l'historique des faits (cf. § 5.3).

Dans un environnement RAD, un formulaire est constitué essentiellement de blocs de données et de canevas. Le diagramme UML de la figure 2 montre un extrait du modèle de la métabase des formulaires à construire en exploitant les fonctionnalités temporelles du framework proposé.

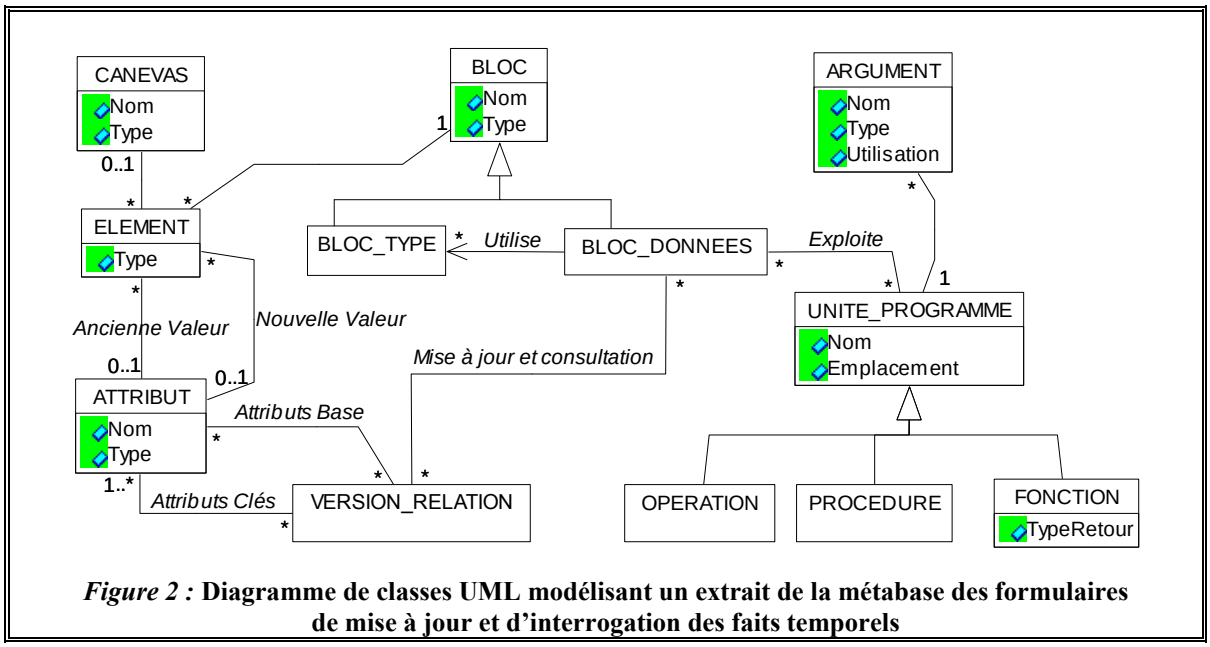

Figure 2 : **Diagramme de classes UML modélisant un extrait de la métabase des formulaires de mise à jour et d'interrogation des faits temporels**

Cet extrait de diagramme ne détaille pas les spécificités des blocs, des éléments et des canevas. Il se limite à montrer qu'un bloc de données est défini pour mettre à jour ou consulter les tuples d'une ou de plusieurs versions de relations. Un tel bloc exploite les unités de programmes et utilise des blocs-types fournis par les fonctionnalités proposées.

Un bloc peut être un bloc-type du framework ou un bloc de données ; il définit un certain nombre d'éléments de types champ de saisie ou de consultation, liste, bouton, texte, image, etc. Parmi ces éléments, on trouve ceux qui mémorisent les nouvelles valeurs et, éventuellement, les anciennes valeurs relatives aux attributs (clés et de base) de ou des versions des relations manipulées. Un élément peut être représenté dans un canevas.

Chaque unité de programmes peut avoir plusieurs arguments de différents types (entier, chaîne de caractères, date, etc.) et utilisations (argument d'entrée, de sortie ou d'entrée-sortie). Une unité de programme est soit une opération, soit une procédure, soit une fonction comportant un type de retour. Les unités de programmes correspondent à des implémentations des opérations recensées dans le diagramme de classes UML de la figure 1.

5 - PROTOTYPE D'EXPERIMENTATION

Le prototype du framework que nous avons développé étend l'outil SQL* Forms du SGBDR Oracle par des fonctionnalités temporelles [Mkaouar et Bouaziz (2003-b)]. Le choix d'Oracle se justifie par sa popularité, d'une part, et par le fait qu'il offre une plate-forme logicielle qui permet de satisfaire les spécificités du type de framework proposé, d'autre part.

5.1 - Architecture du prototype de framework

La figure 3 montre l'architecture de ce prototype, dans laquelle les fonctionnalités temporelles sont mises en relief, entre les applications et les concepteurs, d'un côté, et le SGBDR Oracle, de l'autre.

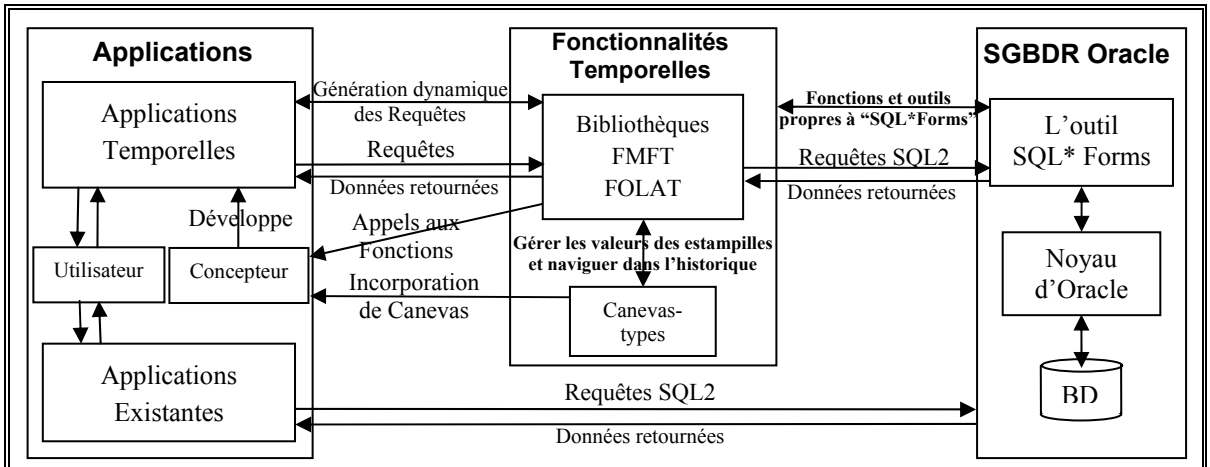

Figure 3 : **Architecture du prototype du framework**

Les fonctionnalités temporelles consistent en :

- Une bibliothèque de fonctions, intitulée FMFT (Fonctions de Manipulation dynamique des Faits Temporels), assurant les opérations de mise à jour et d'interrogation des tuples des BDT conformément au modèle retenu.
- Une bibliothèque de fonctions, intitulée FOLAT (Fonctions assurant les Opérations Logiques et Arithmétiques sur des valeurs Temporelles), assurant la manipulation des valeurs des estampilles, conformément à un format spécifique (cf. § 5.3).
- Un ensemble de canevas-types pour la manipulation des valeurs des estampilles et la navigation dans l'historique des faits.

Ces fonctionnalités sont à utiliser par le concepteur pendant la phase de développement de formulaires de manipulation d'une BDT. À cet effet, le concepteur se suffit, pour ce qui concerne le développement des traitements de mise à jour et d'interrogation des faits temporels, des appels à l'une des cinq opérations, Insertion(), Modification(), Suppression(), RequêteSélection() et AfficheSélection(). Par ailleurs, il peut procéder à des incorporations de canevas-types appropriés au sein du formulaire à construire, conformément aux exigences des interactions de ce formulaire.

Ces fonctionnalités sont utilisées au moment de l'exécution pour :

- analyser toute opération de mise à jour des faits d'une BDT afin de déterminer la combinaison des requêtes élémentaires nécessaires à l'accomplissement de cette opération ;
- générer dynamiquement le code de ces requêtes conformément à la syntaxe de SQL2, supporté par le SGBDR Oracle ;
- générer la ou les requête(s) nécessaire(s) à l'accomplissement de toute opération d'interrogation.

De plus amples informations sur le mode d'utilisation de ces fonctionnalités sont présentées par Mkaouar et Bouaziz (2003-a).

Tel que c'est montré par la figure 3, SQL* Forms ne reçoit que des requêtes SQL2, à exécuter par le noyau d'Oracle. Évidemment, les applications existantes qui utilisent directement Oracle continuent à être

opérationnelles selon cette architecture. Cependant, ces applications ne peuvent pas exploiter les fonctionnalités temporelles proposées. Les schémas des relations qu'elles utilisent doivent tout d'abord être adaptés au modèle utilisé, leurs programmes doivent respecter les spécificités des fonctionnalités temporelles.

5.2 - Bibliothèque FMFT

Cette bibliothèque est le résultat d'une implémentation non objet des opérations recensées dans le diagramme de classes UML de la figure 1, nécessaires à la mise à jour et à l'interrogation des tuples d'une BDT. Ces opérations sont écrites en PL/SQL, en tant que fonctions ayant chacune une signature appropriée. À titre d'exemples, nous détaillons dans cet article les deux fonctions relatives à l'implémentation des deux opérations InsertionTG() et Insertion() d'un tuple bitemporel. Les codes de ces deux fonctions sont fournis en annexe.

L'opération InsertionTG() est implémentée à travers une fonction qui reçoit comme paramètre d'entrée le nom du bloc de données à partir duquel elle est appelée et retourne une valeur booléenne, indiquant le succès ou l'échec de l'opération. Ce bloc mémorise, dans ses éléments, le nom de la relation concernée et toutes les valeurs des attributs nécessaires à l'insertion d'un tuple générique dans sa relation générique. La fonction InsertionTG(Bloc) assure la génération dynamique d'une requête d'insertion, conforme à la syntaxe de SQL supportée par Oracle, et à son exécution.

L'opération Insertion() est implémentée à travers une fonction qui reçoit comme paramètre d'entrée le nom du bloc de données à partir duquel elle est appelée.

Comme c'est spécifié à la § 3, l'insertion d'un tuple bitemporel nécessite la connaissance du nom de la relation R, du numéro de la version I, des nom des attributs de base renseignés {NAB}, du temps de transaction TT, des valeurs des attributs clés {VAC}, ceux des attributs estampilles de validité {VAV} et ceux des attributs de base {VAB}. La fonction Insertion(Bloc) communique au GT, à travers les éléments de son paramètre 'Bloc', toutes les valeurs relatives à ces éléments.

La fonction Insertion(Bloc) assure l'exécution de la combinaison d'opérations élémentaires nécessaires à l'insertion de T. Si, par exemple, T est le premier tuple d'un nouveau tuple générique, la fonction Insertion(Bloc) fait appel aux deux fonctions InsertionT(Relation, Bloc) et InsertionTG(Bloc), comme le montre l'extrait du code PL/SQL pour l'insertion, présenté en annexe.

5.3 - Canevas-types

Ces canevas concernent la manipulation des valeurs des estampilles et la navigation dans l'historique des faits.

Manipulation des valeurs des estampilles

Les valeurs des estampilles gagnent à être manipulées d'une manière propre à ces estampilles. Ceci permet une nette distinction entre ces valeurs de celles relatives aux attributs de type date ou temps.

Nous proposons de manipuler les intervalles temporels de validité, dont chacun est spécifié par deux instants représentant sa borne inférieure et sa borne supérieure, à travers un canevas-type de douze champs. Les six premiers concernent la borne inférieure et sont destinés à la saisie et l'affichage des valeurs de l'année, du mois, du jour, de l'heure, de la minute et de la seconde d'un intervalle. Les six autres champs sont destinés à la manipulation des valeurs correspondant à la borne supérieure de cet intervalle. Certains de ces champs peuvent être optionnels, compte tenu de la granularité choisie.

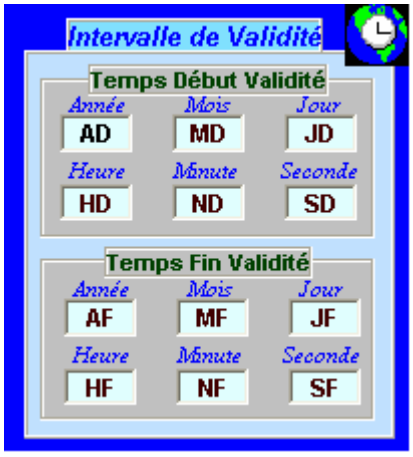

Figure 4 : **Canevas-type pour la manipulation des intervalles temporels de validité**

La figure 4 montre un tel canevas-type. L'icône : , en tant qu'une montre mesurant le temps dans le monde réel, placée dans le coin

haut droit de ce canevas-type, exprime qu'il s'agit d'un temps de validité.

Les valeurs d'un intervalle temporel de transaction, déterminées et gérées par le système, gagnent aussi à être consultés par l'utilisateur à travers un canevas-type similaire à celui de la figure 4. L'icône : , en tant qu'une montre mesurant le temps dans le système informatique, placée dans le coin haut gauche de ce canevas-type, exprime qu'il s'agit d'un temps de transaction.

D'une manière similaire, les instants de validité et les instants de transaction gagnent à être manipulés et consultés d'une manière spécifique à travers des canevas-types. De tels canevas peuvent être d'un à six champs, en fonction de la granularité choisie, de l'année à la seconde.

La bibliothèque FOLAT implémente tous les opérateurs arithmétiques, logiques et d'Allen que l'on peut appliquer sur des valeurs d'estampilles conformes à ce format.

Navigation dans l'historique

La navigation dans l'historique d'un fait temporel, mémorisé à travers les différents tuples d'un même tuple générique, permet une interprétation simple et rapide de cet historique. À cet effet, nous proposons la construction d'une palette de boutons permettant, à partir d'un tuple T consulté, de naviguer vers :

- l'éventuel tuple valide qui précède T ;
- l'éventuel tuple valide qui suit T ;
- l'éventuel tuple erroné, corrigé ou supprimé par T, dans le cas où T est un tuple bitemporel ;
- l'éventuel tuple courant relatif au tuple générique T^G de T, dans le cas où T n'est pas le tuple courant.

Figure 5 : **Palette de boutons**

La figure 5 montre une telle palette. Le code des déclencheurs de type "WHEN BUTTON PRESSED", définis pour ces boutons, se limite à un appel à l'opération LectureT(), en précisant à chaque fois le nom de l'espace, la valeur de la clé générique et le numéro absolu du tuple à lire.

5.4 - Expérimentation du prototype du framework

Ce paragraphe décrit le résultat de l'expérimentation des fonctionnalités du prototype sur le cas de l'application "Gestion du Patrimoine" d'un comité d'organisation de jeux. Un tel patrimoine est constitué de différentes catégories d'équipements. La tâche principale que doit assurer une application de gestion de ce patrimoine concerne le suivi des différentes affectations d'équipements aux membres du comité. Ceci permet de maîtriser la répartition des équipements et de préserver les bonnes relations avec les donateurs de ces équipements.

Un équipement ne peut être exploité que par un seul membre à un instant donné, mais il peut être exploité par plusieurs membres dans le temps. Afin de garder trace des différents membres bénéficiant d'un équipement, un tel fait est considéré bitemporel, devant mémoriser ainsi les intervalles temporels de validité de ses tuples dans le monde réel et les intervalles temporels de transaction où ces tuples sont considérés courants dans la BDT.

La figure 6 montre le canevas du formulaire construit pour la mise à jour des affectations des équipements aux membres du comité. Dans ce canevas, nous avons incorporé le canevas-type défini pour la manipulation des estampilles de validité.

Avec ce formulaire, l'utilisateur commence par sélectionner (à partir d'une liste) le numéro d'inventaire de l'équipement à affecter. La désignation de cet équipement s'affiche.

Si l'équipement sélectionné est déjà affecté à un membre, alors le nom, le prénom et le service de ce membre s'affichent. Il s'agit de même pour le local d'emplacement de l'équipement, le numéro de décharge et l'intervalle de validité de cette affectation. L'utilisateur peut par la suite effectuer une opération de correction, d'évolution ou de suppression de cette affectation.

Si, au contraire, l'équipement n'est pas encore affecté, l'utilisateur peut l'affecter, avec un nouveau numéro de décharge, en sélection le nom du membre concerné (à partir d'une liste) et en saisissant l'intervalle de validité de l'affectation.

Un clic sur le bouton “Valider” du canevas permet d’assurer la validation de l’opération effectuée.

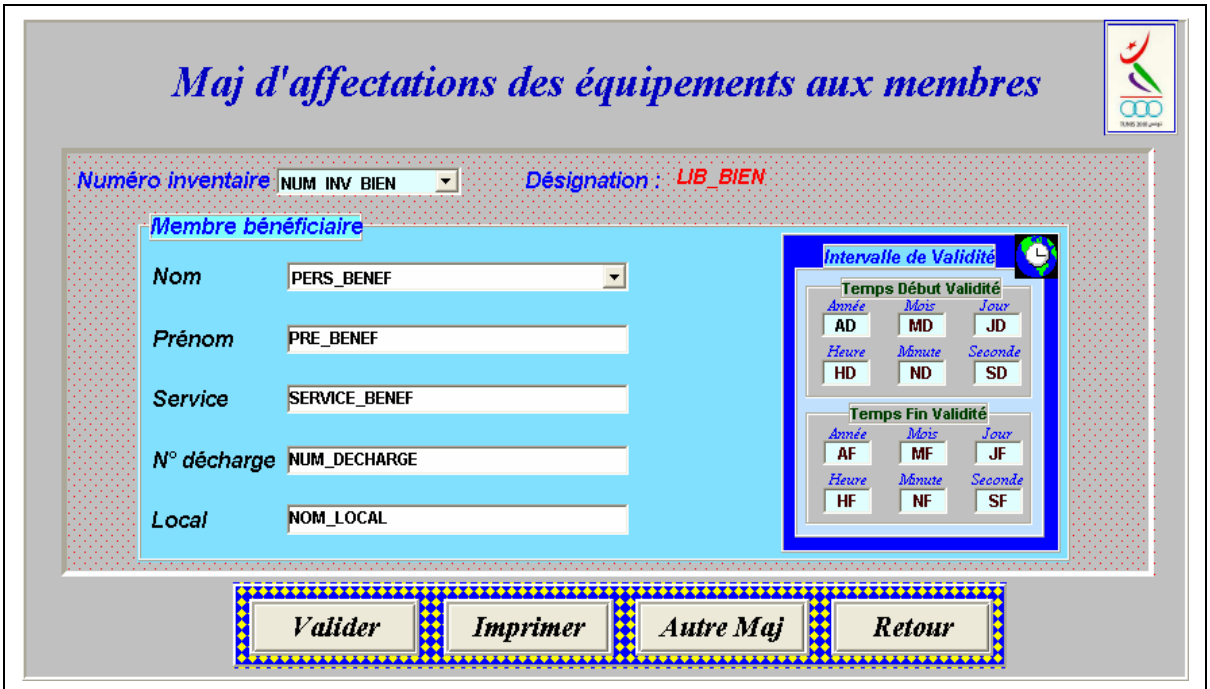

Figure 6 : **Canevas du formulaire de mise à jour d’affectations des équipements aux membres**

L’interrogation des différents membres bénéficiant d’un équipement est assurée par un formulaire dont le canevas est montré en figure 7. Dans ce canevas, nous avons incorporé le canevas-type que nous avons défini pour la navigation dans l’historique d’un fait.

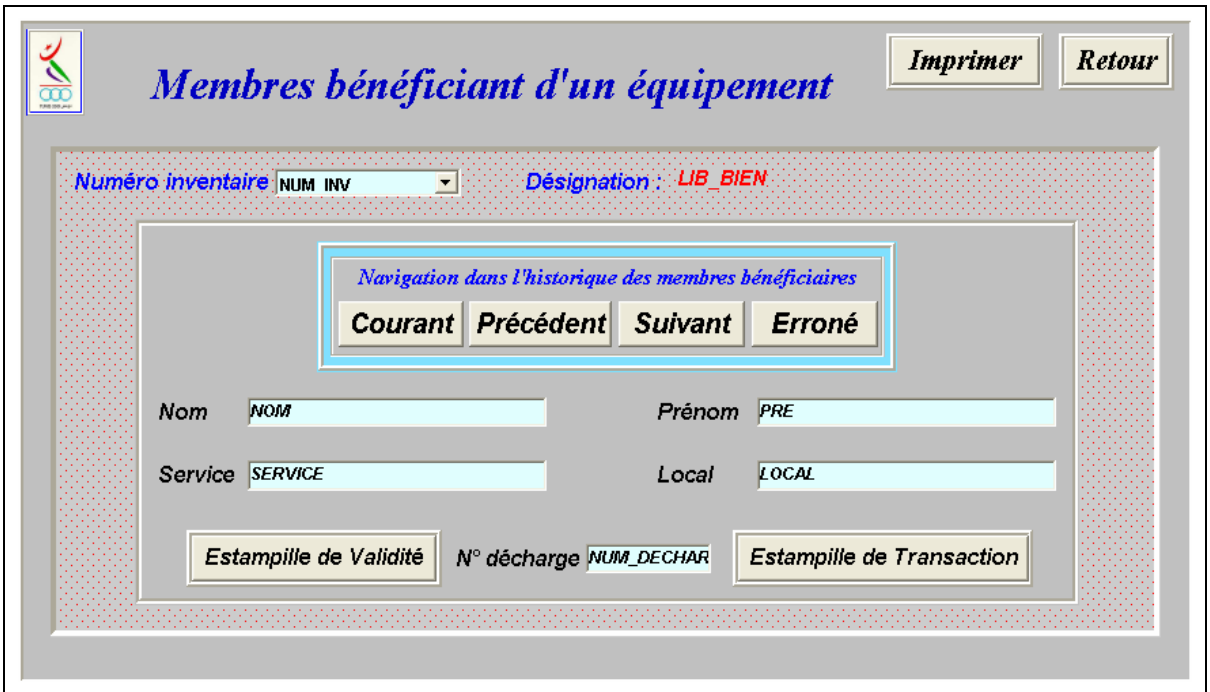

Figure 7 : **Canevas du formulaire d’interrogation de membres bénéficiant d’un équipement**

Avec ce formulaire, l’utilisateur commence par sélectionner le numéro d’inventaire de l’équipement dont les membres bénéficiaires sont à consulter. Le libellé de cet équipement

et les informations relatives à l'éventuelle affectation courante (nom, prénom et service du membre, emplacement de l'équipement et numéro de décharge) sont affichés. Les valeurs de l'intervalle temporel de validité et de l'intervalle temporel de transaction sont affichées, sur demande, en cliquant respectivement sur le bouton "Estampille de Validité" et sur le bouton "Estampille de Transaction".

L'utilisateur peut naviguer dans l'historique des membres bénéficiaires à l'aide des boutons "Précédent" et "Suivant". Le bouton "Courant" permet de retourner directement à l'affectation courante. Le bouton "Erroné" permet d'afficher les éventuelles informations erronées d'une affectation qui a été corrigée, réduite ou supprimée.

6 – CONCLUSION

Plusieurs travaux ont déjà montré la possibilité d'étendre les SGBDR commercialisés pour la prise en compte des faits temporels. Mais la plupart de ces travaux procèdent à l'implémentation et la manipulation de ces faits à travers l'écriture directe de requêtes temporelles. Ceci exige du développeur une connaissance approfondie du modèle temporel utilisé et de la stratégie de mise à jour de tels faits, d'une part, et une syntaxe complexe permettant difficilement de satisfaire les sémantiques requises, d'autre part.

Dans ce article, nous avons proposé un type de framework qui définit des fonctionnalités temporelles s'adressant aux développeurs et qui les dispensent de l'écriture de requêtes pour la mise à jour et l'interrogation des faits temporels. Ces fonctionnalités les déchargent ainsi de la manipulation directe et délicate des structures d'implémentation de ces faits et des techniques utilisées pour les stocker, relier, modifier, supprimer et interroger. Elles sont à exploiter au sein d'un environnement de développement rapide d'applications, permettant de bénéficier des apports de la programmation événementielle et de la convivialité des interfaces graphiques évoluées. Un tel environnement permet aussi d'offrir des fonctionnalités pour une manipulation appropriée des valeurs des estampilles et une interprétation simple de l'historique à travers une interrogation par navigation.

Ces fonctionnalités offrent la possibilité de construire rapidement des formulaires de mise à jour et d'interrogation des faits temporels. De tels formulaires permettent à l'utilisateur final d'alimenter et d'exploiter les BDT d'une manière confortable et aisée.

Le prototype d'expérimentation intègre ces fonctionnalités au-dessus de SQL* Forms d'Oracle. Ce prototype a prouvé son utilité suite au développement d'applications concrètes.

Nous considérons donc que l'objectif visé, consistant à montrer la faisabilité de l'édification d'un framework qui dispense le développeur de l'écriture de requêtes de manipulation des faits temporels, est atteint. Cependant, il nous reste à compléter les dites fonctionnalités pour assurer la génération de requêtes d'interrogation complexes, même si la normalisation temporelle [Navathe et Ahmed (1987)] et le recours à des vues permettent de réduire considérablement le besoin à de telles requêtes.

Par ailleurs, il nous reste à mesurer les performances de l'exécution. L'amélioration de ces performances passe par la définition de nouvelles techniques d'indexation et de nouveaux algorithmes d'optimisation des requêtes.

Nous prévoyons également l'intégration d'une panoplie d'outils que nous avons définis pour gérer les BDT :

- fonctionnalités de manipulation des faits temporels,
- fonctionnalités de versionnement des schémas [Bouaziz et Moalla (1996), Brahmia et Bouaziz (2005)],
- expression des contraintes d'intégrité temporelles,
- redressement, automatique ou assisté, des erreurs générées par l'utilisation des données erronées, que l'on vient de corriger, par les traitements de l'applicatif [Bouaziz et Moalla (1998)],
- algorithmes de contrôle de concurrence d'accès [Elloumi, Bouaziz et Moalla (1998), Bouaziz et Makni (2005)].

Remerciements

Les auteurs tiennent à exprimer leurs vifs remerciements au Professeur Mohamed MOALLA pour son aide et sa contribution au niveau du développement du prototype et de son expérimentation.

Dédicace

Le premier auteur dédie ce travail à sa mère Najet et espère que son âme sainte trouve dans ce travail le fruit de ses véritables sacrifices.

BIBLIOGRAPHIE

Bernstein, P.A., Hadzilacos, V., Goodman N. (1987), *Concurrency control and recovery in DBS*, Addison-Wesley.

Böhlen, M.H., Busatto, R., Jensen, C.S. (1998), "Point-based versus interval-based temporal data models.", In *Proceedings of the 4nd International Conference on Data Engineering*, pp192-200.

Bouaziz, R., Makni, A. (2005), "ACCO_CF/RTT : Un Algorithme de Contrôle de Concurrence Optimiste pour les Relations Temporelles de Transaction.", *ISDM*, vol 19.

Bouaziz, R., Moalla, M., Rolland, C. (1992), "Approche globale pour la gestion de l'historisation dans les bases de données temporelles.", *Congrès INFORSID*. Clermont Ferrand, France.

Bouaziz, R., Moalla, M. (1996), "Gestion des versions des schémas dans les bases de données temporelles.", *Forum de Recherche en Informatique (FRI)*, Tunis, Tunisie.

Bouaziz, R., Moalla, M. (1998), "Historisation des Données et Redressement des Effets de Bord.", Actes des 14èmes Journées *Bases de Données Avancées (BDA'98)*, Tunisie.

Bowman, I.T., Toman, D. (2001), "Optimizing Temporal Queries: Efficient Handling of Duplicates.", In *Eighth International Symposium on Temporal Representation and Reasoning (TIME'01)*.

Brahmia, Z., Bouaziz, R. (2005), "Problématique du versionnement des schémas dans les BDT Relationnelles.", 5èmes Journées des jeunes chercheurs en Génie Électrique et Informatique : *GEI'05*, Sousse, Tunisie.

Chomicki, J. (1994), "A Temporal Query Languages: A Survey". In *Proceeding of the First International Conference on Temporal Logic*, Bonn, Germany, pp506-534.

Eisenberg, A., Melton, J. (2000), "SQL Standardization: the next steps.", *ACM SIGMOD Record*, vol 29, n° 1.

Elloumi Dhouib, S., Bouaziz, R., Moalla, M. (1998), "Contrôle de Concurrence multiversion dans les bases de données temporelles.", Actes des 14èmes Journées *Bases de Données Avancées (BDA'98)*, Tunisie.

Gao, D., Snodgrass, R.T. (2003), "Syntax, Semantics, and Query Evaluation in the τXQuery Temporal XML Query Language.", *Technical Report TR-72*, TIMECENTER.

Grandi, F. (2003), "An Annotated Bibliography on Temporal and Evolution Aspects in the WorldWideWeb.", *Technical Report TR-75*, TIMECENTER.

Jensen, C.S., Dyreson, C.E. (1998), "the Consensus Glossary of Temporal Database Concepts.", In Etzion, O., Jajodia, S., Sripada, S.M., *Temporal Databases: Research and Practice*, LNCS 1399, Spring Verlag.

Jensen, C.S., Snodgrass, R.T. (1997), "Temporal Data Management.", *Technical Report TR-17*, TIMECENTER.

Matus-Castillejos, A., Jentzsch, R. (2005), "A time series data management framework.", *International Conference on Information Technology: Coding and Computing, ITCC*, Vol 1, pp220-225.

Mkaouar, M. (1999), "Conception d'un système d'information sous un environnement de bases de données temporelles multiversions de schémas.", *Mémoire de DEA*, Faculté des Sciences de Tunis.

Mkaouar, M., Bouaziz, R. (2003-a), "A Support toolset for the development in a temporal database environment.", *Arab International Conference on Computer Systems and Applications (AICCSA'03)*, Tunis, Tunisie.

Mkaouar, M., Bouaziz, R. (2003-b), "Outils et fonctionnalités d'aide au développement de Bases de Données Temporelles sous

Oracle.", 19èmes Journées *Bases de Données Avancées (BDA'03)*, France.

Navathe, S.B., Ahmed, R. (1987), "TSQL: A Langage Interface for History Database.", In *Proceedings of the Conference on Temporal Aspects in Information Systems*, AFCET, pp113-128.

Nobecourt, P., Rolland, C., Lingat, Y. (1988), "Temporal management in an extended relational system.", *BNCOD6 Conférence*, Grande Bretagne.

Noh, S.Y., Gadia, S.K. (2005), "An XML-Based Framework for Temporal Database Implementation.", 12th *International Symposium on Temporal Representation and Reasoning (TIME'05)*.

Sethuraman, M. (2003), "Implementation and Evaluation of a Partitioned Store for Transaction-Time Databases.", *Technical Report TR-76*, TIMECENTER.

Skjellang, B. (1997-a), "Temporal Data: Time and Object Databases.", University of Oslo, Department of Informatics, *Research Report* 245, ISBN 82-7368-160-2.

Skjellang, B. (1997-b), "Temporal Data: Time and Relational Databases.", University of Oslo, Department of Informatics, *Research Report* 246, ISBN 82-7368-161-0.

Slivinskas, G., Jensen, C.S., Snodgrass, R.T. (2001), "Adaptable Query Optimization and Evaluation in Temporal Middleware.", *Technical Report TR-56*, TIMECENTER.

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

Snodgrass, R.T. (2000), *Developing time-oriented database applications in SQL*, Morgan Kaufmann Publishers.

Steiner, A. (2005), TimeDB 2.2, A TimeConsult Product, www.TimeConsult.com.

Toman, D. (2000), "SQL/TP: a Temporal Extension of SQL.", In Kuper, G., Libkin, L., Paredaems, J., *Constraint Databases*.

Torp, K., Jensen, C.S., Snodgrass, R.T. (1997), "Stratum Approaches to Temporal DBMS Implementation.", *Technical Report TR-5*, TIMECENTER.

Tsotras, V.J., Wang, X.S. (1999), "Temporal Databases.", *Encyclopedia of Electrical and Electronics Engineering*, John Wiley and Sons.

Wu, Y., Jajodia, S., Wang, X.S. (1998), "Temporal Database Bibliography Update.", Etzion, O., Jajodia, S., Sripada, S.M., *Temporal Databases: Research and Practice*, LNCS 1399, Spring Verlag.

Yang, J., Widom, J. (2003), "Incremental Computation and Maintenance of Temporal Aggregates.", *The VLDB Journal*, vol 12, n° 3, pp262-283.

Yang, J., Ying, H.C., Widom, J. (2000), "TIP: A Temporal Extension to Informix.", *ACM SIGMOD Record*, vol 29, n° 2.

ANNEXES : CODE PL/SQL DE QUELQUES FONCTIONS

A.1 - Code PL/SQL de la fonction InsertionTG(Bloc)

```
FUNCTION InsertionTG (Bloc in varchar2)
RETURN boolean
IS
   requete          varchar2(2000);
BEGIN
   requete:='INSERT INTO VG_'||upper(:Bloc.relation);
   requete := requete||'
   ('||upper(:Bloc.relation).NomsAttributsClés();
   requete := requete||'NA_PT, NVS_PT, ESP_PT,
   NA_DT, NVS_DT, ESP_DT, NA_NT, NVS_TC)
   VALUES(';
   requete:=requete||ValeursAttributsClés(Bloc);
   requete:=requete||:Bloc.na_pt||', '||:Bloc.nvs_pt||',
   '''||:Bloc.esp_pt||'''||', '||:Bloc.na_dt||',
   '||:Bloc.nvs_dt||', '''||:Bloc.esp_dt||'''||',
   '||:Bloc.na_nt||', '||:Bloc.nvs_tc||')';
   FORMS_DDL(requete);
   if not form_success
   then
      :Message := 'Insertion du tuple générique échouée :
      Code de l'erreur :'||sqlcode|| 'Raison :'||sqlerrm;
   end if;
   return(form_success);
END;
```

Code PL/SQL de la fonction InsertionTG

La fonction InsertionTG(Bloc) fait appel à la fonction NomsAttributsClés() qui retourne la liste des noms des attributs clés de la relation manipulée, et à la fonction ValeursAttributsClés(Bloc) qui retourne les valeurs de ces attributs, mémorisées dans des éléments du bloc passé en argument. La variable ':Message' utilisée par la fonction InsertionTG(Bloc) mémorise la raison de l'éventuel échec de l'exécution de la requête d'insertion.

A.2 - Extrait du code PL/SQL : Insertion d'un nouveau tuple générique

```
FUNCTION INSERTION (Bloc in varchar2)
IS
   relation          varchar2(50);
   resultat          boolean := true;
   test              boolean;
BEGIN
   ValeursAttributsSystèmeTG(Bloc); --Valorisation des
   attributs du tuple générique de T
   LocaliserT(Bloc); --Valorisation des attributs système
   du tuple prédécesseur et du tuple successeur de T
   ValeursAttributsSystèmeT(Bloc); --Valorisation des
   attributs système de T
   if (:Bloc.n_absolu = 1)
   --Il s'agit du premier tuple d'un nouveau tuple
     générique
   then
     --Confection et exécution de la requête d'insertion
       de T
     relation := 'V'||:Bloc.Numéro||'_V'||:Bloc.Espace||'_'
     ||:Bloc.relation
     --Nomination des espaces courant, dépassé et,
       éventuellement, futur, de toute version I d'une
       relation temporelle R, respectivement par
       VI_VC_R, VI_VP_R et VI_VF_R.
     test:= InsertionT (relation, Bloc);
     if not test
     then
       set_alert_property('message', alert_message_text,
       :Message);
       al_button := show_alert('message');
     end if;
     --Confection et exécution de la requête d'insertion
       de T^G
     test:= InsertionTG (Bloc)
     if not test
     then
       set_alert_property('message', alert_message_text,
       :Message);
       al_button := show_alert('message');
     end if;
   else
     --Traitement du cas où T^G existe
      ¦
   end if;
   return(resultat);
END;
```

Extrait du code PL/SQL pour l'insertion : cas d'un nouveau tuple générique

{VAC}, {VAV}, {VAB} et {NAB} sont déterminés, au sein de la fonction InsertionT(Relation, Bloc), respectivement à l'aide des fonctions :

- ValeursAttributsClés(Bloc),
- ValeursAttributsEstampilles(Bloc),
- ValeursAttributsBase(Bloc) et
- NomsAttributsBase(Relation).